****

**Orchestration, SCXML and Composer**

Version 1.0

Document Status: Working Draft

Document Owner: Dwight & Vasu

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Ed. | Date | Author(s) | Change |
| 1.0 | 04/01/2016 | Vasu, Dwight | Final Doc |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Content

Introduction 24

Pre Training 24

Training Installation 25

Installation of ORS 25

Installation of URS 26

Installation of Composer 28

Resources Used during the training 28

Post Training Course Syndrome 29

Project 1 Hello world project 31

Select a workspace 31

Composer Interface 31

Creating the new Project 32

Project Language 32

Project Type 33

Project Templates 34

Project Locales 34

Making the project changes 36

Clear the Wait for Event within the IPD 36

Creating the Hello World Structure 38

Saving and Generating the SCXML Code 39

Checking that we have the Project Deployed 41

Checking the application further 43

Starting an SCXML Session 44

ORS Web Services Interface 44

WizTools RESTClient 45

Hello World 46

ORS Session and Events 46

Execute again with the change. 47

Querying the running session 48

Sending in event we are waiting on 50

Multiple Events 51

Introducing the debugger 52

Configuring the debugger 52

Start a debug session - Creating the debug configuration 53

Starting the session 54

Modify the debugged flow 56

Project 2 - Interacting with the ORS Session 62

Create the basic project structure 62

A Simple State Machine 62

Introduce the state block 63

Building the Traffic Light State Machine 63

Procedural State Model 64

Getting the lights to cycle 64

Allowing the Traffic Light to Terminate 65

What a MESS! 66

Hierarchical States 67

States and Sub States 67

Outline of a Generated IPD 67

Taking advantage of the Hierarchical States 68

Adding a Transition Handler on Enter 69

Debug the flow 71

Handing a deploy issue error message 72

Hierarchical event order. 73

Running the debug with the terminate event 75

Transition Conditions 75

Set up the Rest request 75

Build the condition on the GREEN state 76

Why the Error. 78

Let’s correct the error and continue. 79

Starting the application again 80

Why the need for <script>? 81

Sending in the terminate event with data 81

Student Exercise 82

Multiple Transitions with the same event name 82

Adding a target-less transition 82

Evaluating the new terminate event 83

Event Wild Cards 84

Add additional Transitions on GREEN state 85

Testing with event wild cards 86

Composer Global Handler - Or why your calls just stop 87

Faking an error 87

The Composer Global Error Handler 87

Passing Data on start 88

Add a Project Level variable called Color 89

Passing Initialization Data on start of the application to the <datamodel> 89

Inspecting the <datamodel> 90

Student EXERCISE: - Add a branch based on start up color 90

List of Figures

[Figure 1: Workspace Selection 21](#_Toc479611022)

[Figure 2: Composer Interface 22](#_Toc479611023)

[Figure 3: Select Project Type 23](#_Toc479611024)

[Figure 4: Project Details 23](#_Toc479611025)

[Figure 5: Project Templates 24](#_Toc479611026)

[Figure 6: Project Locales 25](#_Toc479611027)

[Figure 7: Change Composer Perspective 25](#_Toc479611028)

[Figure 8: Change from Composer Design to Composer Perspective 26](#_Toc479611029)

[Figure 9: IPD Wait For Event Clearing 27](#_Toc479611030)

[Figure 10: Wait For Event Dialog 27](#_Toc479611031)

[Figure 11: Wait For Event Dialog with No Events 28](#_Toc479611032)

[Figure 12: Hello World Workflow 28](#_Toc479611033)

[Figure 13: Hello World Block Properties 29](#_Toc479611034)

[Figure 14: Unsaved Changes 29](#_Toc479611035)

[Figure 15: Code Generation Dialog 30](#_Toc479611036)

[Figure 16: Running Code Generation Dialog 30](#_Toc479611037)

[Figure 17: Success Code Generation 30](#_Toc479611038)

[Figure 18: Composer Tomcat Deployment Check 31](#_Toc479611039)

[Figure 19: Apache Tomcat Confirmation page 32](#_Toc479611040)

[Figure 20: Tomcat Web Application Manager 32](#_Toc479611041)

[Figure 21: Browser retrieval of SCXML Code 33](#_Toc479611042)

[Figure 22: SCXML file not found 34](#_Toc479611043)

[Figure 23: Composer Deploy button 34](#_Toc479611044)

[Figure 24: RestClient Interface 35](#_Toc479611045)

[Figure 25: IPD Wait For events 36](#_Toc479611046)

[Figure 26: Adding a Custom Event 37](#_Toc479611047)

[Figure 27: Waiting for Event in Log file 37](#_Toc479611048)

[Figure 28: Pretty print of JSON from jsonviewer.stack.hu 39](#_Toc479611049)

[Figure 29: Event Receipt in Logs 40](#_Toc479611050)

[Figure 30: Configuring Multiple Events 41](#_Toc479611051)

[Figure 31: Debugger ORS Configuration 42](#_Toc479611052)

[Figure 32: Composer ORS Debugger Configuration 42](#_Toc479611053)

[Figure 33: Debug Configuration for Workflow 43](#_Toc479611054)

[Figure 34: Debug Launch Configuration 44](#_Toc479611055)

[Figure 35: Composer Metric View 45](#_Toc479611056)

[Figure 36: Break point on Entry 45](#_Toc479611057)

[Figure 37: Composer signifies our position 46](#_Toc479611058)

[Figure 38: Workflow Variables access from Toolbar 46](#_Toc479611059)

[Figure 39: Application Variables 47](#_Toc479611060)

[Figure 40: Advanced Hello World 48](#_Toc479611061)

[Figure 41: Branching Conditions 48](#_Toc479611062)

[Figure 42: Multiple Exits in one diagram 48](#_Toc479611063)

[Figure 43: Debugger Console 49](#_Toc479611064)

[Figure 44: Wrong Path taken 49](#_Toc479611065)

[Figure 45: Using Console to inspect variables 49](#_Toc479611066)

[Figure 46: Viewing Metrics 50](#_Toc479611067)

[Figure 47: Add watch expression 50](#_Toc479611068)

[Figure 48: Current state of watch expression 50](#_Toc479611069)

[Figure 49: Modifying value from watch expression 51](#_Toc479611070)

[Figure 50: Observing the modified behavior 51](#_Toc479611071)

[Figure 51: The Traffic Light State Machine 52](#_Toc479611072)

[Figure 52: The SCXML State Block 53](#_Toc479611073)

[Figure 53: SCXML Transitions common dialog box 53](#_Toc479611074)

[Figure 54: Procedural Traffic Light State Machine 54](#_Toc479611075)

[Figure 55: Incomplete loop 55](#_Toc479611076)

[Figure 56: Handling Terminate at all possible locations 56](#_Toc479611077)

[Figure 57: Hieratical State Machine 57](#_Toc479611078)

[Figure 58: Code Outline of IPD 58](#_Toc479611079)

[Figure 59: Rebuilding the state machine 59](#_Toc479611080)

[Figure 60: Adding Transition 60](#_Toc479611081)

[Figure 61: Adding Transition on Entry Block 60](#_Toc479611082)

[Figure 62: Updated Flow with hierarchical state event handling 61](#_Toc479611083)

[Figure 63: Project 2 debug configuration 62](#_Toc479611084)

[Figure 64: Project 2 ORS Debugger Launch Information 62](#_Toc479611085)

[Figure 65: Debugger Deploy issue detected 63](#_Toc479611086)

[Figure 66: Terminate Handler 64](#_Toc479611087)

[Figure 67: Green Terminate Handler 65](#_Toc479611088)

[Figure 68: Green Terminate Metrics 65](#_Toc479611089)

[Figure 69: Set content type and Charset 66](#_Toc479611090)

[Figure 70: Set Body of message to include the property COLOR 66](#_Toc479611091)

[Figure 71: Green Terminate Transition 67](#_Toc479611092)

[Figure 72: HTTP 400 Error response 68](#_Toc479611093)

[Figure 73: Platform Metrics indicating error 68](#_Toc479611094)

[Figure 74: Composer's file error indicator 69](#_Toc479611095)

[Figure 75: SCXML file error display 69](#_Toc479611096)

[Figure 76: Corrected code 70](#_Toc479611097)

[Figure 77: OK file syntac 70](#_Toc479611098)

[Figure 78: Terminate Event Handling 70](#_Toc479611099)

[Figure 79: Top level Terminate taken 71](#_Toc479611100)

[Figure 80: Lowest level Terminate taken 71](#_Toc479611101)

[Figure 81: View of the \_event in JSON 72](#_Toc479611102)

[Figure 82: Additional Criteria 72](#_Toc479611103)

[Figure 83: Target-less Transition 73](#_Toc479611104)

[Figure 84: Document Order Transition Evaluation 73](#_Toc479611105)

[Figure 85: Terminate State transitions 74](#_Toc479611106)

[Figure 86: Modification of terminate event 75](#_Toc479611107)

[Figure 87: Hierarchical Transition Handlers with Wild Card 76](#_Toc479611108)

[Figure 88: terminate.green event handling 76](#_Toc479611109)

[Figure 89: terminate.black event handling 77](#_Toc479611110)

[Figure 90: Composer Global Error Handling 77](#_Toc479611111)

[Figure 91: Global Error Handling Code 78](#_Toc479611112)

[Figure 92: Project Variables 79](#_Toc479611113)

[Figure 93: Passing Data on Start 80](#_Toc479611114)

[Figure 94: Console view of <datamodel> 80](#_Toc479611115)

List of Tables

[Table 1: Definitions & Acronyms 11](#_Toc479611116)

[Table 2: Terminology 12](#_Toc479611117)

[Table 3: ORS Install Wizard Screen 15](#_Toc479611118)

[Table 4: URS Install Wizard Screen 17](#_Toc479611119)

[Table 5: Required Configuration Items 18](#_Toc479611120)

[Table 6: Additional Resources 19](#_Toc479611121)

References

|  |  |  |
| --- | --- | --- |
| N | Document Title | File Name |
|  | Composer Deployment Guide |  |
|  |  |  |
|  |  |  |

Acronyms

|  |  |
| --- | --- |
| Acronym | Definition |
| ERS | Enhanced Routing Script |
| FM | Functional Module |
| IXN | Interaction |
| ORS | Orchestration Server |
| SCXML | State Chart XML |
| TM | Technical Marketing |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Table 1: Definitions & Acronyms

Terminology

|  |  |
| --- | --- |
| Term | Definition |
| Block | A block is equivalent to an IRD object and is used to graphically build workflows. |
| Composer | This is an Eclipse based IDE that allows users to build SCXML either visually or via the SCXML text editor capabilities |
| Perspective | This refers to the view that you are seeing within Composer. Different perspectives are opened to accomplish different tasks in a more task orientated manner. |
| TM Image | Short hand used to imply Technical Marketing Vmware Image |
|  |  |

Table 2: Terminology

# Introduction

Welcome to the world of Orchestration and Composer! The aim of this document is to take you on a self-guided tour of ORS and Composer and explore the power of ORS. This is a living document and as such we will likely be adding additional topics as we explore the needs of additional training.

The intent for this document is to be an internal facing document and shall not be delivered or extended to customers, who should be steered towards recognized Genesys Training from GU. As such the information in this material is provided 'as-is' with no expressed warranty.

In the current version of this document we are limiting our focus on to what we have seen often as the learning curve namely ORS and Composer. We will be focusing on Composer features exposed by ORS, especially on fundamental building blocks that can be used to build world class customer experience applications.

This information is accompanied with full project files and restful requests files, although we would hope you don't cheat and look ahead at them before trying to build the projects yourself.

All of these documents will be available on Alfresco site for your reference.

<https://alfresco.genesys.com/share/page/site/orchestration-central/documentlibrary>

# Pre Training

Before embarking on the projects within this document the following has been assumed to be at hand

1. The latest Technical Marketing(TM) Image
2. A computer (Laptop) that is able to run the TM image adequately

As for student skill sets we have not assumed anything specific other than a general understanding of the Genesys Product Suite and Internet Technologies such as HTTP, XML and JSON as well as being familiar with VMWare Workstation

## Training Installation

It is recommended that you make a snapshot of the TM image.

During this training since we have the basic installation undertaken for us within the Technical Marketing Image we will not be addressing the topics of installation of the various components other than those required to complete the training and that are not shipped on the current TM Image.

Also, we encourage you to install latest version of ORS and Composer version before you start building applications. Installation instructions are as stated below and can also be followed through using docs.genesys.com documents.

### Installation of ORS

The following table outlines the responses for the ORS installation that should be given. No changes to Cassandra are required.

|  |  |
| --- | --- |
| Installation Screen | |
| Step 1 | Step 5 |
| Step 2 | Step 6 |
| Step 3 | Step 7 |
| Step 4 | Step 8 |

Table 3: ORS Install Wizard Screen

### Installation of URS

The version of URS used for this training is compatible with the current version installed on the TM Image. As such the URS version indicated may be upgraded directly.

|  |  |
| --- | --- |
| Installation Screen | |
| Step 1 | Step 5 |
| Step 2 | Step 6 |
| Step 3 | Step 7 |
| Step 4 | Step 8 and 9 |

Table 4: URS Install Wizard Screen

### Installation of Composer

## Resources Used during the training

The various projects described where possible will leverage existing resources on the TM Image. However, the following resources have been identified to be required and so should be configured or validated. Additional configuration required on a per project basis will be described within each project.

|  |  |
| --- | --- |
| Resource | Details |
| Agent | Ksippo |
| Skill | Sales |
| Route Point | 5555 |
| Route Point | 5556  Should be configured as Route point 5555 but with 5556 used as the number and within the alias |
| Orchestration | Ensure that the verbose option is set to all |

Table 5: Required Configuration Items

Tip: - It may also be beneficial for some of these projects to also create a solution consisting of JUST Orchestration and Cassandra. This may help with memory and speed of your system.

## Post Training Course Syndrome

We understand that as you may receive a lot of information during the training you may get a little bit too use to this level of information, which may lead to 'Post Training Course Syndrome', common when you become addicted to ORS, SCXML and Composer and crave more information. Well we want to help you come down from the training high but in a gradual way. That's why we have the following resources for you to help you get your fix and to engage with the people that have brought you Orchestration and Composer, together with your peers (Who may also suffer from PTCS).

|  |  |
| --- | --- |
| Support Groups for PTCS ! | Description |
| ORSCentral.png  <https://alfresco.genesys.com/share/page/site/orchestration-central/documentlibrary> | Orchestration Central on alfresco will provide you the latest documents on different customer projects, Composer applications, ORS and SCXML related documents. |
| Orchestration Wiki  <http://docs.genesyslab.com/wiki/index.php?title=Orchestration_Server> | Details about deployment and also the core ORS SCXML references together with samples |
| Orchestration Wiki (Internal Site)  <http://internalwiki.us.int.genesyslab.com/draft_docs/index.php/Orchestration_Server> | This site is the living document that is not yet pushed to the outside world. |

Table 6: Additional Resources

# Project 1 Hello world project

All good software courses typically start with that “Hello World” project well we are no different. In this project the first of hopefully many successful ones we will simply create Composer and ORS's version of the Hello World Project.

## Select a workspace

Having started up Composer, you will often be required to select a workspace unless you have selected previously 'do not ask this again'. This is something that you typically only need to do once on startup of composer.

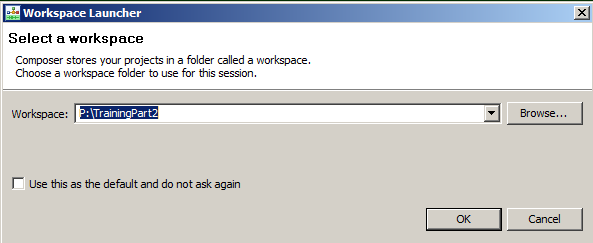


Figure 1: Workspace Selection

The project workspace will contain all related projects for you. it is recommended that you should use workspaces to group related projects together. Also its not required to place such a workspace in the same installation folder as Composer so the workspace can reside anywhere. You may find it handy to place it on the root of your C drive and so can use the browse button for that purpose. Also you may benefit from placing a short cut to the workspace on your desktop.

## Composer Interface

If starting Composer for the first time with a new workspace you will be met with something like this or the installation screen if it’s the first time ever starting Composer. For more information on this process please refer to the Composer deployment guides.



Figure 2: Composer Interface

## Creating the new Project

A project is a collection of files that provides your application logic. Within Composer since it bridges bother GVP and ORS you have the choice of creating projects that support both. So let's first of all select new Project.

### Project Language

Composer can support creating two types of project either a Java composer project or a .Net project. For the TM image **since we have Tomcat preinstalled and configured for Composer we will only be working with Java Projects**. The key difference between them is in the application server side files that are bundled with the projects, the generated SCXML code should be identical regardless of the project type since ORS only knows about SCXML and JavaScript.

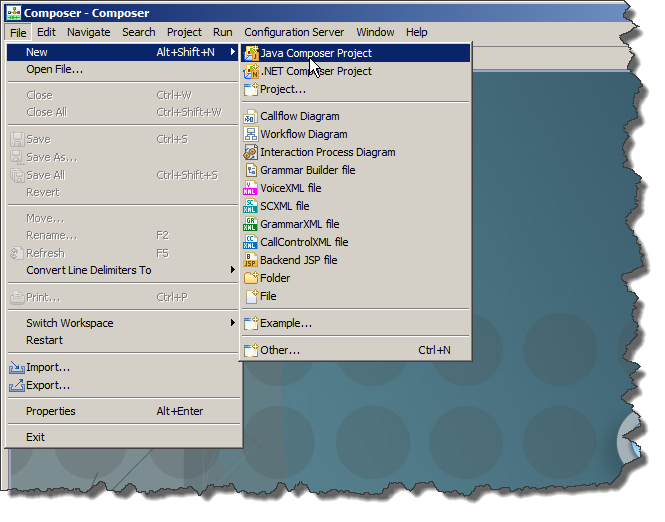


Figure : Select Project Type

### Project Type

After selecting the project language type you are then asked to provide details for the project such as the project name and also the type of composer project that you want to create.

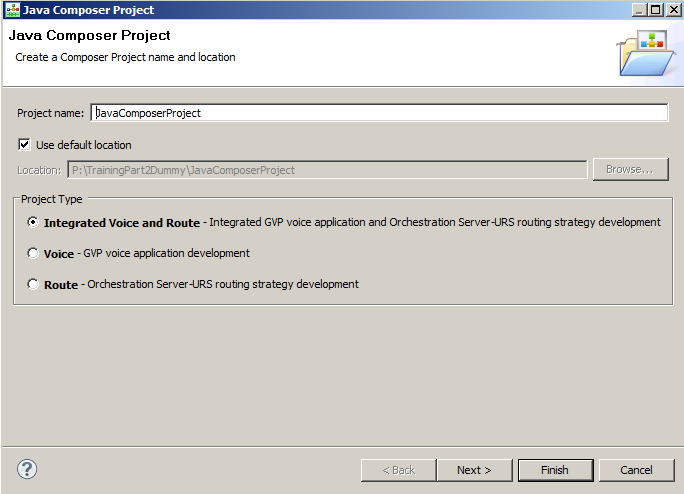


Figure : Project Details

It is best not to use any spaces within a project name. For most of these projects we will select Integrated Voice and Route which allows us to create a single project that can support GVP call flows and ORS workflows. If you are just working with GVP (Which is not covered in much detail in the rest of this document) then select Voice and if it is just routing, then select Route.

### Project Templates

Composer ships with some preselected templates that can help as a basis to get you started for all of the projects in this document we will always be starting from scratch. So simply select next on the previous dialog box and select the Blank Template from the dialog shown.

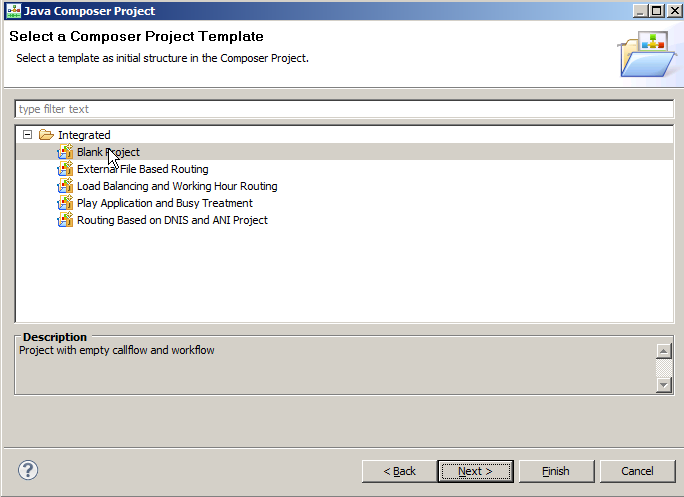


Figure 5: Project Templates

The project templates shown will be based on the type of Project you have selected. That is - Route, Voice and Integrated all have different templates.

### Project Locales

If you have selected an integrated project type, then the following dialog box may be presented this is required so you can support different locales for the various prompts and grammars. You can simply press next on this to skip passed this dialog.

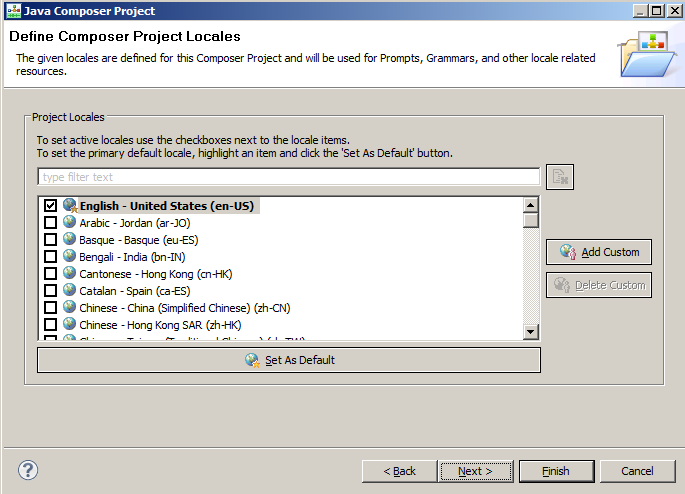


Figure 6: Project Locales

At this point, you may simply select finish to skip additional configuration. Once completed you may still be presented with the default perspective in Composer and so you should open the Composer perspective simply click on the following to open the perspective where you should be treated to the view of your new project in the Composer Design View.

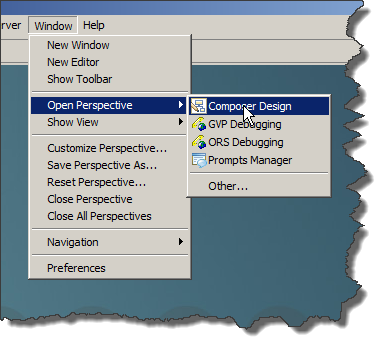


Figure 7: Change Composer Perspective

You may prefer to switch to the Composer Perspective and this can be achieved by clicking on the following from Composer.

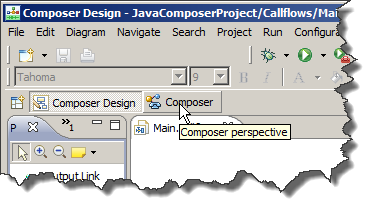


Figure 8: Change from Composer Design to Composer Perspective

In doing this you will then be greeted with the default layout. Each perspective may be modified and customized to your liking, but is not documented further in this document. Please refer to Composer documentation.

## 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.

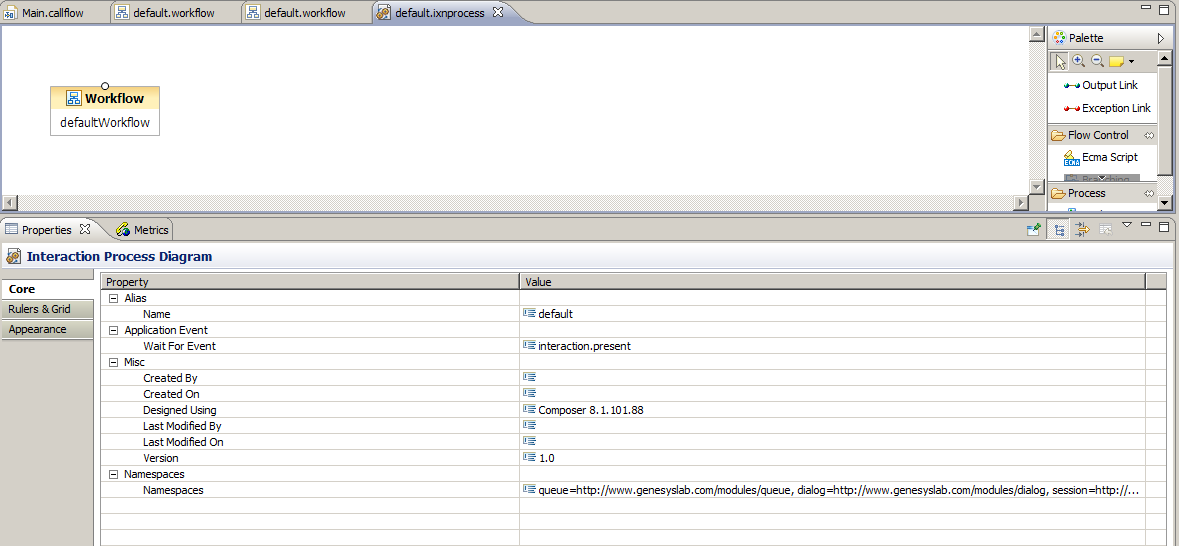


Figure 9: IPD Wait For Event Clearing

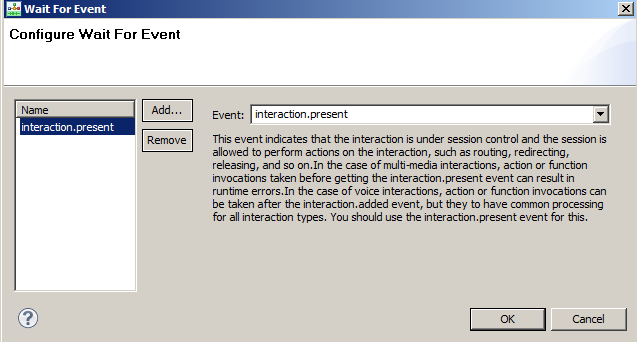


Figure 10: Wait For Event Dialog

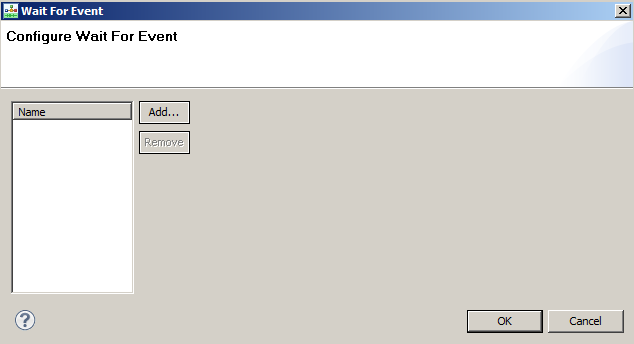


Figure 11: Wait For Event Dialog with No Events

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

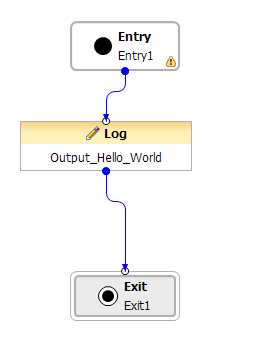


Figure 12: Hello World Workflow

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

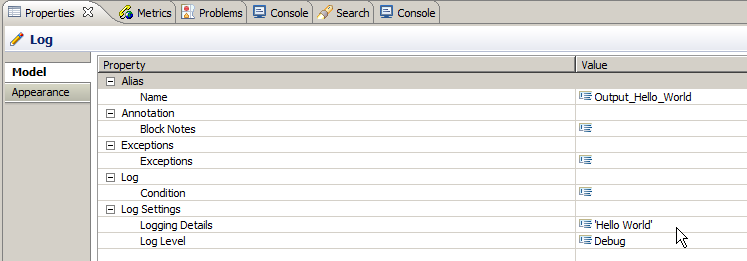


Figure 13: Hello World Block Properties

### 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 14: 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.

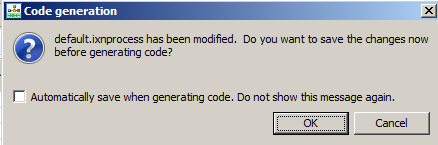


Figure 15: Code Generation Dialog

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

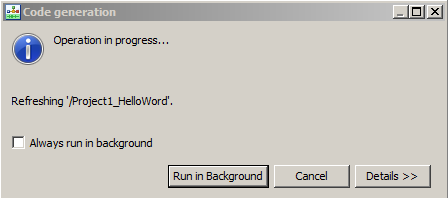


Figure 16: Running Code Generation Dialog

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.

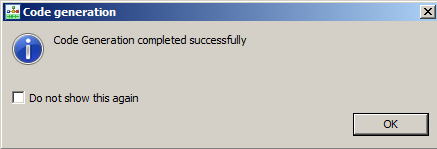


Figure : Success Code Generation

### Checking that we have the Project Deployed

Now let’s see if it is deployed or not. Right MC on Project. Then select tomcat deployment

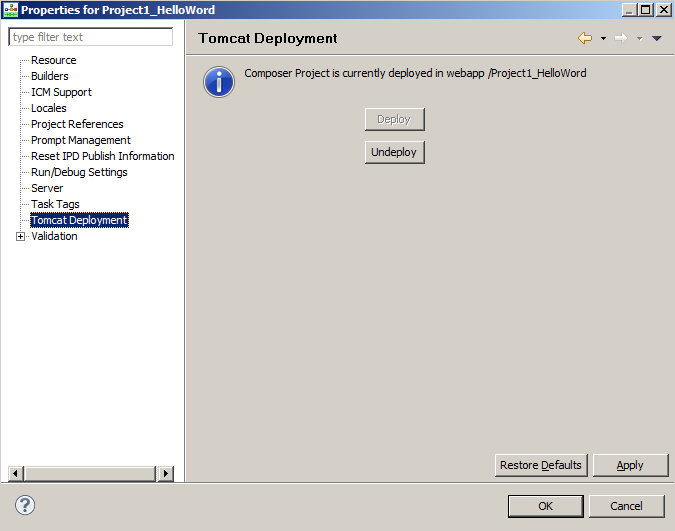


Figure 18: Composer Tomcat Deployment Check

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.

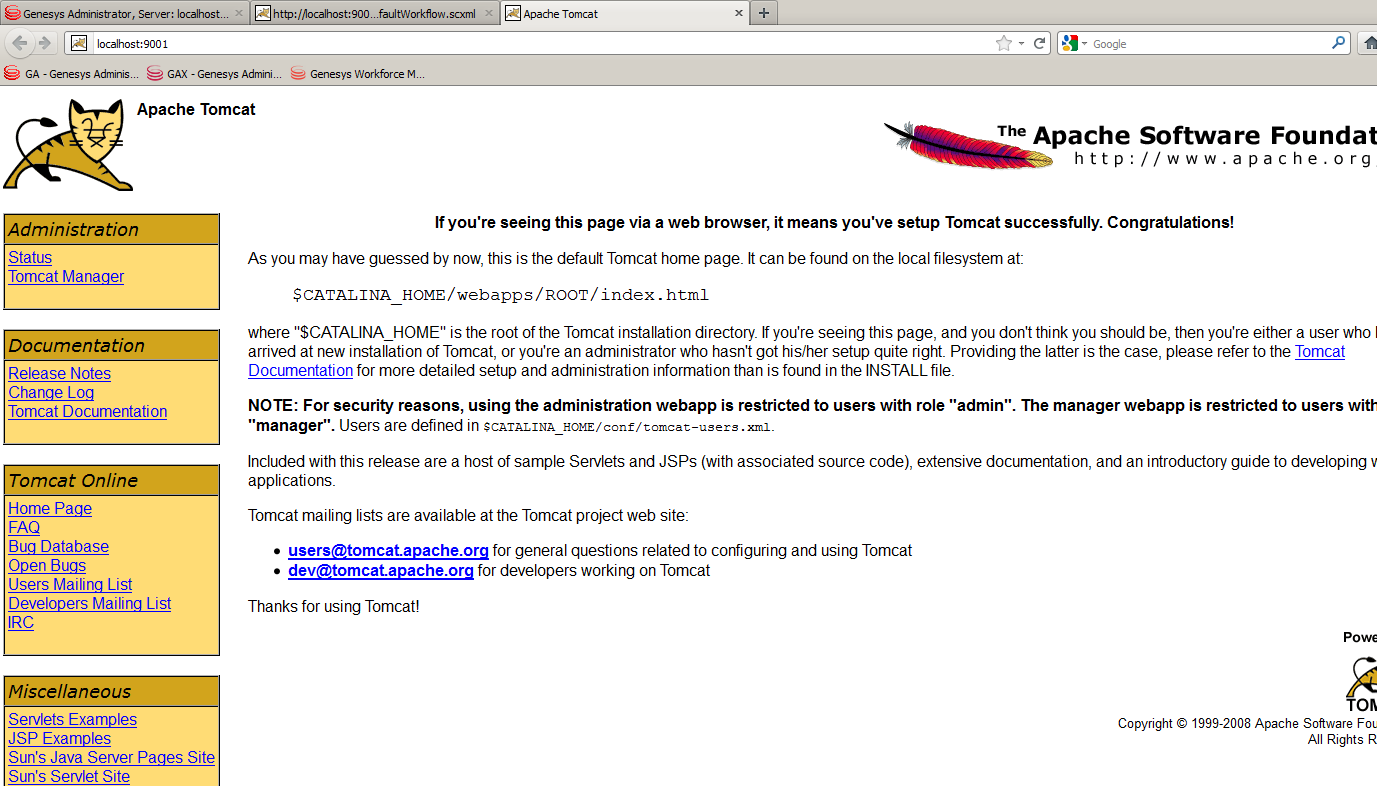


Figure 19: Apache Tomcat Confirmation page

Click on Tomcat Manager and enter the login details of admin/admin

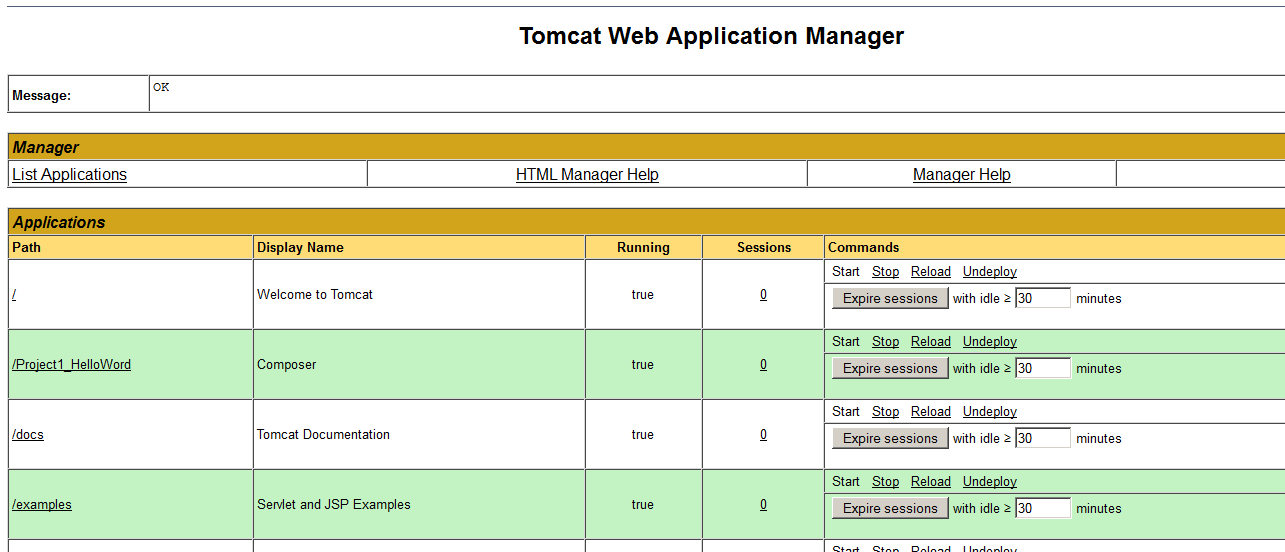


Figure 20: Tomcat Web Application Manager

### Checking the application further

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

http://localhost:9001/Project1\_HelloWorld/src-gen/IPD\_default\_defaultWorkflow.scxml

And you should get something like this…

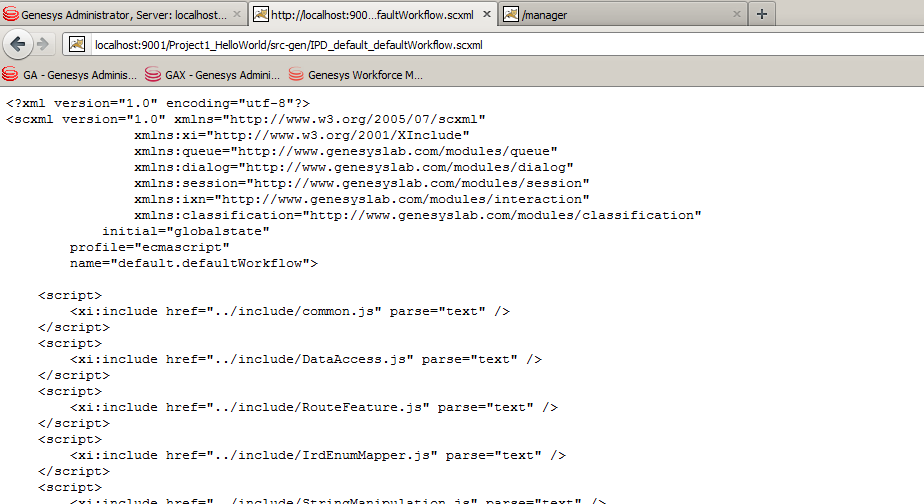


Figure : Browser retrieval of SCXML Code

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

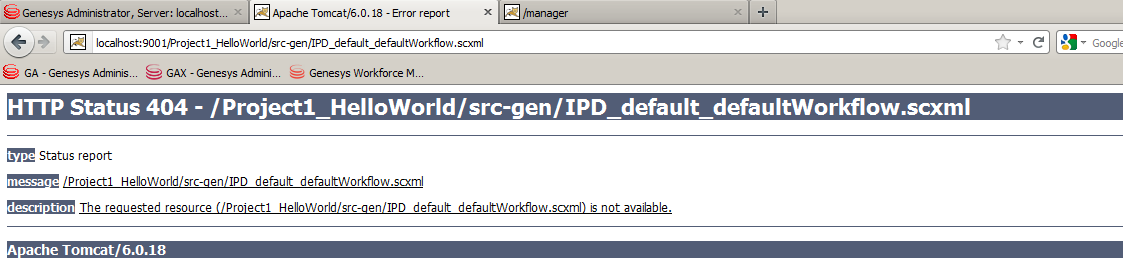


Figure 22: SCXML file not found

You will likely see this in Composer when this happens

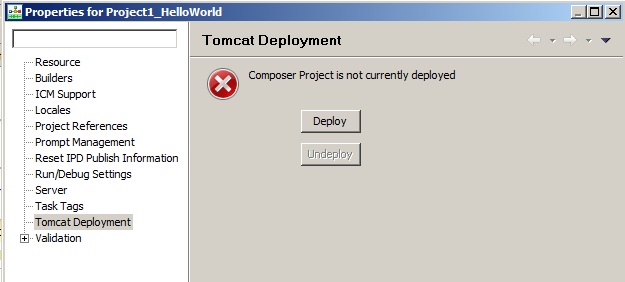


Figure 23: Composer Deploy button

## 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 if it is not installed on your TM Image.

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

**Please note – The current tech marketing image has RESTClient 3.4.2, please use this client for this example –**

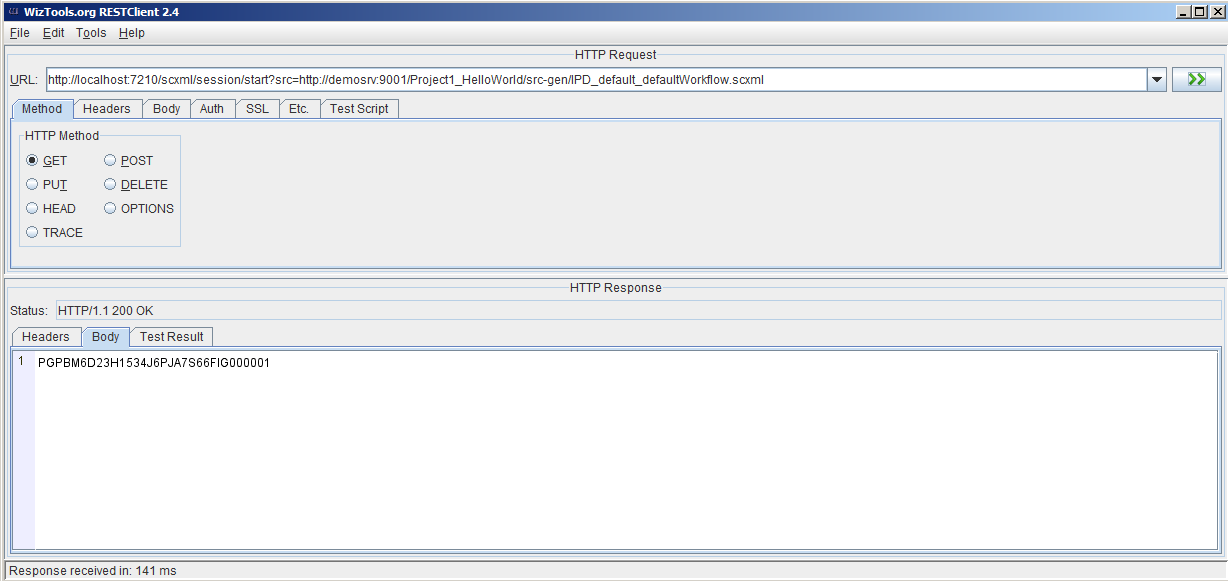


Figure 24: RestClient Interface

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

### Hello World

Once executed while there is a flurry of data in logs the main item of interest is the following. Yes, this is our 'Hello World' statement.

METRIC <log sid='PGPBM6D23H1534J6PJA7S66FIG000001' expr='Hello World' label='' level='4' />

## ORS Session and Events

The original application within its IPD had a wait for 'interaction.present' event which is related to strategies that are executing interaction based applications. However, we can define any event here and submit it via the RESTClient. This allows us to stall for a specific event before continuing on within the state machine but does not stop the state machine from being loaded. It only pauses the state machine from going any further in its execution.



Figure : IPD Wait For events

Add an event (events can’t have any spaces, but are hierarchal and can have periods (.))

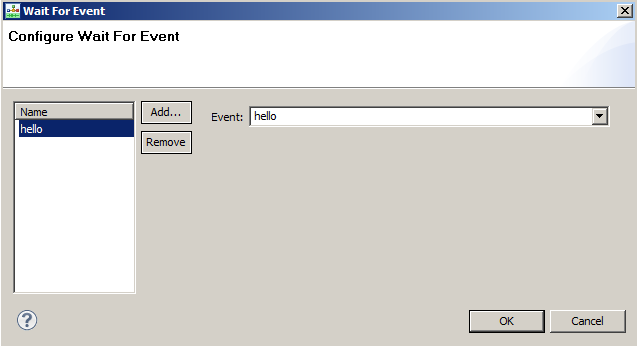




Figure 26: Adding a Custom Event

### Execute again with the change.

Run the application again after generating the code. From the rest client there is no difference but look at the log files.

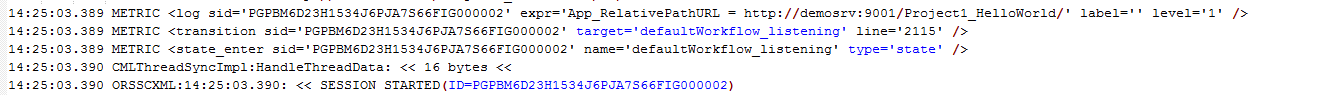


Figure 27: Waiting for Event in Log file

We land in the defaultWorkflow\_listening state. We are now 'stuck' waiting for an event.

### 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://localhost:7210/scxml/session/PGPBM6D23H1534J6PJA7S66FIG000002/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 28: 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://localhost:7210/scxml/session/PGPBM6D23H1534J6PJA7S66FIG000002/event/HELLO

Using the logs, we see the following



Figure : 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://localhost:7210/scxml/session/PGPBM6D23H1534J6PJA7S66FIG000002/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.

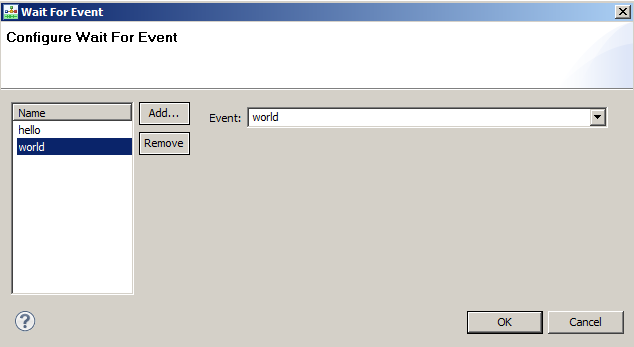


Figure 30: Configuring Multiple Events

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 having an understanding of the logs and the manual way of debugging allows you to understand more of what the debugger is actually showing you.

### Configuring the debugger

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



Figure 31: Debugger ORS Configuration

1. Need to configure Composer to access the debugger

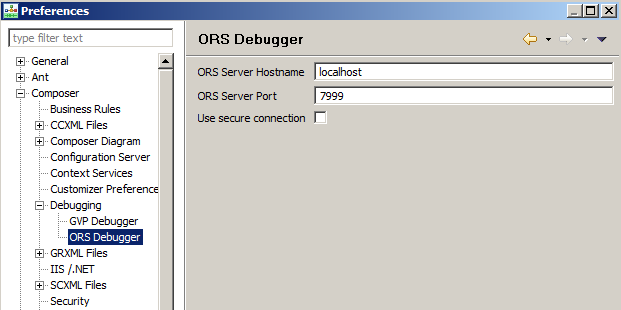


Figure 32: Composer ORS Debugger Configuration

### 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

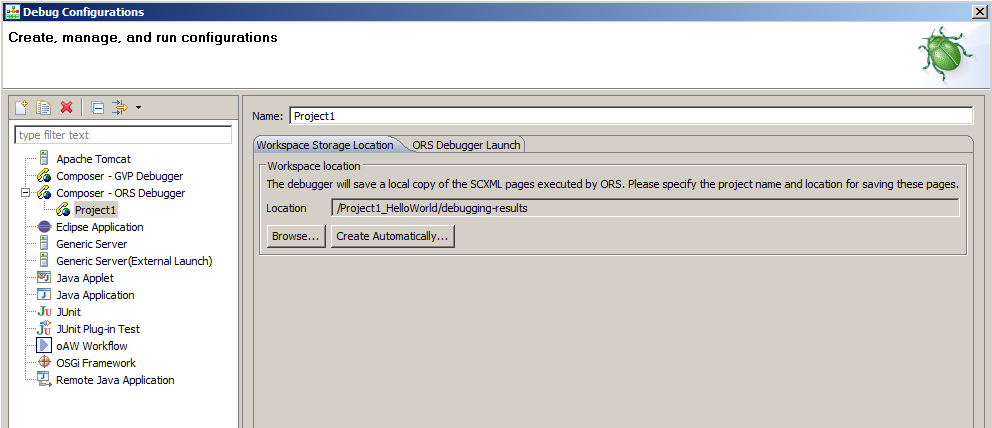


Figure 33: Debug Configuration for Workflow

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

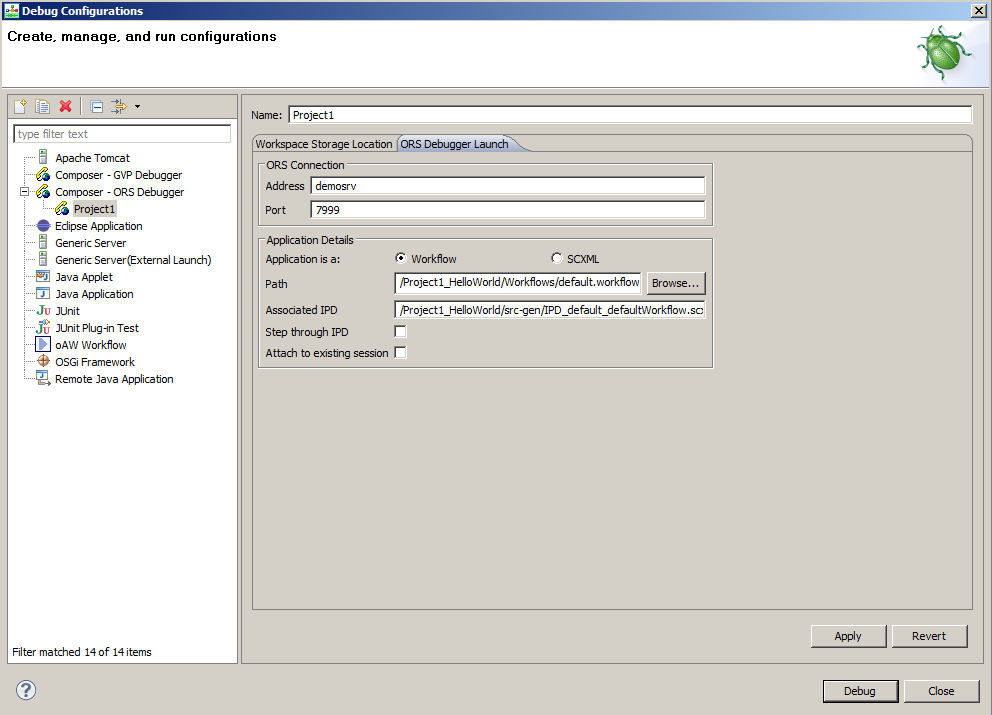


Figure 34: Debug Launch Configuration

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.

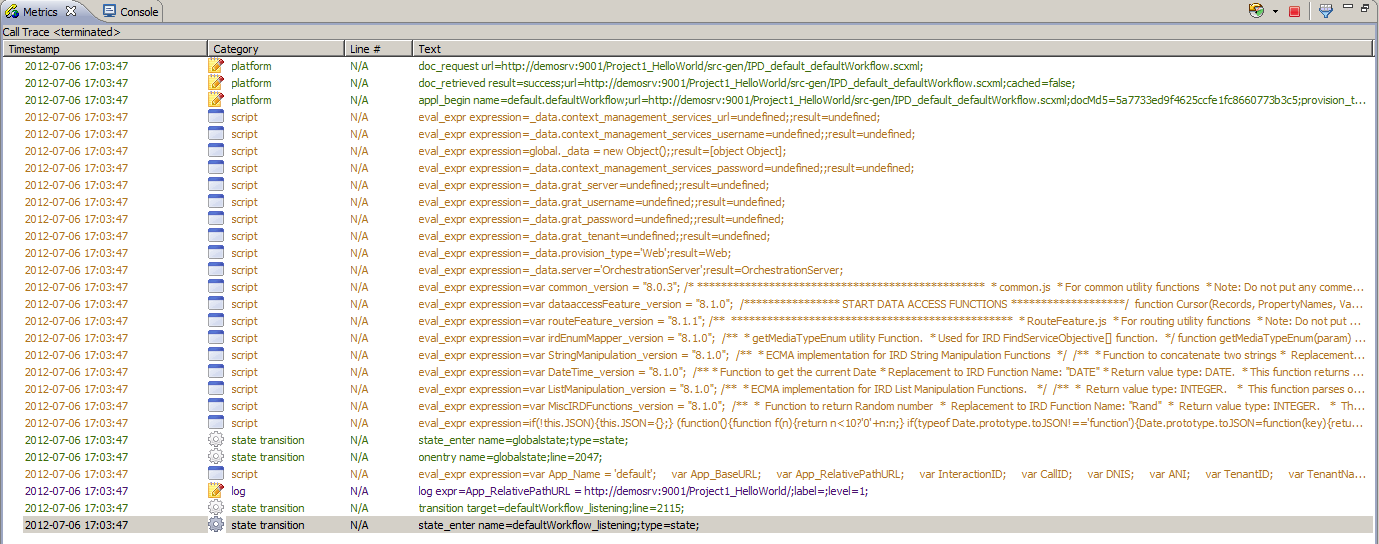


Figure : Composer Metric View

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

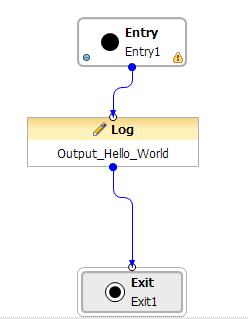


Figure 36: Break point on Entry

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

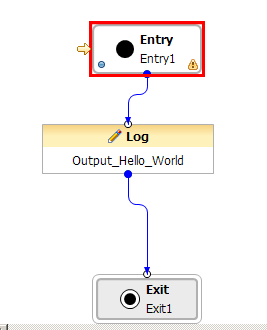


Figure 37: Composer signifies our position

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'

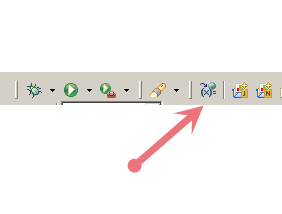


Figure 38: Workflow Variables access from Toolbar

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

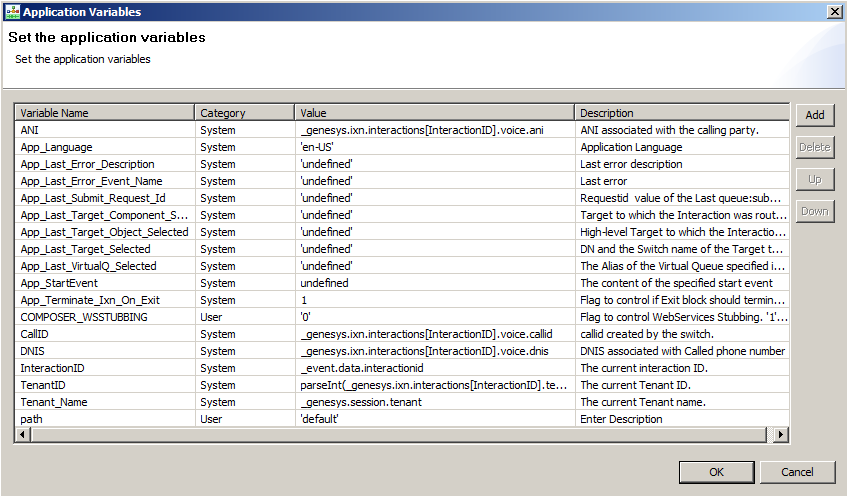


Figure 39: Application Variables

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.

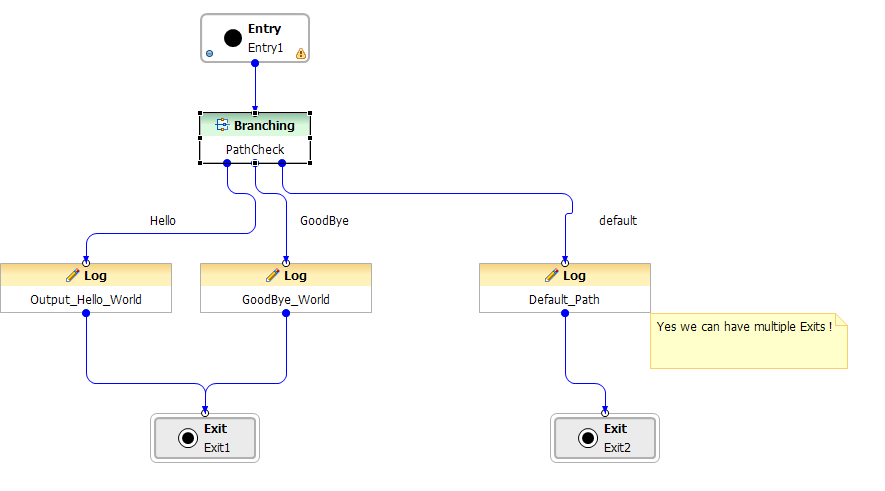


Figure 40: Advanced Hello World

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

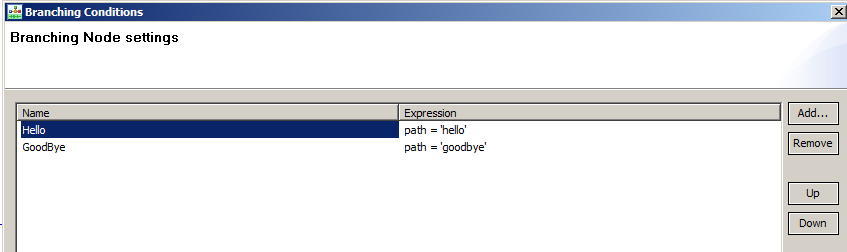


Figure 41: Branching Conditions

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

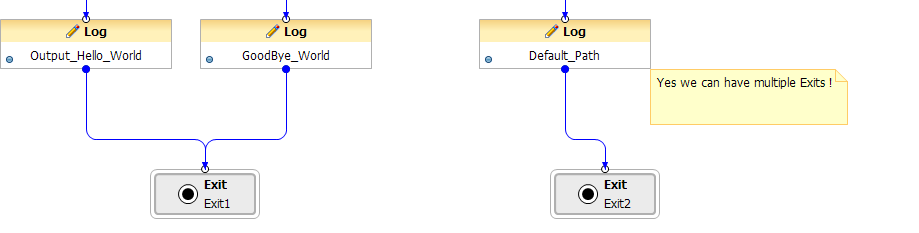


Figure 42: Multiple Exits in one diagram

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

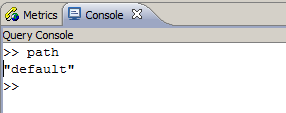


Figure 43: Debugger Console

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

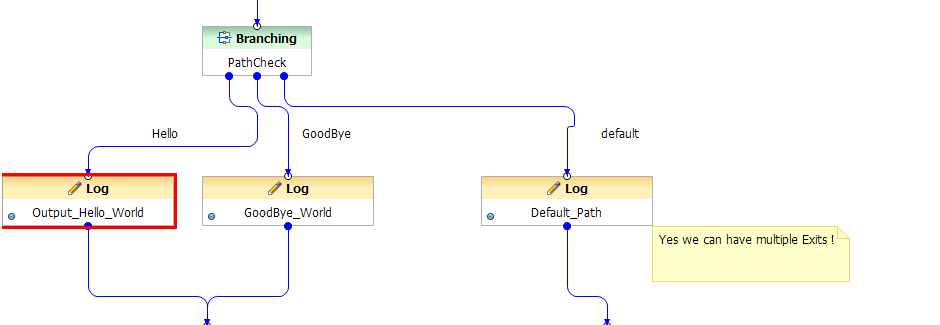


Figure 44: Wrong Path taken

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

Let’s check what path is set to now -

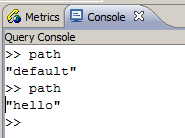


Figure 45: Using Console to inspect variables

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.

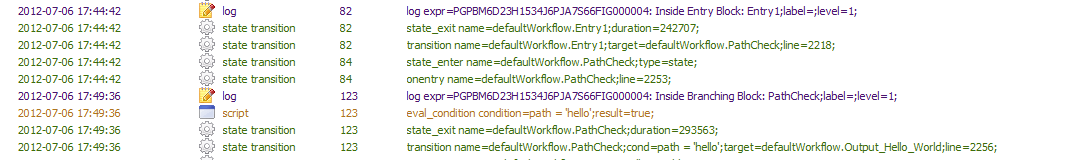


Figure 46: Viewing Metrics

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.

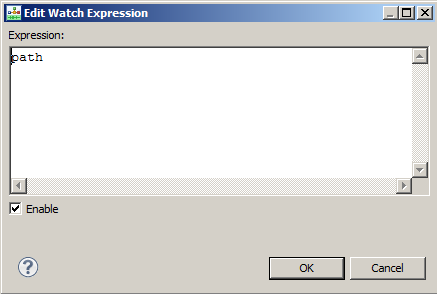


Figure 47: Add watch expression

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

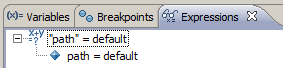
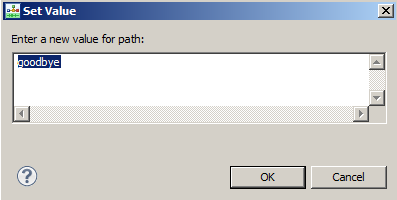


Figure 48: Current state of watch expression

Now let’s change the value to goodbye



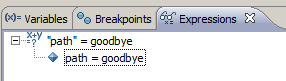


Figure 49: Modifying value from watch expression

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

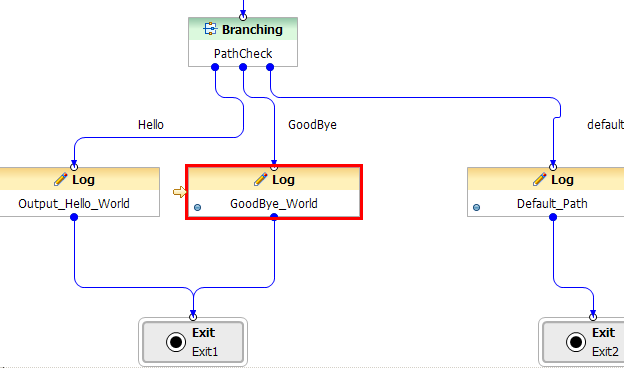


Figure 50: Observing the modified behavior

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

# Project 2 - Interacting with the ORS Session

Within this project, we will build a state machine that resembles a traffic light and interact with it externally, to drive it via events. In doing so, we will observe how events are handled in the state machine and how external systems can influence a state model.

## Create the basic project structure

Create a new java project and **don't place any spaces in the project name**. Use the blank template so we have a raw minimal project. Call this Project2\_StateModel.

### A Simple State Machine

State machines are all around us, and a really simple one (Well to Start with) is a traffic light.



Figure 51: The Traffic Light State Machine

Here we can see a definition of a state machine that models a traffic light that consists of 3 atomic states. Green, Yellow and Red, which may transition between and cycle periodically typically driven by events (timers).

### Introduce the state block

To model the traffic light in Composer we have a block that represents a state.

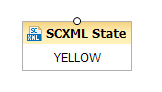


Figure 52: The SCXML State Block

The basic state block only has an entry point and no transitions. We first need to add a transition -

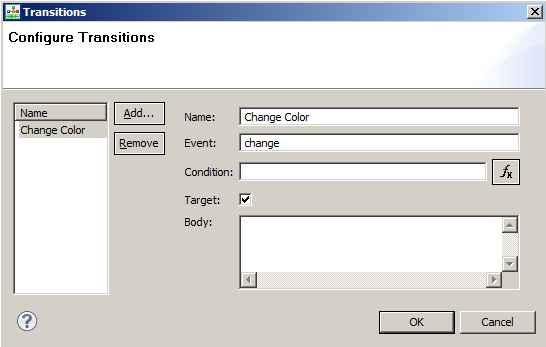


Figure 53: SCXML Transitions common dialog box

Make sure that the event is called change and that it is a target based transition.

## Building the Traffic Light State Machine

We can then build a simple traffic light state model by now copying the previous state and linking them together to form something similar to the following which represents a linear flow or procedural state transition model.

### Procedural State Model

The procedural state model simply allows each state to flow from one state to another on the raising and handling of an event and allows us to provide the basic traffic light flow. This pattern is often produced when you use composer to create your flows as each composer block typically needs to complete before advancing the flow.

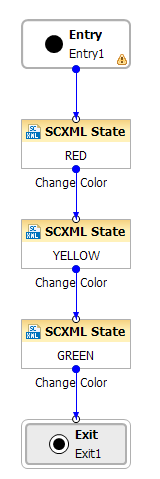


Figure 54: Procedural Traffic Light State Machine

### Getting the lights to cycle

Traffic lights pretty much need to loop continuously, so let's reconnect the GREEN state back to RED. It now looks like this

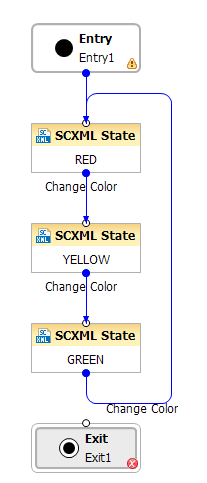


Figure 55: Incomplete loop

Notice the red X on exit. We cannot have a session that never exits we need to be able to have a session that can exit and this is flagged by Composer for you.

### Allowing the Traffic Light to Terminate

OK so we need to terminate the Traffic Light and we can do that by introducing a new event called terminate to each state. So we now have something like this which makes Composer very happy.

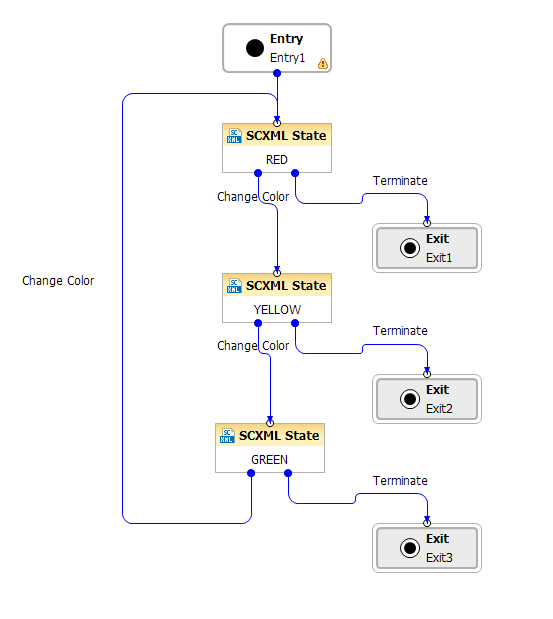


Figure 56: Handling Terminate at all possible locations

### What a MESS!

OK great so now you can handle the event on every state. However, this now allows your diagrams to become very cluttered. In essence you are also having to do more work since each state has the potential to be terminated and you have had to repeat the code in all places that this event is able to be raised. There is room for improvement here.

Tip - If you find yourself creating diagrams with duplicate code then it's probably time to grab a coffee (or your beverage of choice) and look at refactoring how you have created it you probably are opening yourself up for all sorts of issues.

## Hierarchical States

As you have seen thinking in a linear or procedural manner really does result in a more complicated flow. You now have now duplicated events on all blocks that represent a state since you can terminate in any state, when we want something more global such that we can terminate from any state. This is where we can take advantage of the Hierarchical States within SCXML.

### States and Sub States

States can contain sub states which intern provides a hierarchy of states because sub states become nested, which is important when modeling state machines and event handling because you can be in more than one kind of state at a time. For example, we could be within a 'Handle Call State' That is currently 'Identifying A Customer' and which would then 'Find A Target' and then 'Route to the Target'. Which can be represented as follows;



Figure 57: Hieratical State Machine

### Outline of a Generated IPD

Show the outline of a typical IPD project, use the debug file since this is the complete document and use the outline view to show that we have nested states.

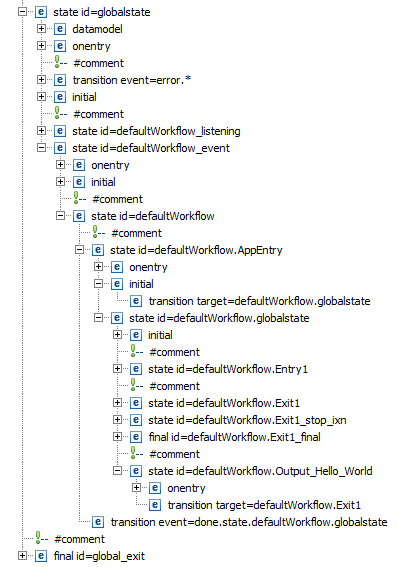


Figure 58: Code Outline of IPD

We can now recreate the same behavior simply by building the global transition into the Entry block, since all States (and blocks) within a workflow (or subflow) are child states of the entry block. This corresponds to the workflow.AppEntry state.

### Taking advantage of the Hierarchical States

To leverage the Hierarchical nature of the state machine that Composer creates we can remove all of the transitions from the states

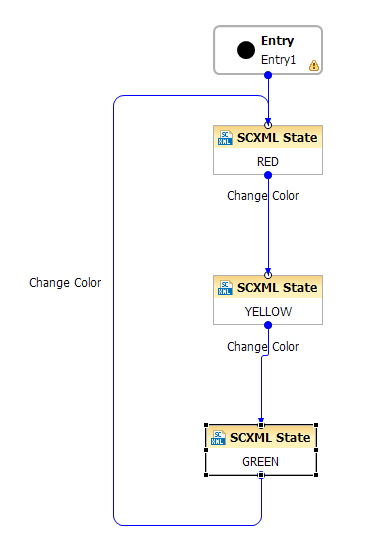


Figure 59: Rebuilding the state machine

### Adding a Transition Handler on Enter

The individual links to exits can be replaced with a single transition handler at the parent state which is the Enter block within the diagram. To be able to achieve this we currently need to add an exception handler, although the term 'exception' is really something that Composer introduces and not SCXML. Exceptions are present on most blocks.

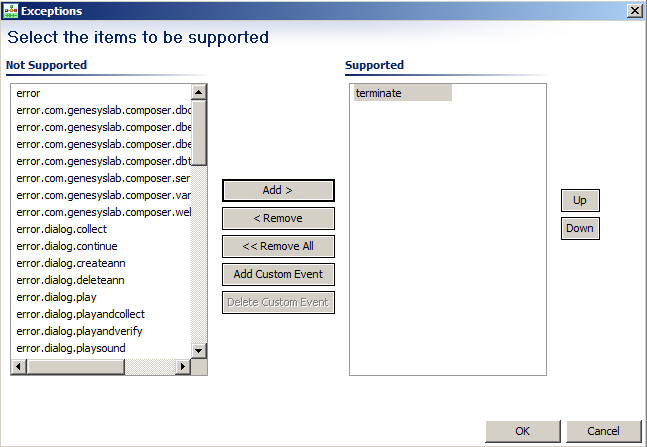


Figure 60: Adding Transition

Remove the other events. We now have this on the Entry block

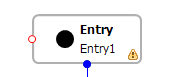


Figure 61: Adding Transition on Entry Block

Simply add an exit block

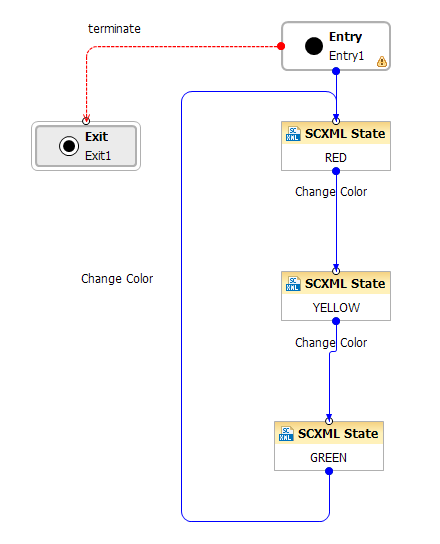


Figure 62: Updated Flow with hierarchical state event handling

Note: - Composer will be supporting Transitions on the Entry block in its final release, which can be used in place of Exceptions which do not include the ability to specify order, target or condition

### Debug the flow

Using debug configuration. Duplicate project 1 configuration and modify the settings so we can debug this application

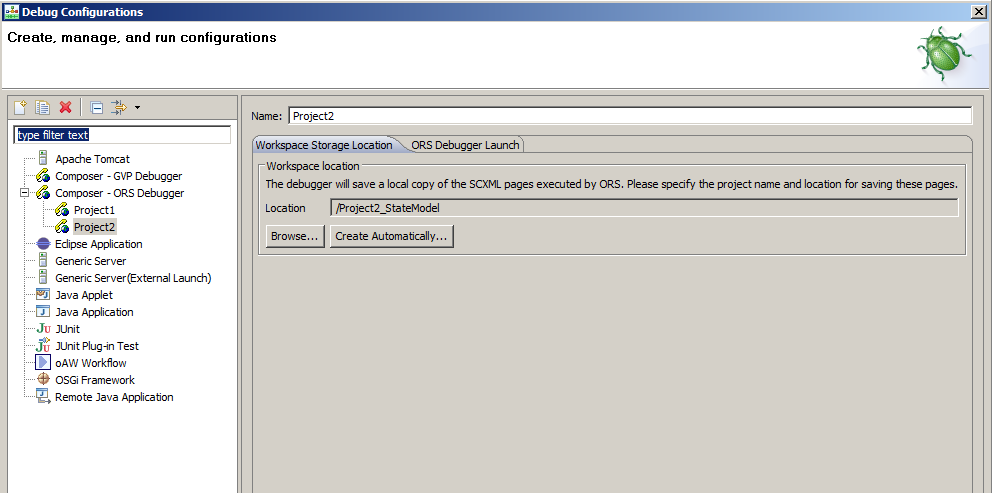


Figure 63: Project 2 debug configuration

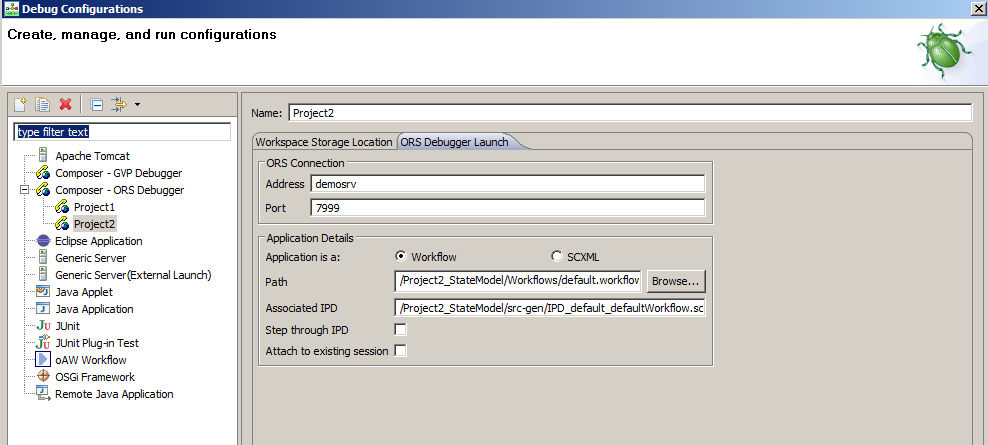


Figure 64: Project 2 ORS Debugger Launch Information

Then start a debug session and see how sending the change event can advance the session. The state of the session can also be queried by executing the following with the returned session ID.

http://localhost:7210/scxml/session/PGPBM6D23H1534J6PJA7S66FIG000002/query

### Handing a deploy issue error message

If you are starting or attempting to start a debug session and come across this. You more than likely have failed to deploy the application code.

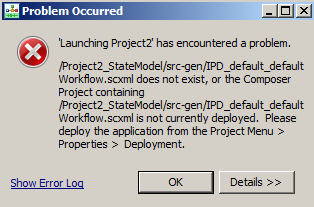


Figure 65: Debugger Deploy issue detected

You now need to perform the previous steps to ensure that the application is deployed. Once deployed, experiment with submitting the terminate event and see what the metrics information looks like.

### Hierarchical event order.

When events are handled there is a series of semantics that are entertained as described within the W3C specification for SCXML. However, the main key item to remember is that events are processed in a hierarchal order. They are first applied to the lowest state and if not handled are applied to its next parent state. If the event is not handled it is essentially ignored. But in most cases as soon as the event is handled it will stop being handled in any parent state.

To help demonstrate this we will now add a terminate event handler while in the 'green' state.

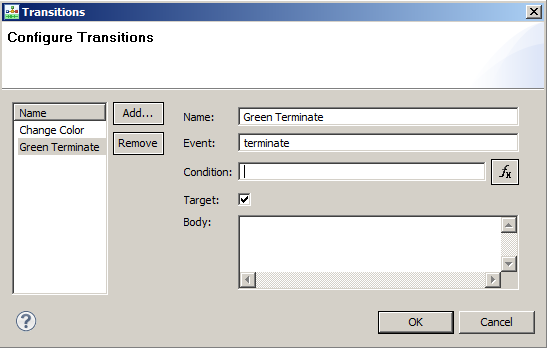


Figure 66: Terminate Handler

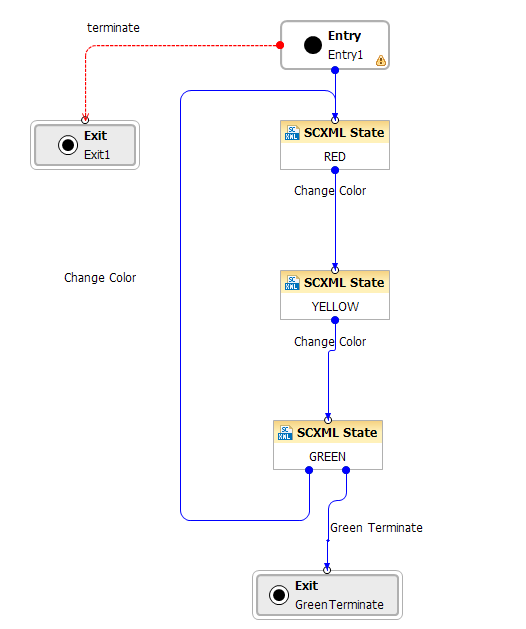


Figure 67: Green Terminate Handler

From this we will now add in a specific exit and see which exit we will terminate at

### Running the debug with the terminate event

We will now re-run the application in the debugger and transition down to the GREEN state. Once there we will then issue the terminate while in the Green state and see that this takes precedence.

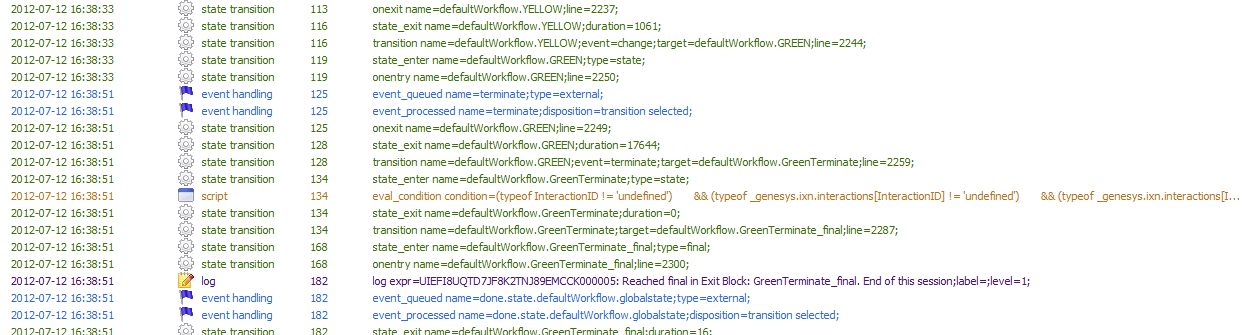


Figure 68: Green Terminate Metrics

## Transition Conditions

When an event is submitted it may also provide data with it which is very common. Transition are allowed to inspect the event data and can influence if a transition is taken or not. We will modify the application to now expect a parameter called COLOR to be passed and only if in the Green state the data is passed as GREEN will the green terminate be taken.

### Set up the Rest request

1) Set it to be POST with the following Body content

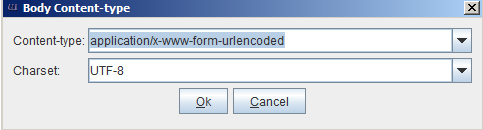


Figure 69: Set content type and Charset

2) Set the body to contain the data we want to pass

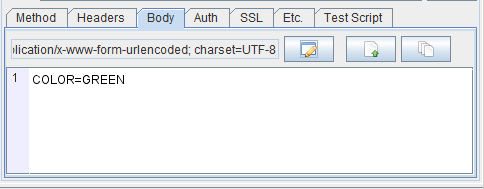


Figure 70: Set Body of message to include the property COLOR

### Build the condition on the GREEN state

Here we will look to see if the \_event has the properties we expect and also in the transition body log a message to say we are within the transition. The condition will be as follows;

\_event.data.param.hasOwnProperty("COLOR") && \_event.data.param.COLOR == 'GREEN'

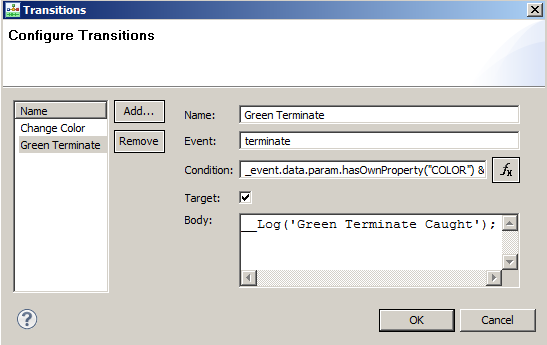


Figure 71: Green Terminate Transition

Note:- This is an incorrect Body entry for a transition. It really needs the <script> tag to perform any Java Script to be valid. But we are doing this to show how its tracked as an error.

#### Execute or attempt to execute the session

We will now attempt to initiate the application which has an invalid execution body for the transition that we just added. Any client that attempts to execute such an invalid document will receive an error response back from ORS.

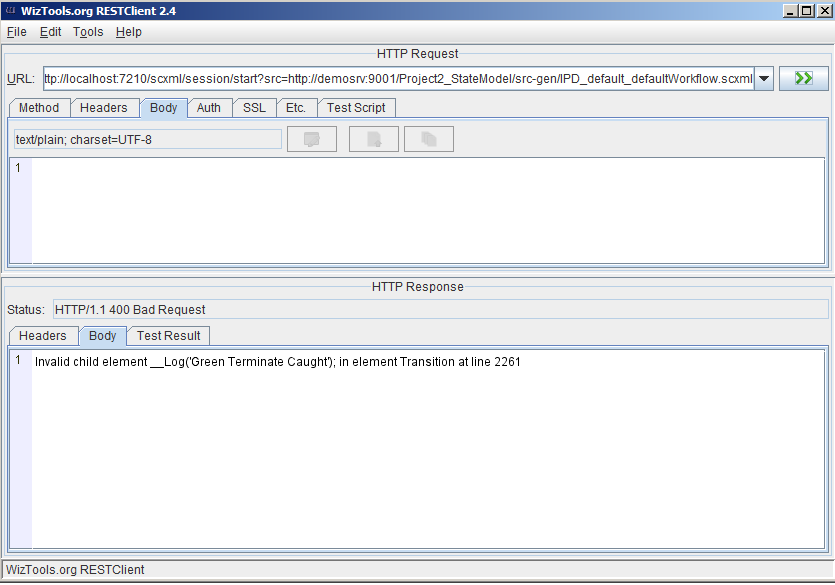


Figure 72: HTTP 400 Error response

And in the debugger we see the following since we know someone tried to execute the request but we could not start the application.

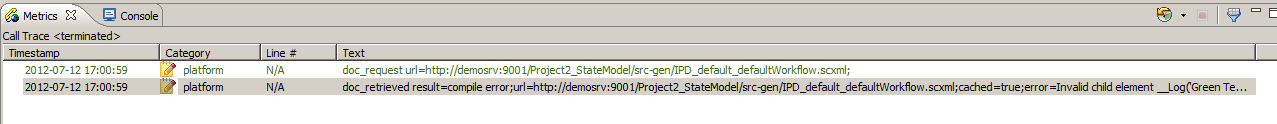


Figure 73: Platform Metrics indicating error

### Why the Error.

So why do we error out, well let’s have a look at the SCXML code generated. Composer actually did try and warn us about this as showing in the src-gen folder with the x against the generated file.

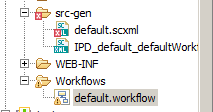


Figure 74: Composer's file error indicator

Then if we open up the file we can now see that the error is also flagged for us as well.

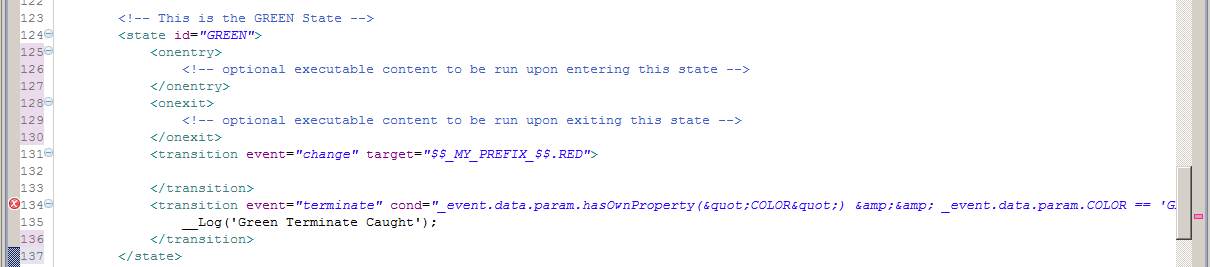


Figure 75: SCXML file error display

Notice the X and red bar. It says the code is invalid....

### Let’s correct the error and continue.

Correcting the code, is fairly easy. We simply need either a <script> tag surrounding it since it is a ECMAScript function. Or alternatively you may want to use the <log> tag that is built into SCXML. Here we will use the <script> tag.

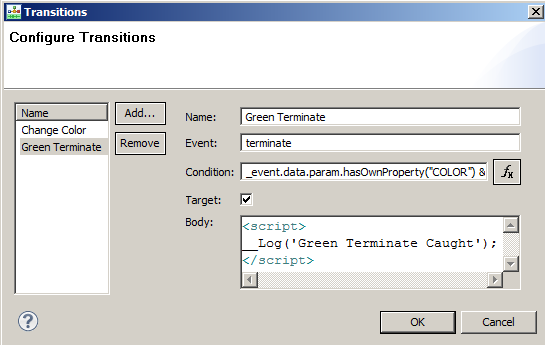


Figure 76: Corrected code

Upon code generation we can now see that the previous X on the generated file is no longer there.

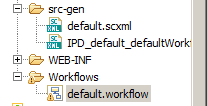


Figure 77: OK file syntac

### Starting the application again

The error has gone so we can now start the application. So let's launch the debugger for the application and advance to the green state. We will then submit the terminate event without any data.



Figure 78: Terminate Event Handling

Note the condition was false and we exit out of the higher transition for terminate since the expression for the Green transition did not evaluate to be true and so was ignored but the higher transition was taken as shown below.

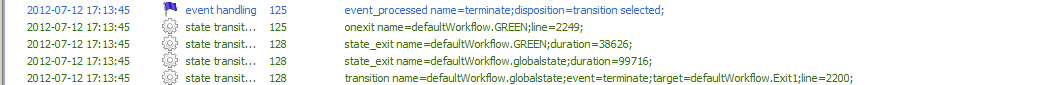


Figure 79: Top level Terminate taken

### Why the need for <script>?

Q) Why do we not have <script> auto added for us in a transition?

A) The Transition can execute any executable content, which means any ORS SCXML extension can be placed in a transaction body in addition to just JavaScript.

### Sending in the terminate event with data

We will now repeat the process and this time add in the data to signify the COLOR. So let's restart the process and again move to the green state. But this time lets submit the terminate with the actual data payload. We should place a break point on the GREEN exit and should observe that we take that transition.

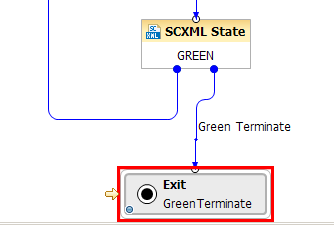


Figure 80: Lowest level Terminate taken

#### Inspect the Event to see what is populated

While situated on the Exit we can see what the last event was that was sent. All events are sent passed as \_event. We can inspect the content of the \_event by using the debug console.

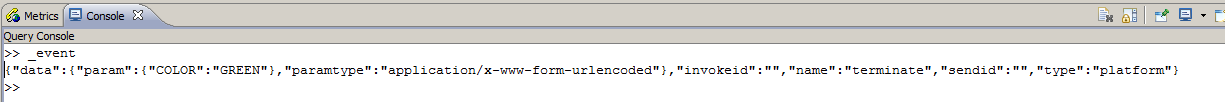


Figure 81: View of the \_event in JSON

## Student Exercise

You can of course send multiple properties in such a request and also build up condition to inspect more than one property. As an exercise create a condition that only executes the green terminate transition if the COLOR is GREEN and the DAY is MONDAY.

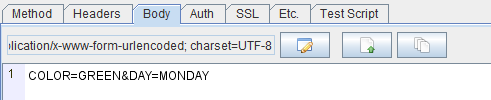


Figure 82: Additional Criteria

## Multiple Transitions with the same event name

Transitions are not only hierarchal but are also order dependent as well, this is referring to as document order, and transitions are executed within a state in top down manner. Currently we do not expose in the transition dialog box the ability to reorder (In this current version of Composer).

The ability to add multiple transitions for events allows us to provide the concept of filtering and more refined transition choice. We can also add transitions that do not transition anywhere but that may perform some function in its executable body.

### Adding a target-less transition

Let’s add another transition for terminate and set the condition to be the following

\_event.data.param.hasOwnProperty("COLOR") && \_event.data.param.COLOR != 'GREEN'

Within the body of the transition we will Include a regular log statement. Since <log> is not ECMAScript we do not need to wrap this with the <script> tag.

<log label=*"WRONGCOLOR"* expr=*"'$$$$$ TERMINATE EVENT='+ uneval(\_event) "* />

The following is what the transition dialog should look like once populated.

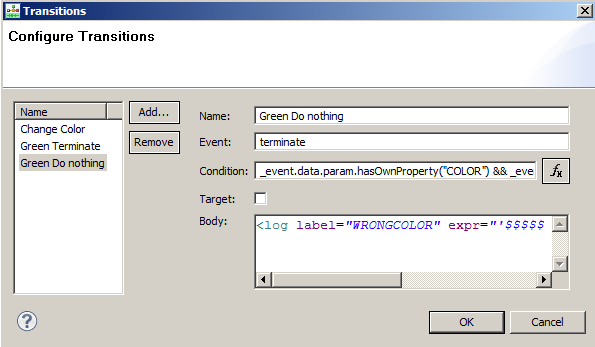


Figure 83: Target-less Transition

### Evaluating the new terminate event

We will now rerun the application and submit the terminate event when we are in the GREEN state. We can see the evaluation once the event is submitted occur in DOCUMENT ORDER



Figure 84: Document Order Transition Evaluation

Once the event is submitted if we don't send the COLOR in the event then the global transition for the terminate will fire

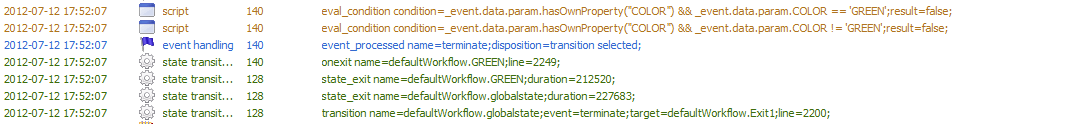


Figure 85: Terminate State transitions

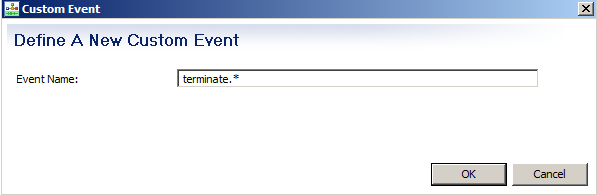
### Event Wild Cards

Events within SCXML can be hierarchal or structured which can be seen from the likes of the interaction events as outlined in the ORS wiki located here [http://docs.genesyslab.com/wiki/index.php?title=Interaction\_Interface\_](http://docs.genesyslab.com/wiki/index.php?title=Interaction_Interface_Events).

Within ORS most events defined by ORS start with the Functional Module name (i.e. session, interaction etc.), followed by the operation and can have many other sub levels to it. In some cases, this hierarchy can be used to provide event handlers that catch a subset of events. SCXML only supports a single level of wild cards and does not support partial event matching i.e.

**interaction.\*.done** is not valid.

We can add a wild card event handler by modifying the terminate so that we have terminate.color and terminate.\* as the event names. Wild cards allow you to provide a catch all and some level of hieratical event structure.



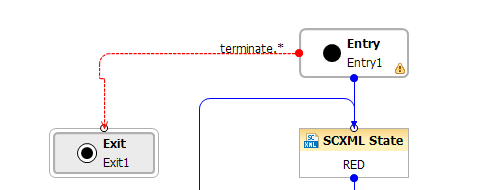
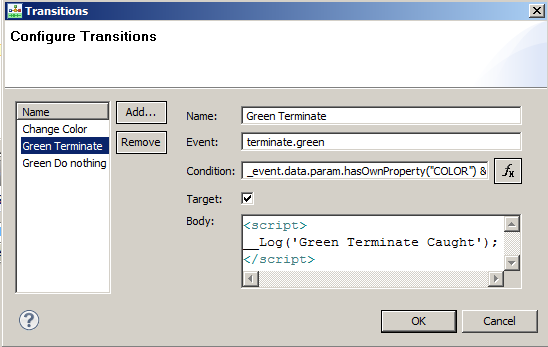


Figure 86: Modification of terminate event

### Add additional Transitions on GREEN state

On the GREEN state we can change it so that we now have the following transitions



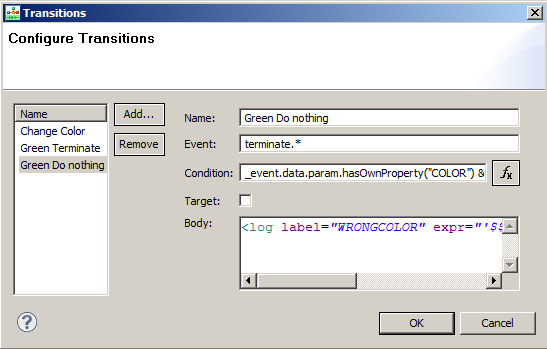


Figure 87: Hierarchical Transition Handlers with Wild Card

### Testing with event wild cards

Again we will start the application in the debugger and submit the terminate event. But this time we will send in send in terminate.green.

http://localhost:7210/scxml/session/UIEFI8UQTD7JF8K2TNJ89EMCCK00000E/event/terminate.green

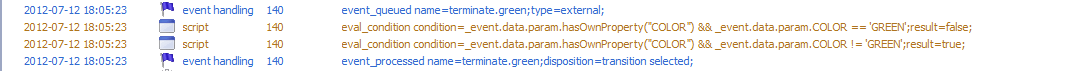


Figure 88: terminate.green event handling

Here since the event matches exactly to our event we see that we handle that. However if we now send in terminate.black with color green we should see the higher level transition handler get called and terminate the application.

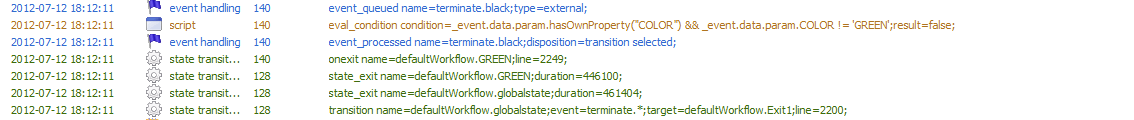


Figure 89: terminate.black event handling

Notice we don't see the terminate.green transition handler show up in the log, as there is no match on the event so the transition is not selected.

## Composer Global Handler - Or why your calls just stop

Handling or mishandling errors in Composer often leads to the Global Error getting called. Any time an error is raised that is not caught it will traverse the state hierarchy and get handled at the global error handler. This can be a good and a bad thing. Good in that your sessions are at least going to exit gracefully but bad in that it's really a default handler.

### Faking an error

If we use the current application but send in an error event (error.foo) we will see the global event handler getting caught.

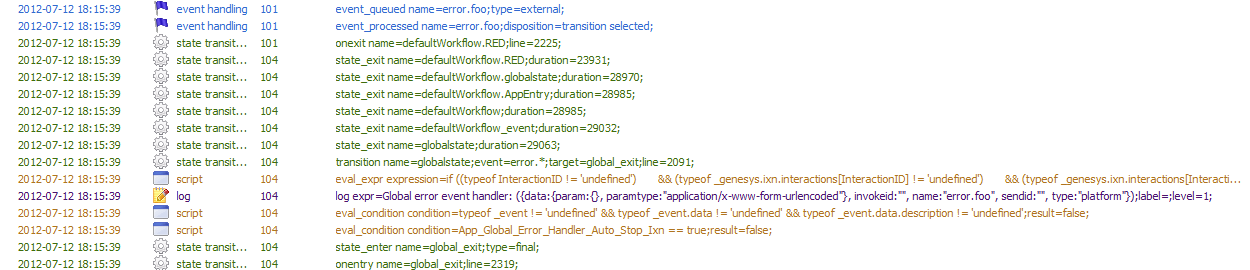
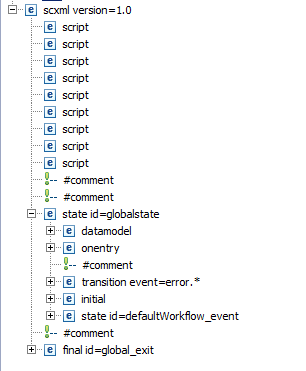


Figure 90: Composer Global Error Handling

**Any event that starts with error, is a special event within ORS. All FM actions on error will return this event and if you don't handle the explicit sub category of error Composer will do it for you. Therefore, you should never create a custom event called error or anything starting with error and rather leave error to be reserved to be raised by ORS and any of the Functional Modules within ORS.**

### The Composer Global Error Handler

When looking at the IPD generated code you can see the global error handler that Composer creates. This ensures that the erorr is handled if it is ignored elsewhere. If you don't want the default behavior of the application just quitting then you should add your own error handling code to act accordingly.



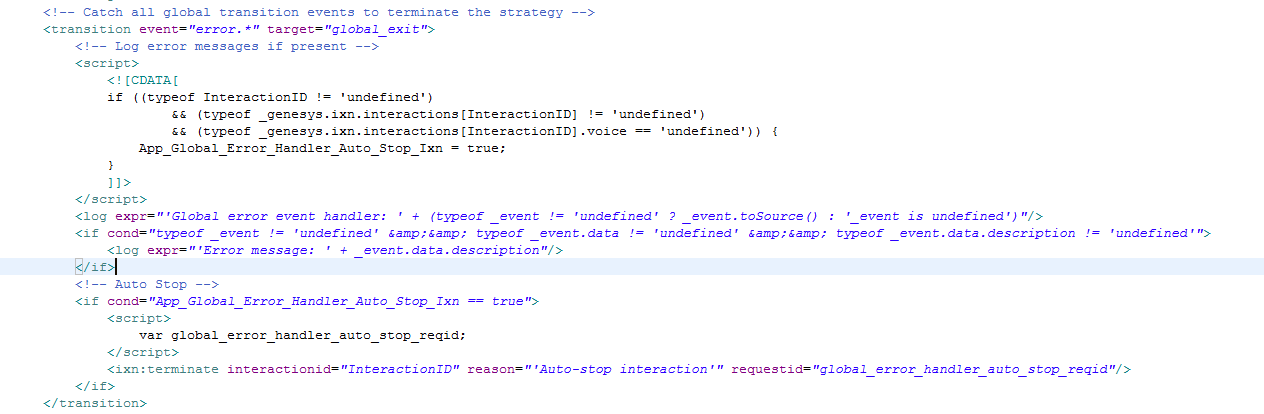


Figure 91: Global Error Handling Code

## Passing Data on start

When an application is started via the ORS web Rest API. We not only can state whcih SCXML document to load and to start but we can like sending an event to a session pass data on start. Such data then either replaces data in the <datamodel> or is appended to the <datamodel> whcih is accessible via the \_data property from within the session.

Let’s now send in an initial color value when the application start and it will then select the appropriate starting color of our traffic light.

### Add a Project Level variable called Color

To support data passing we need to create a project level variable in Composer. In doing this it will create a <datamodel> section in the generated document that can be populated via the start mechanism (As well as provisioning that you will see later)

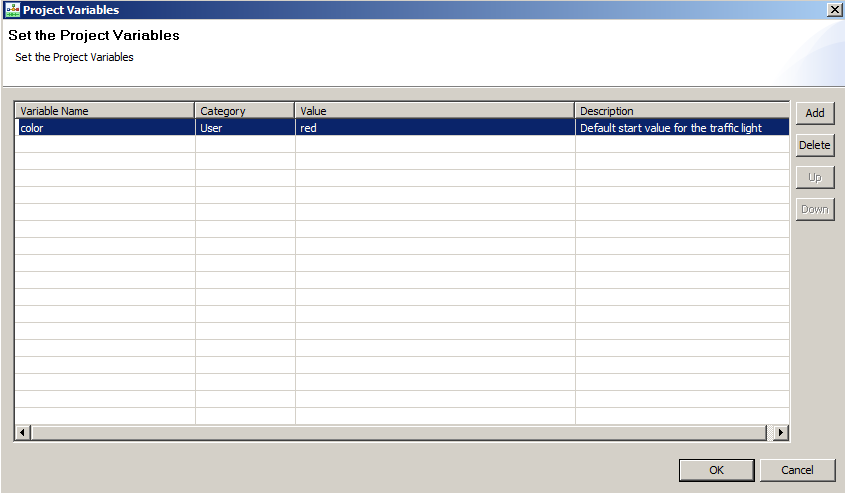


Figure 92: Project Variables

Once the IPD is created we can now observe the generated <datamodel> code within the SCXML document, including how the default value of red is handled.

### Passing Initialization Data on start of the application to the <datamodel>

Within the Rest client we can set it up so that we pass in color and a value. This can be passed as URL encoded or within the body of the start request. (We have two rcq files provide for this)

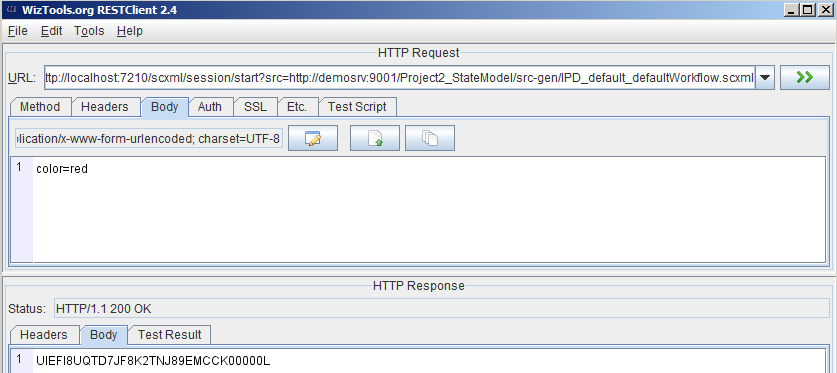


Figure 93: Passing Data on Start

If we start the debugger (Try pressing F11!) and the start the session with either a form post or the URL encoded values for the color, we should see that the color we pass on such a request is submitted to the session. If you place a break point on the Entry block you can inspect the project level value called color.

### Inspecting the <datamodel>

Once started within the debug console you can inspect the data model by simply typing \_data

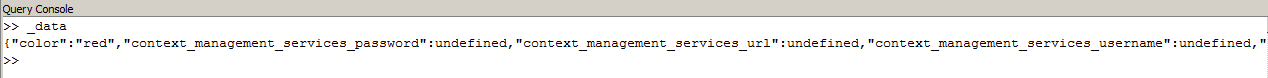


Figure 94: Console view of <datamodel>

The project variable has been populated with the value from the start request.

## Student EXERCISE: - Add a branch based on start up color

Now that you have the initialization property passed extend the project so that you can now branch based upon that initialization color and start at the correct color based upon that. If there is no color or an invalid color, then simply start as if nothing was passed.