

Hello, World!

Version: 10.0 / Modifications: 0

Introduction

```
Hello, World!
```

As with all new programming languages, the "Hello, World!" program generally is a computer program that outputs or displays the message "Hello, World!". Such a program is very simple in most programming languages and is often used to illustrate the basic syntax of a programming language. It is often the first program written by people learning to code.\

✨ Now step inside and follow these steps to complete your very first composition with Composable Architecture Platform.

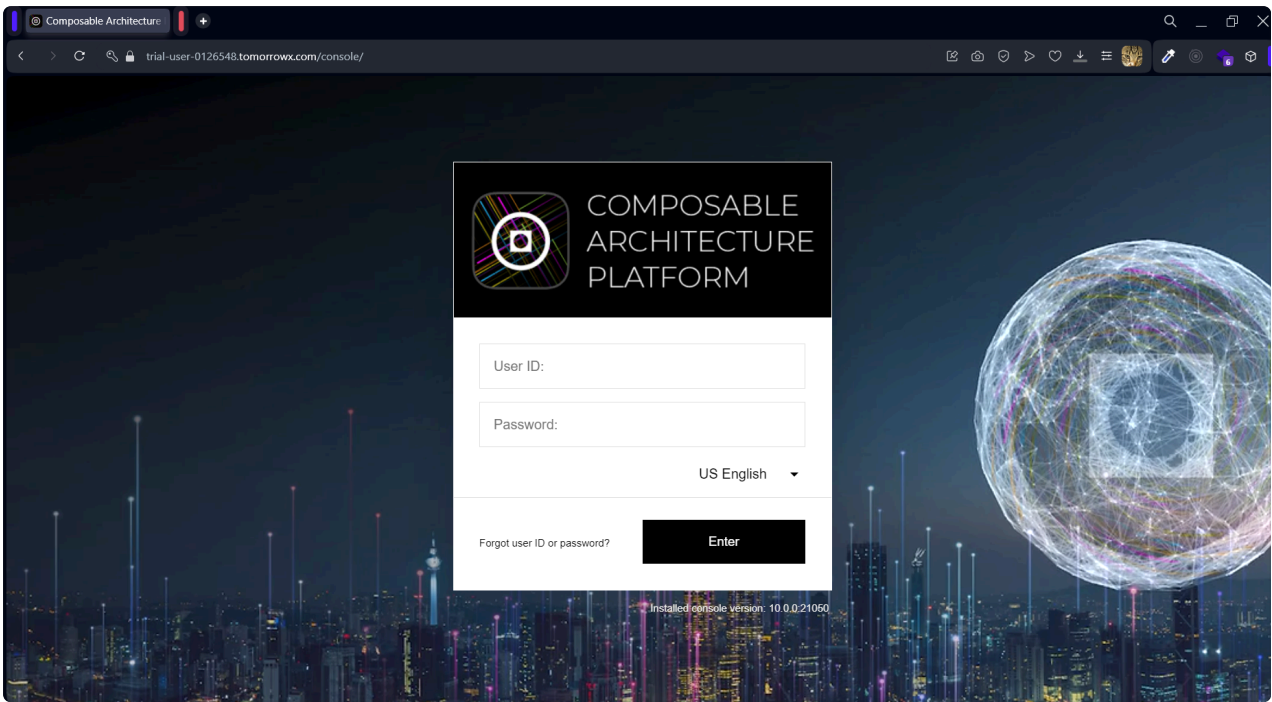
Requirements

- A running local, or cloud hosted instance of X.
- Installed console version 10.0.0.21050 or later.
- Chrome or Firefox browsers are supported.
- Ports 80 and 443 are required to be available to run the console and X Engine.

For the purposes of these instructions **[your server name] = localhost**

For example: [http://\[your server name\]/console/](http://[your server name]/console/) = <http://localhost/console>

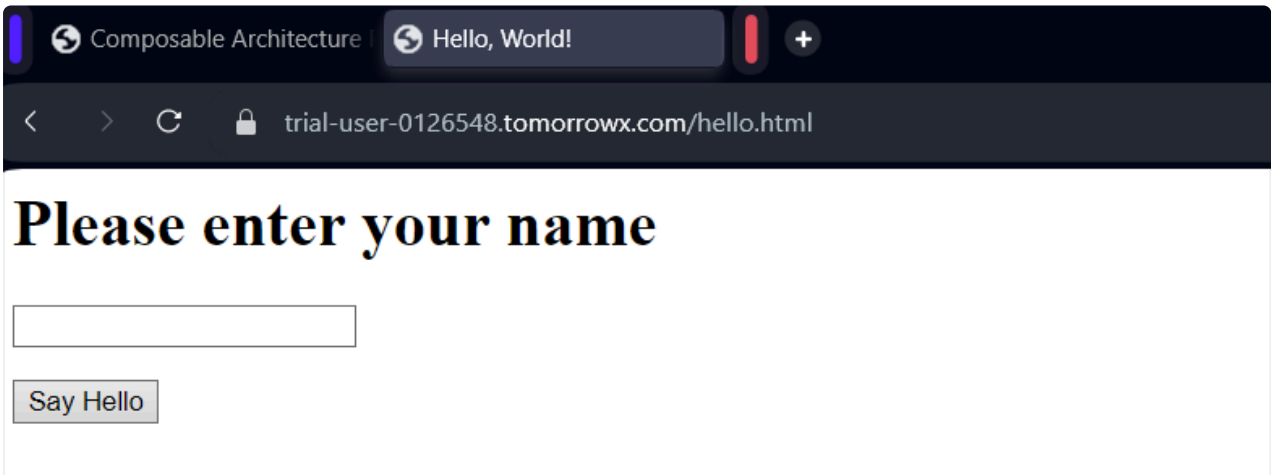
You need access to a console login screen like this:



Composable Architecture Platform - Console -

Say Hello [content file]

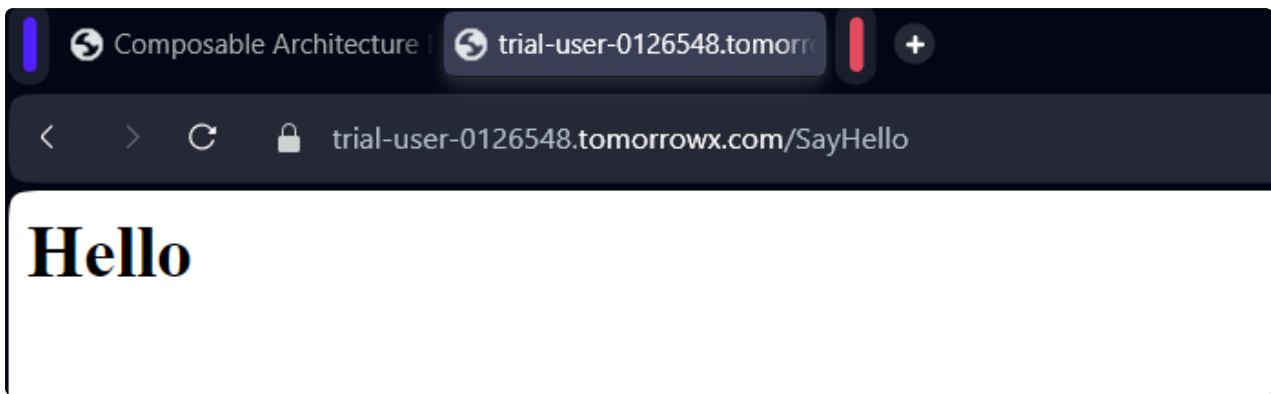
Click the link to open <http://localhost/hello.html> in a new browser tab:



Hello World - Content file -

You'll see a simple html content file called `hello.html` has already been **pre-deployed** and is **served** up to the browser by the running **X Engine**.

Go ahead and enter your name in the form and press the **Say Hello** button. The form submission responds with `Hello`.



SayHello Response

Background Information

The X Engine loads `hello.html`, prompting the user to enter a name and to click a button labelled **Say Hello**. When the button is clicked, the text entered should be appended to **"Hello"**. For example, if the text entered is **"World!"** then the result will be **"Hello World!"**

Objective

The user experience needs improving because any text entered is currently ignored. Can you follow this guide to improve the user experience?

First up, let's go and see where the `hello.html` file lives....

Login [console]

Login to the console using the default credentials. In your case if you are working on the localhost console, use the default credentials: <http://localhost/console/>

- *User ID:* admin
- *Password:* admin

Open Repositories

Once logged in, press **Start** followed by **Repositories**.

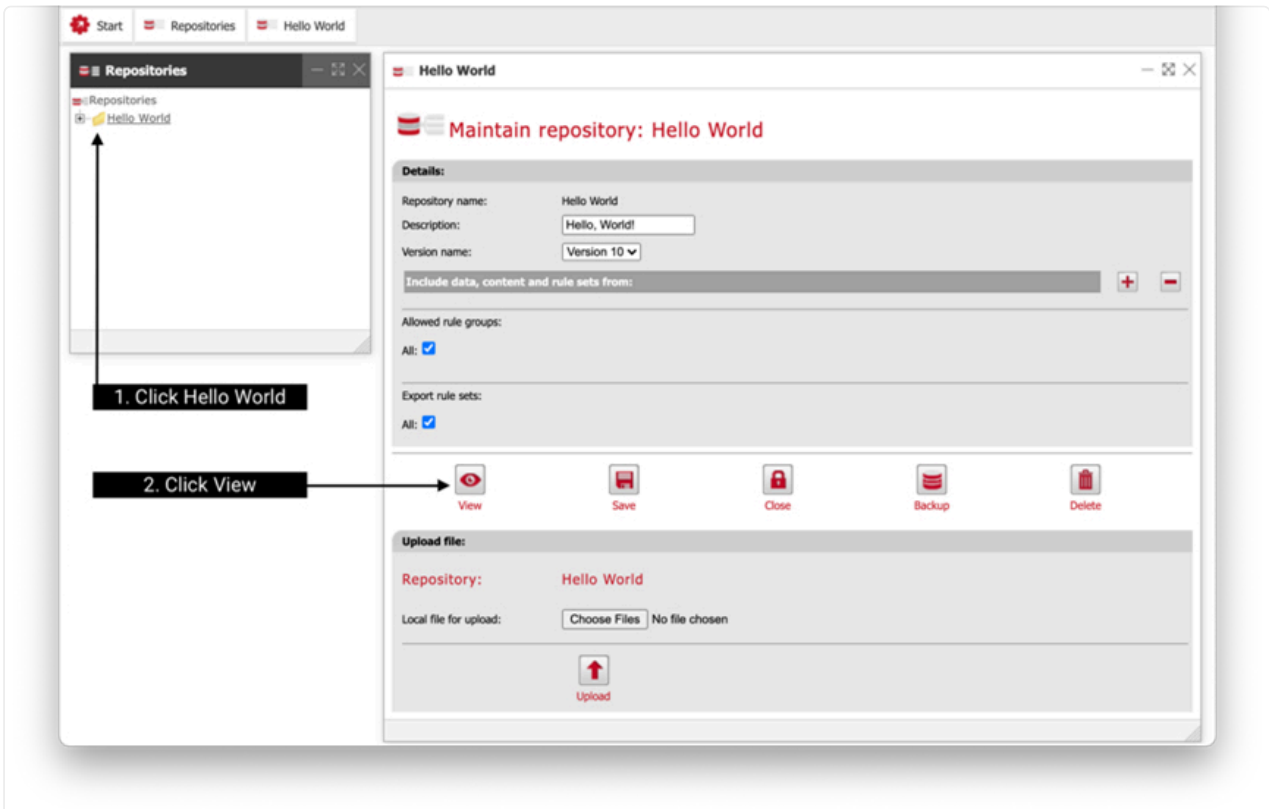
We typically call these “**repos**”. It’s the home, or workspace in the console of where your work lives.



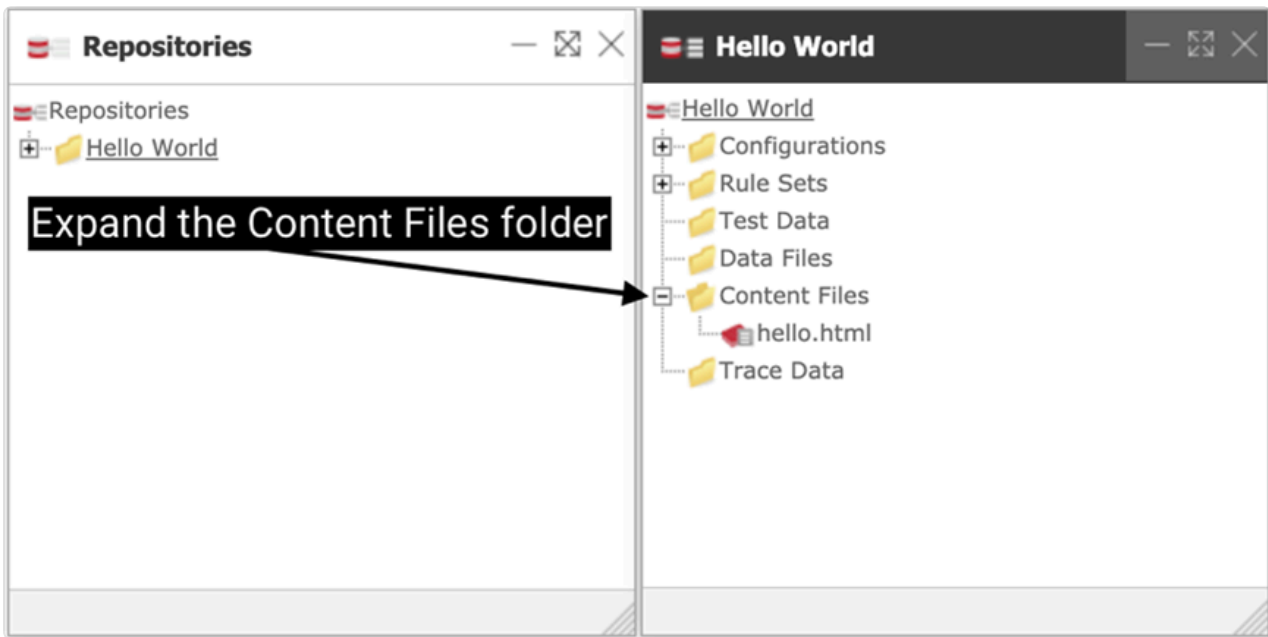
Navigate to Repositories

Now Click on the **Hello World** repository folder (no need to expand the folder tree just now, as that’s where you can save and restore your repository backups – we’ll get to that soon enough!).

Now press **View** and then expand the **Content Files** folder.



Hello World repo



Hello World Content Files - navigation -

Content files can be **HTML, XML, images, or any other binary content** that may be required to be served when requested.

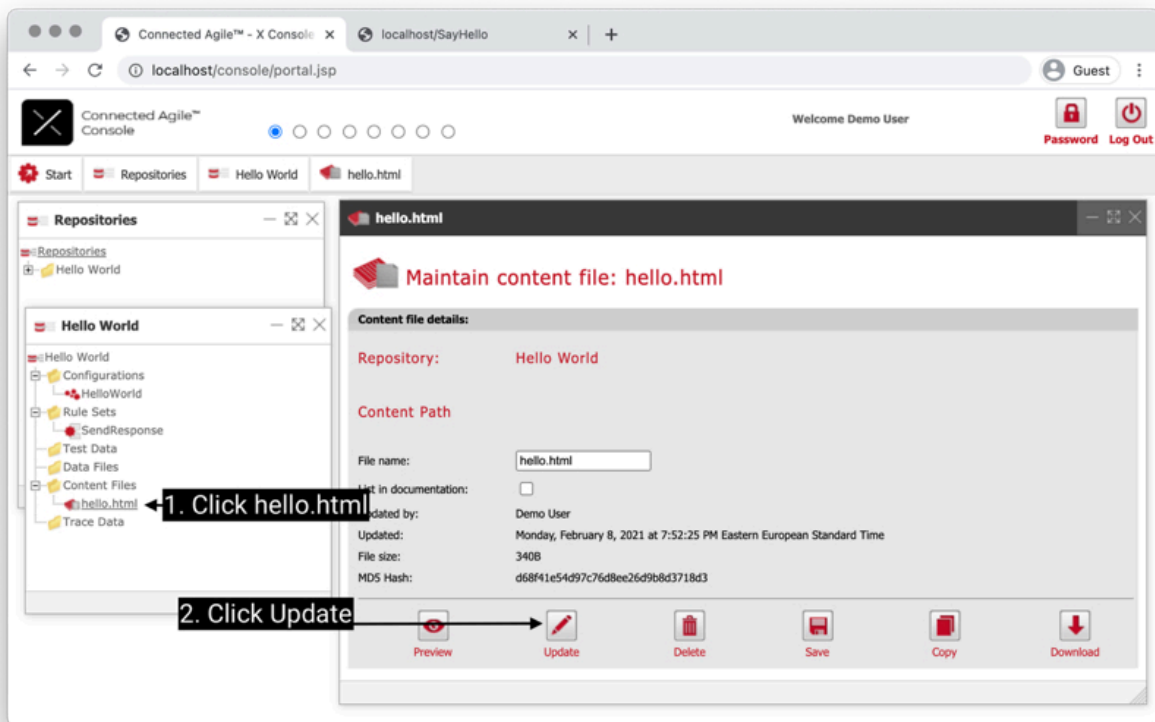
Content files can also be **dynamically modified** by **content rule sets**, we're not covering those in this example. Content files live within a content path that must map to the content path of the application. In our simple example, `hello.html` is served under localhost being the root directory, so therefore **it resides in the top-level Content Files folder**.

<http://localhost/hello.html>

As the Hello World configuration has already been deployed from the console to the target server **X Engine**, this is why the page loads when requested.

Update Content Files

So, let's inspect the html file. Click on **hello.html**, and a new portal window will open for the file. Click on the **Update** button as follows.



Updating the hello.html

A new browser window opens to show a html editor for the **hello.html** content file. Note the input parameter name on the form is set to **Name**. We don't need to make any changes to the html file so you can close this window.



The screenshot shows a web browser window with the address bar containing the URL: localhost/console/contenteditor.jsp?session=node01rvqyychkjxo9wezppprz9t4414&cookie=node01rvqyycc... The browser displays a content editor interface with a 'Save' button and a list of lines of HTML code. The code is as follows:

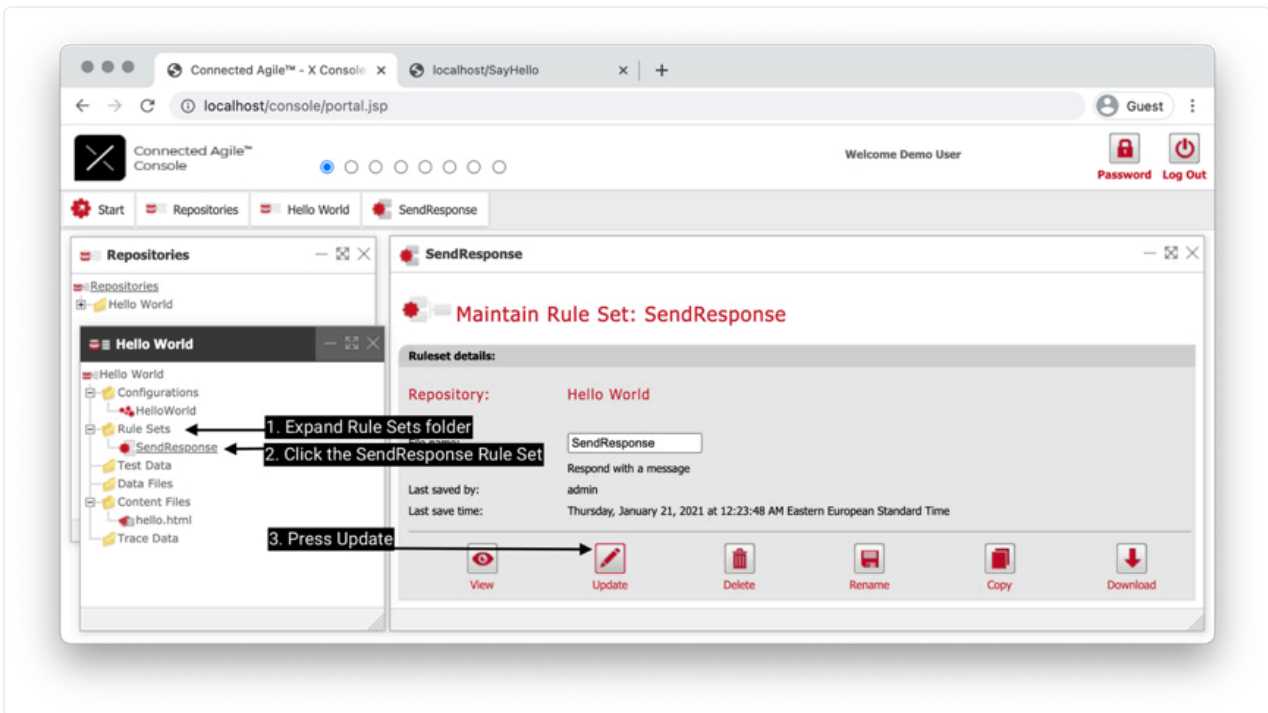
```
1- <html>
2-   <head>
3-     <title>Hello, World!</title>
4-   </head>
5-   <body>
6-     <h1>Please enter your name</h1><p>
7-       <form action="SayHello" method="post">
8-         <input type="text" name="Name"><p>
9-         <button type="submit">Say Hello</button>
10-       </form>
11-     </body>
12-   </html>
```

hello.html content file

So that's a small introduction to **Content Files**. Next, let's take a look at **Rule Sets**.

SendResponse [rule set]

With the Hello World repository open, expand the **Rule Sets** folder, then click the **SendResponse** rule set and press **Update** in the portal window that opens.



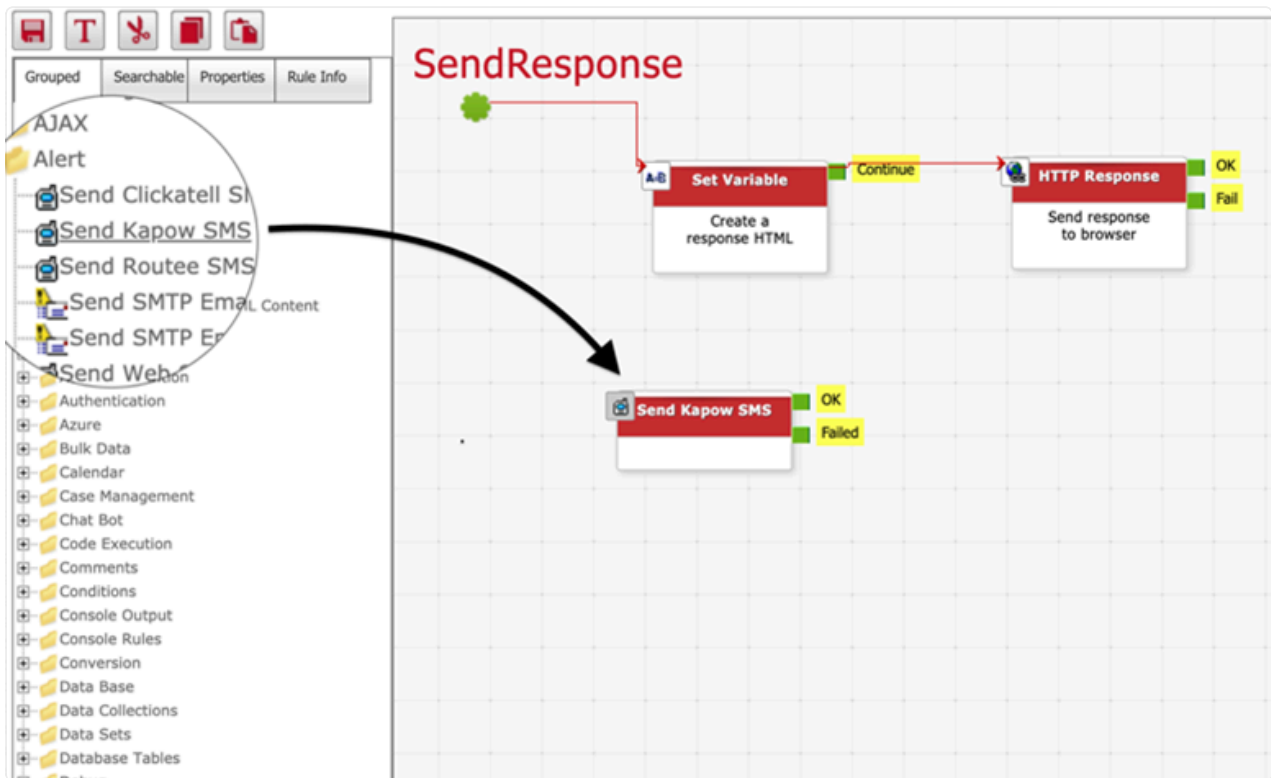
Update a Rule Set

The rules editor is the graphical design tool for composing and maintaining rule sets. The rules editor is launched as a separate browser window from within the console application when you press **Update**.

Rules Editor – *example for reference only*

Go ahead and browse the vast catalogue of what we describe as “**digital blocks**” on the left-hand side. The catalogue is grouped into collections. To use any block in the catalogue, expand the group folder, then click and drag a block onto the main canvas as shown.

In this example, you can expand the **Alert** group folder and drag the **Send Kapow SMS** block onto the canvas.



Send Kapow SMS block

Rules Properties – *example for reference only*

Now click to select the **Send Kapow SMS** block on the canvas, and the left-hand side catalogue will switch to the Properties tab.

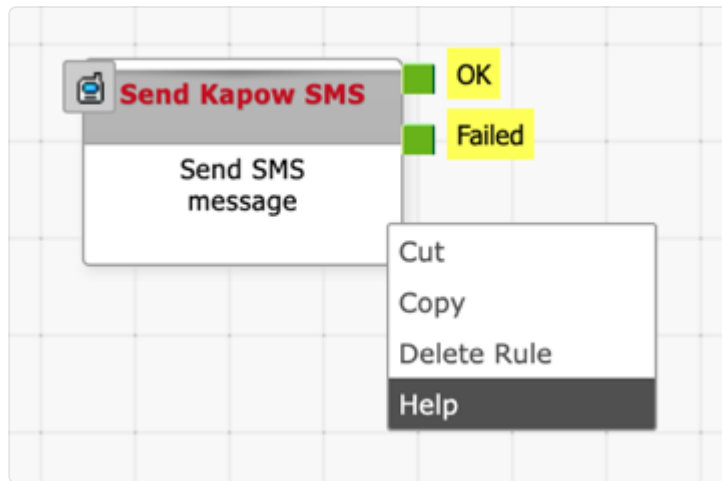
Grouped	Searchable	Properties	Rule Info
Send Kapow SMS			
Label	Send Kapow SMS		
Rule Class	software.tomorrow.rules.rules		
Description	Send SMS message		
Message Variable	MESSAGE		
Phone Number Variable	MOBILE		
Chain Id	OK		
Last connection			
Chain Id	Failed		
Last connection			

Properties tab for a rule

Each block has properties you need to set when composing, along with adding a more meaningful description (like adding comments in code).

In this example you can set the properties as two variables called **MESSAGE** and **MOBILE**. The properties of this the block requires these in order to perform its intended function. These variables would need to contain the values of the SMS message, and the phone number to send the SMS message to.

Everything else is taken care of.



Checking the help section for a Rule

Each block has additional online help you can access by right-clicking over the selected block and pressing **Help**.

Give it a try.

Set Variable

So, let's get back to our example. Click to select the first block called **Set Variable** and view its **Properties**.

Selected blocks banner colour turns grey.

Grouped	Searchable	Properties	Rule Info
Set Variable			
Label	Set Variable		
Rule Class	software.tomorrow.rules.rulelet		
Description	Create a response HTML		
Variable Name	RESPONSE		
Value	<pre><html><body><h1>Hello +NAME+</h1></body></html></pre>		
<input type="button" value="+"/>			
Chain Id	Continue		
Last connection	HTTP Response		

Set Variable block

The block does exactly what it says on the tin. It sets a new variable. In this example we've set the variable name to **RESPONSE**. With the value set to a snippet of html code.

We enclose this snippet in quotes.

```
"<html><body><h1>Hello "+NAME+"</h1></body></html>"
```

Note how this value has been constructed in three parts.

```
"STRING"+VAR+"STRING"
```

You'll remember from earlier, the form submission responds with **"Hello"**, that's because the **NAME** value hasn't been defined or "passed into" this rule so therefore it processes **NAME** as a blank value, so the value of **RESPONSE** would look like this on exit.

```
"<html><body><h1>Hello </h1></body></html>"
```

HTTP Response

Click to select the second block called **HTTP Response** and inspect the **Properties**. Selected blocks banner colour turns grey.

You can also *COPY/ CUT / DELETE / PAST* block(s) with a simple right click.

How easy is that?!

The screenshot displays a rule editor interface. On the left is a properties panel for an 'HTTP Response' block. The panel includes fields for Label, Rule Class, Description, Response Data, HTTP Status code, Content Type, Optional Response, Override Header V, Override Header F, Compress respons, Response data eni, Chain Id, and Last connection. The 'Response Data' field is set to 'RESPONSE', 'HTTP Status code' is '200', and 'Content Type' is 'text/html; charset=utf-8'. The 'Chain Id' is set to 'OK' and 'Fail'. On the right is a workflow diagram titled 'SendResponse'. It shows a sequence of blocks: a green star icon, a 'Set Variable' block (red) with the description 'Create a response HTML', a 'Continue' block (yellow), and an 'HTTP Response' block (grey) with the description 'Send response to browser'. The 'HTTP Response' block is selected, indicated by a grey banner. The workflow ends with 'OK' and 'Fail' outcomes.

HTTP Response block Properties

Guess what!?

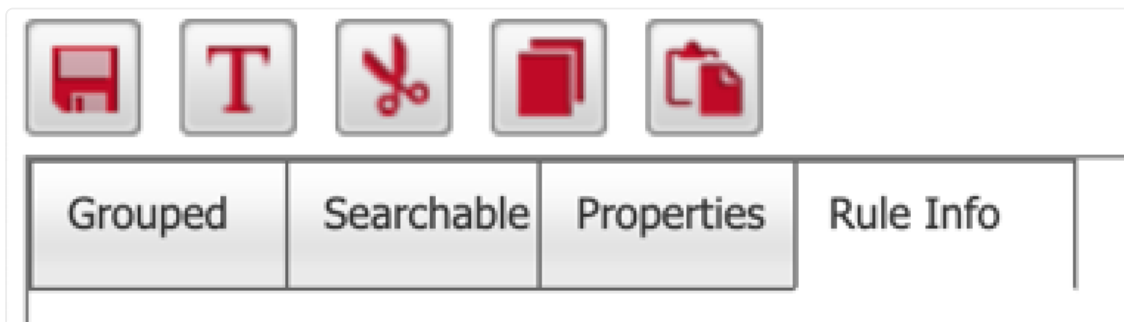
This block *also* does exactly what it says on the tin. It responds to an http request with content of the response data that has been set in the property. In this case the variable **RESPONSE** is the html snippet value set in the preceding **Set Variable** block.

You'll see this block also requires an **HTTP Status code** and **Content Type** set.

This rule performs the final response behavior by the **X Engine** you've already experienced when you clicked the <http://localhost/hello.html> link and pressed the Say Hello button.

Rule Info

Click on the fourth tab called **Rule Info** for the **SendResponse** rule set.



Rule Info Tab

The **Export to Group** and **Short Description** represent this rule set as a new block that can then be (re-)used in other compositions. We will use the **Send Response** rule that lives in the Hello World Grouped folder in the next steps.

Page Description


Respond with a message

Export to group

Hello World


Short Description


Send Response



Description, Export and Short Description inputs

Note it has the **Parameter Type** set to **Input**, **Parameter Name** set to **NAME**, and has been given a **Label** of **Name**.

Parameter Name	NAME
Parameter Label	Name
Parameter Type	Input 
Set values (optional)	



Parameters section

We've finished looking at the **SendResponse** rule set now, so go ahead and close it by closing the Rules Editor window.

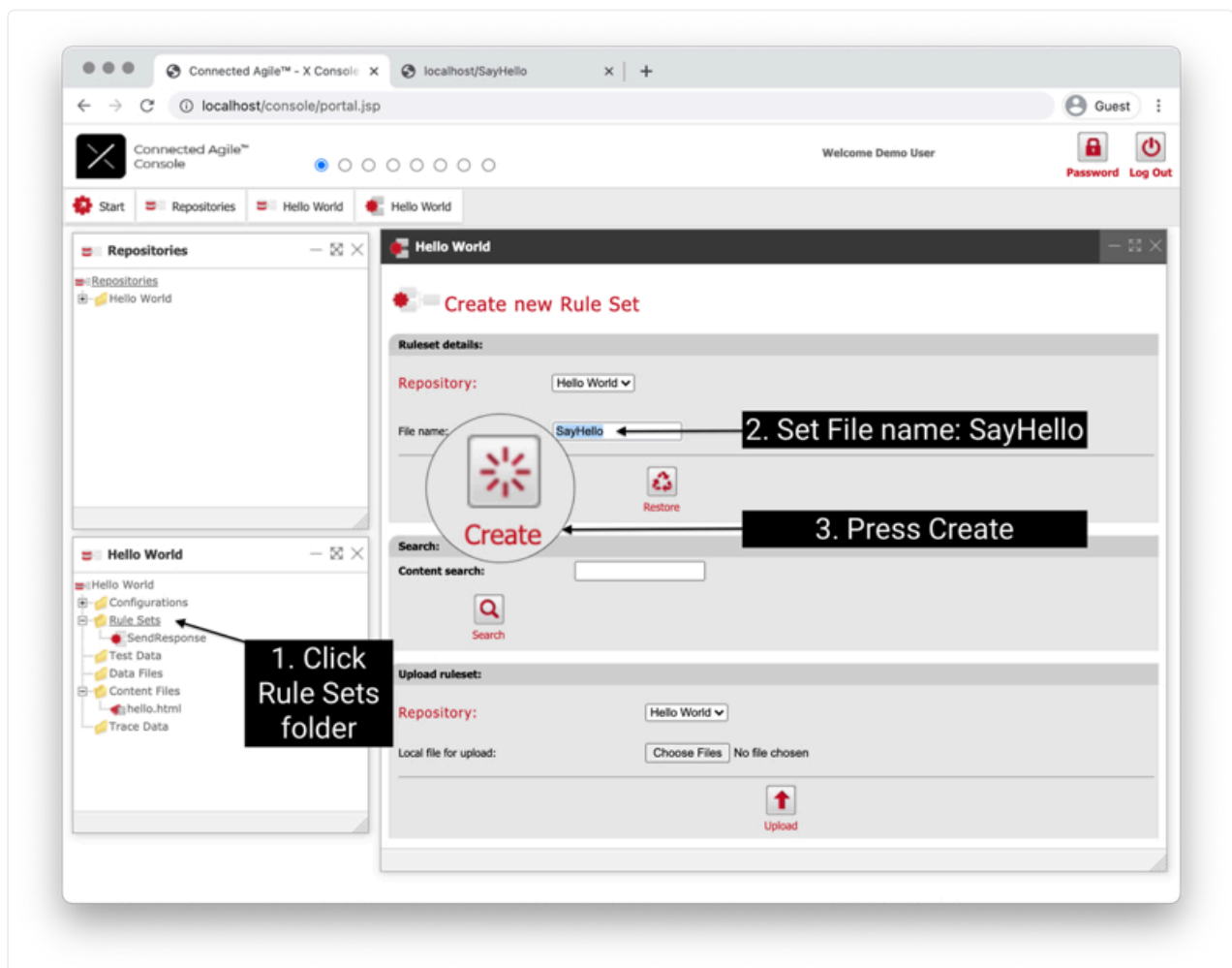
Do NOT save any changes if prompted to do so.

SayHello [rule set]

So, let's create a new rule set that will pass the html form's **Name** value into the response.

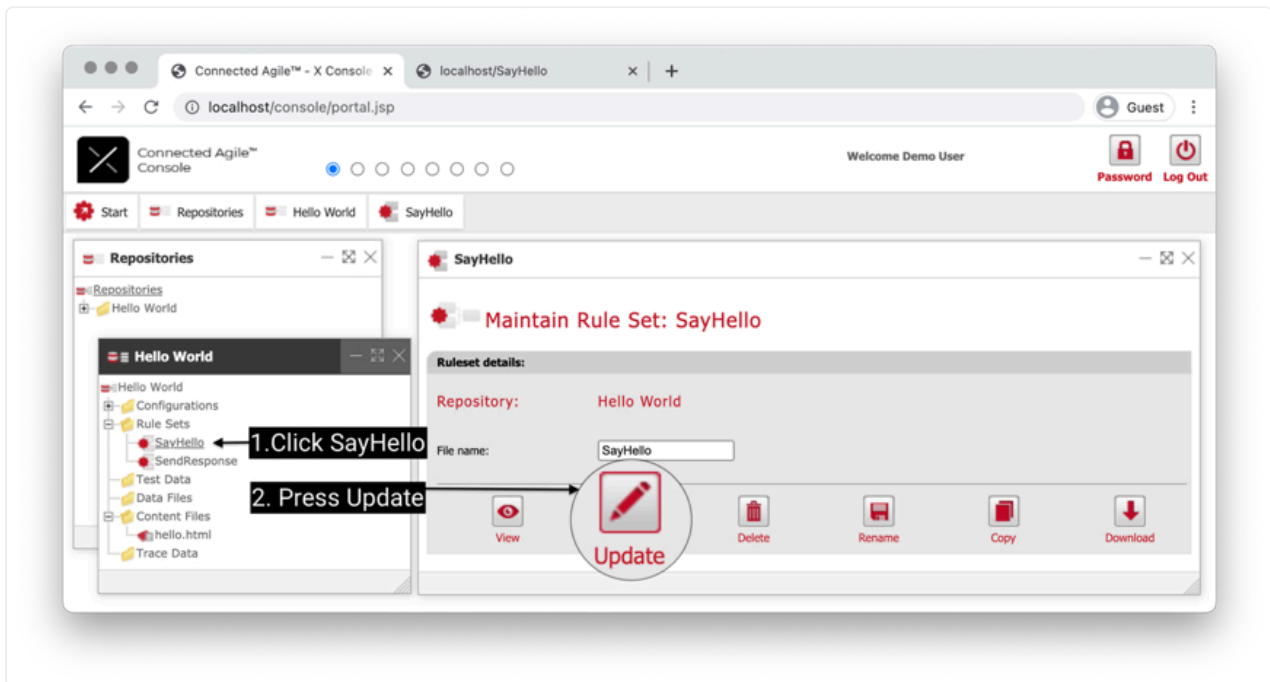
Create a new rule set

Click on the **Rule Sets** folder in the Hello World repository. In the portal window that opens, set the **File Name** to **SayHello** (*case sensitive*) and press the **Create** button.



Create New Rule Set

Now open the newly created **SayHello** rule set for editing. Click on the **SayHello** rule set that has now appeared in the rule set folder of the Hello World repository and press **Update** just as you did to inspect the **SendResponse** rule set.



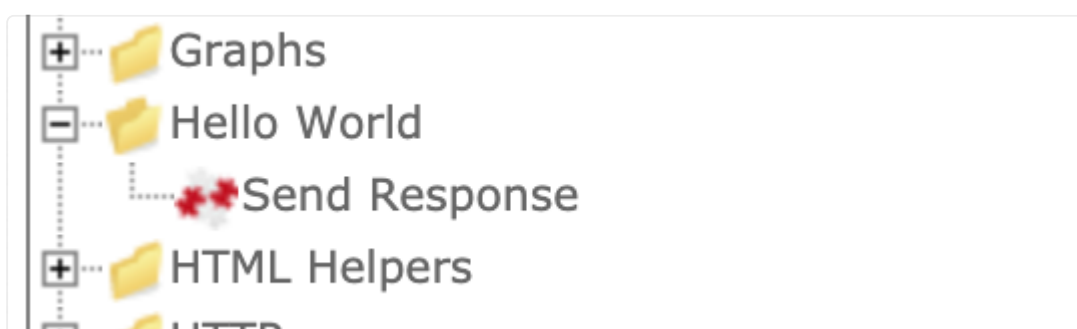
Update the newly created Rule Set - SayHello

Search the catalogue

Go to the search tab and search for “**Response**” and drag the **Send Response** block onto the rules editor canvas.

Alternatively, you can find the same block in the catalogue from the first **Grouped** tab, located in the **Hello World** group folder. This is because the **Rule Info** tab of the **SendResponse** rule set has an export group defined as Hello World.

Either method is fine to search the catalogue and drag blocks onto the canvas.



Wire blocks together

Click on the **Send Response** block

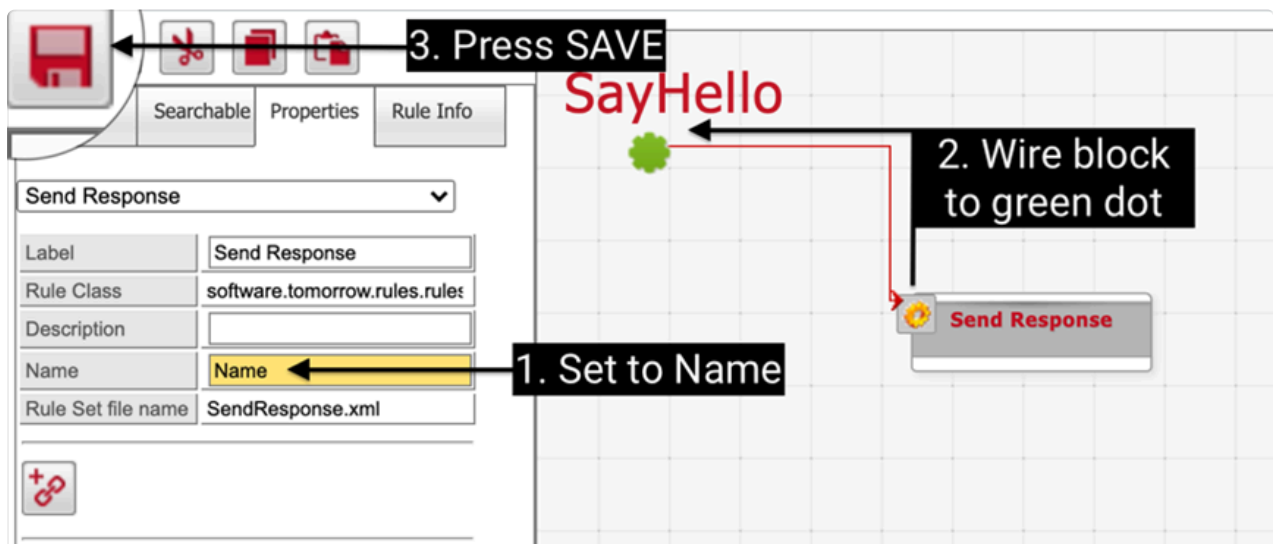
yes, we've turned a rule set into a new block in the catalogue for re-use

and once again just set the properties. So, now set the Name property to **Name** (*case sensitive, no quotes*).

Remembering this was the input parameter set in the **hello.html** content file we looked at earlier.

Click and hold over the orange cog, then click-release over the green dot to "wire" the first block into the rule set in a right to left direction. Incidentally, all subsequent blocks are wired from the block exit chain point (*right hand side*) to the input of the next block (*left hand side*).

Press **SAVE** and close the rules editor window as shown.



Save the Rule Set

That's all you need for your new rule set.

HelloWorld [configuration]

The **HelloWorld** configuration defines the input into the **X Engine** and the rule sets to run.

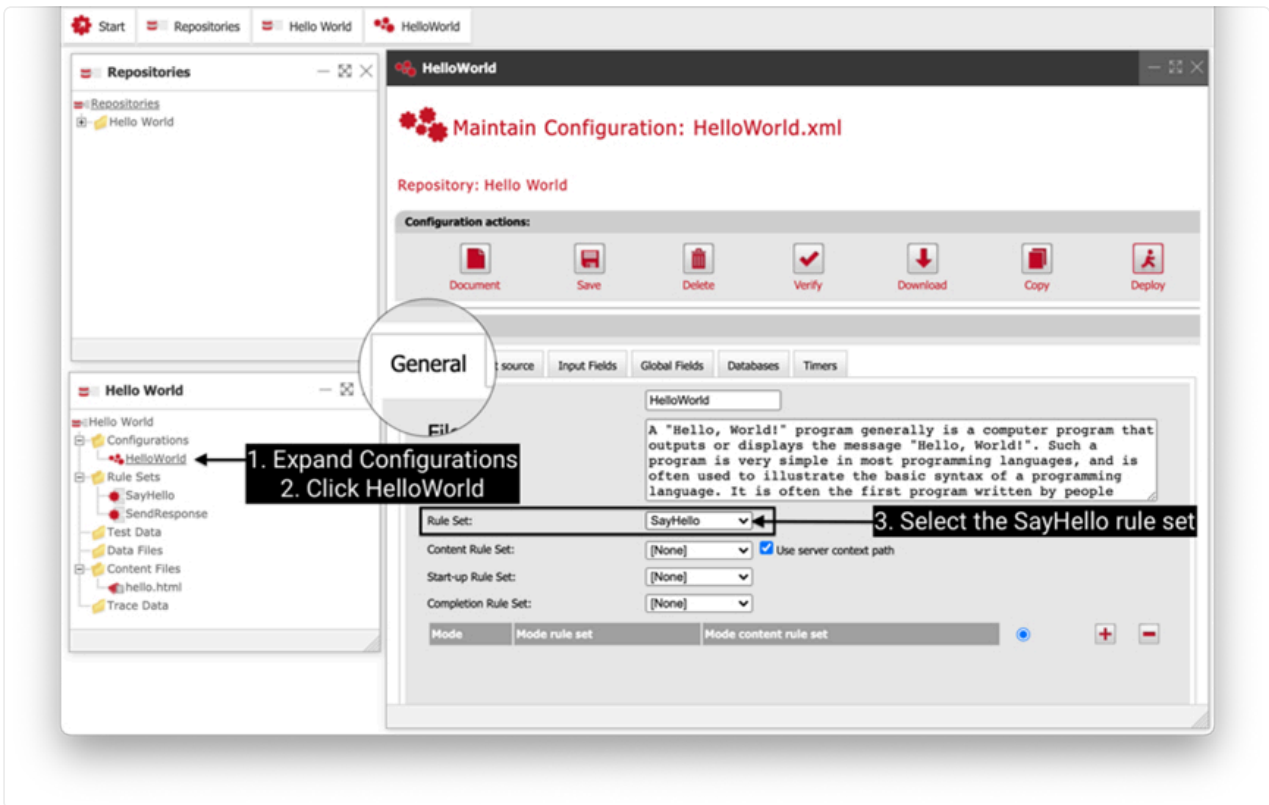
General tab

Expand the **configurations** folder and click the **HelloWorld** configuration. The General tab is the default view, and ensure you now select the **SayHello** rule set from the dropdown list of available rule sets. This is the “**initialising**” rule set that is processed by the X Engine on the very first transaction it receives.

Embedded (*dependent*) rule sets that have been wired within the SayHello rule set are also deployed along with it's parent, so you only need to set the top-level ruleset.

Therefore, any dependent rule sets will get deployed along with the configuration without having to define them.

You'll note here that there are three other types of rule set that can be set to initialize and run when processing data. These are for (1) **CONTENT**, on (2) **STARTUP**, and on (3) **COMPLETION**. These are not required in this example.



Specifying the initial Rule Set in our Hello World Configurations

Timers tab – information for reference only

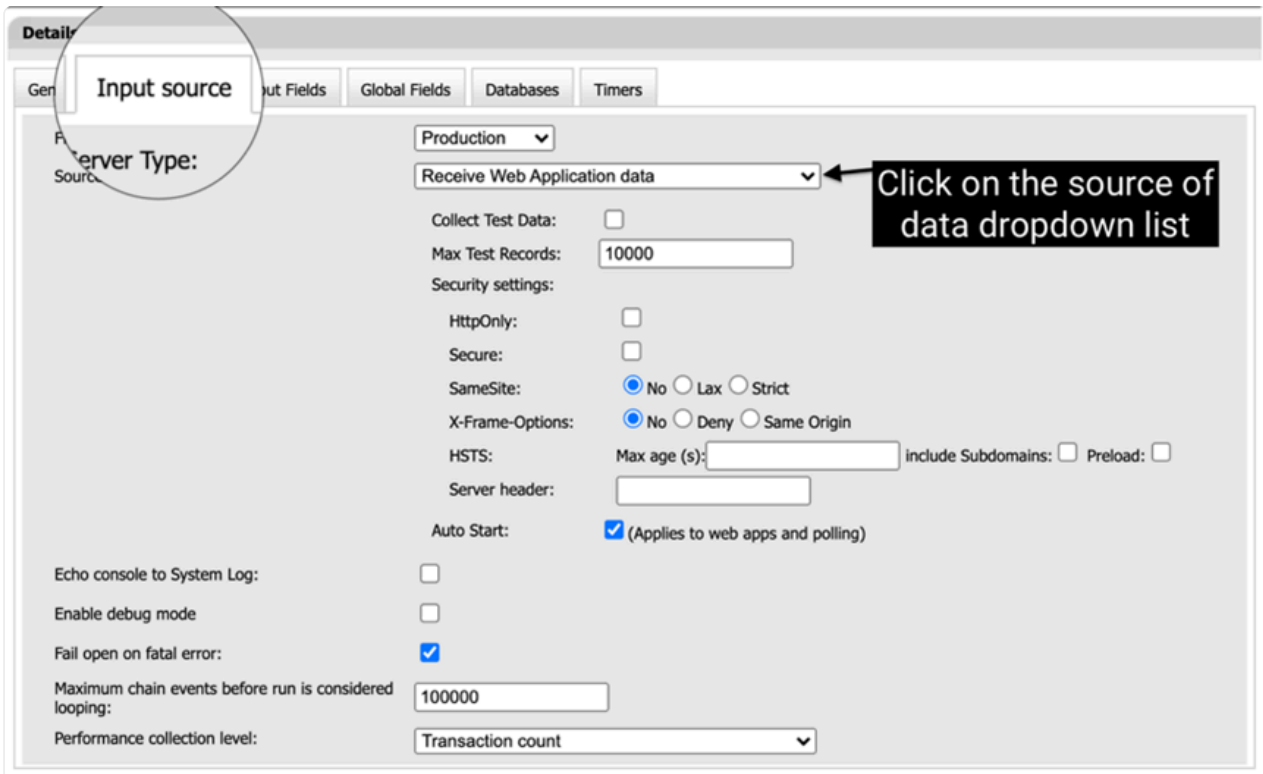
Just to mention in passing, there is a fifth rule set you can set in the **Timers** tab of the configuration. These are rule sets that are initiated and run (*as the name suggests*) on a timed basis. For example, when a rule set is required to perform a defined process say, every 24 hours.



Timers Tab

Input source tab

Click on the Input Source tab and inspect the different sources of data options available.



Input Source Tab

For this example, we are configuring the X Engine to process web application data, but as you can see this is just one of a multitude of available options to define in the configuration, dependent on the composition and data sources being processed.



Configuring X Engine for web application data

Databases tab – *information for reference only*

Click on the **Databases** tab. It's here where you define the databases being made available to the X Engine. You are not required to define a database for this example so there's no need to configure a database.

Example only:

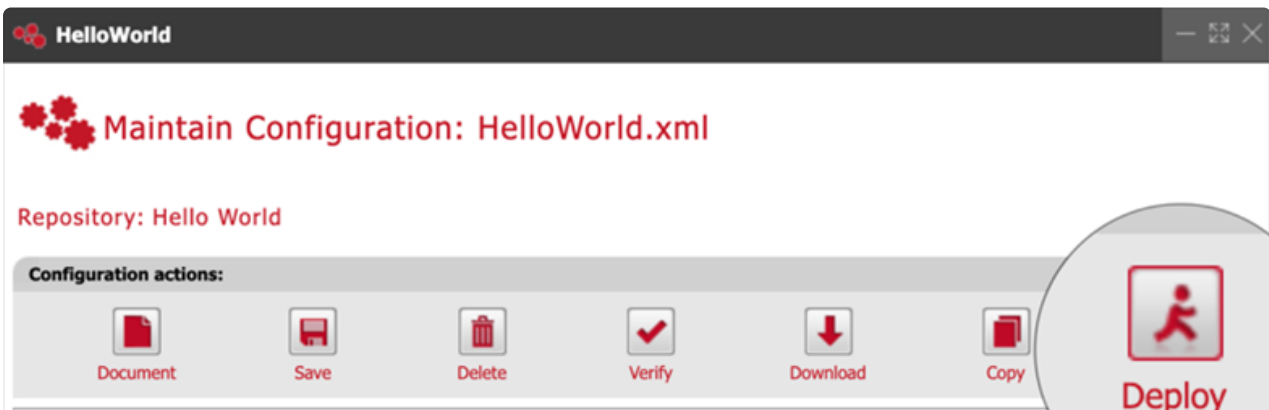


Databases Tab

If you are interested, database connectivity specifying JDBC driver, connection string and schema credentials is an administrator set-up task in the console. You don't need to complete that right now.

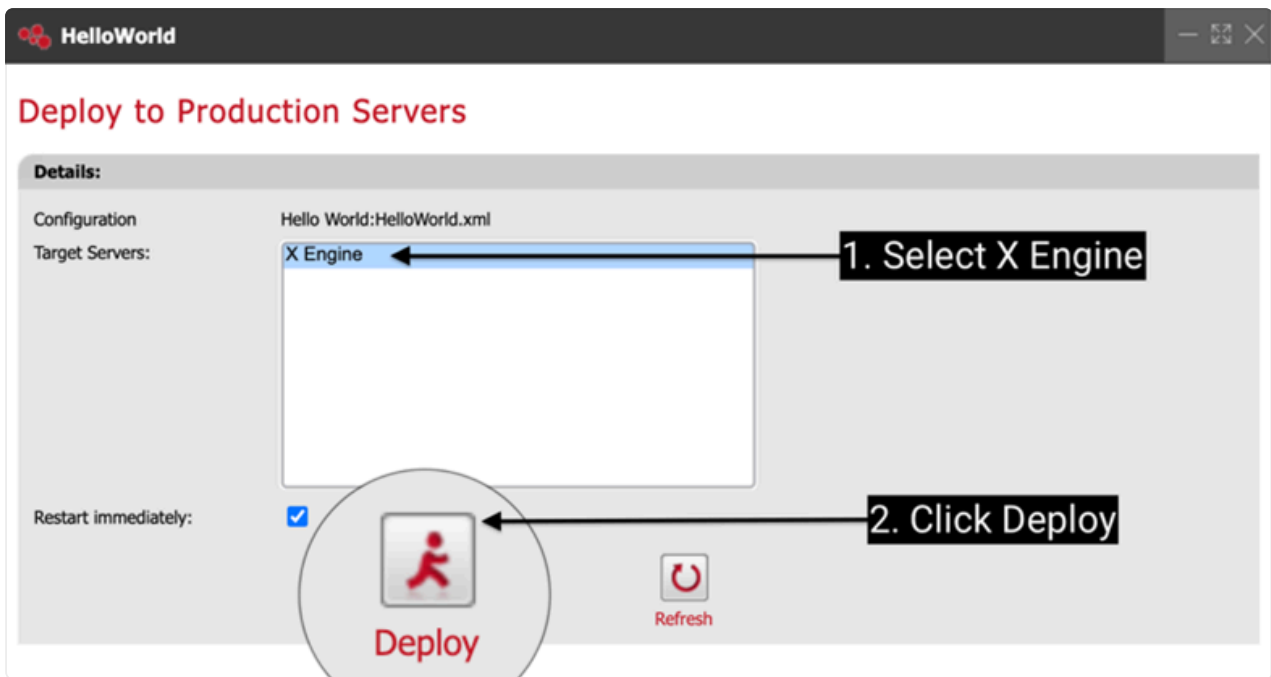
Deploy

With the new **SayHello** rule set defined as the rule set in the configuration, you can go ahead and press the **Deploy** button.



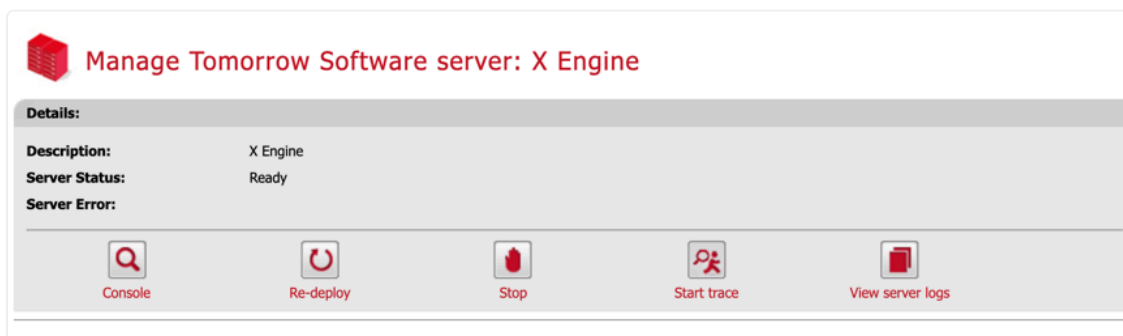
Deploy our repo

Select **X Engine** as the target server and press the **Deploy** button.



Specifying the Target Server and deploying

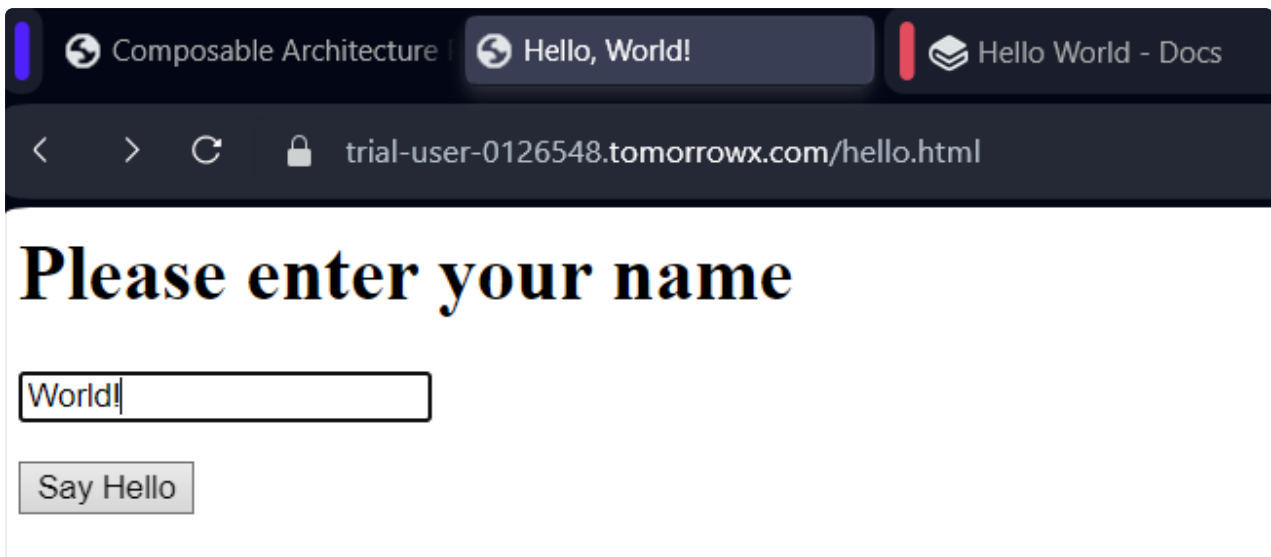
Wait for the deployment to complete and the server restart in a few seconds, and you'll see the X Engine server details are shown.



X Engine details

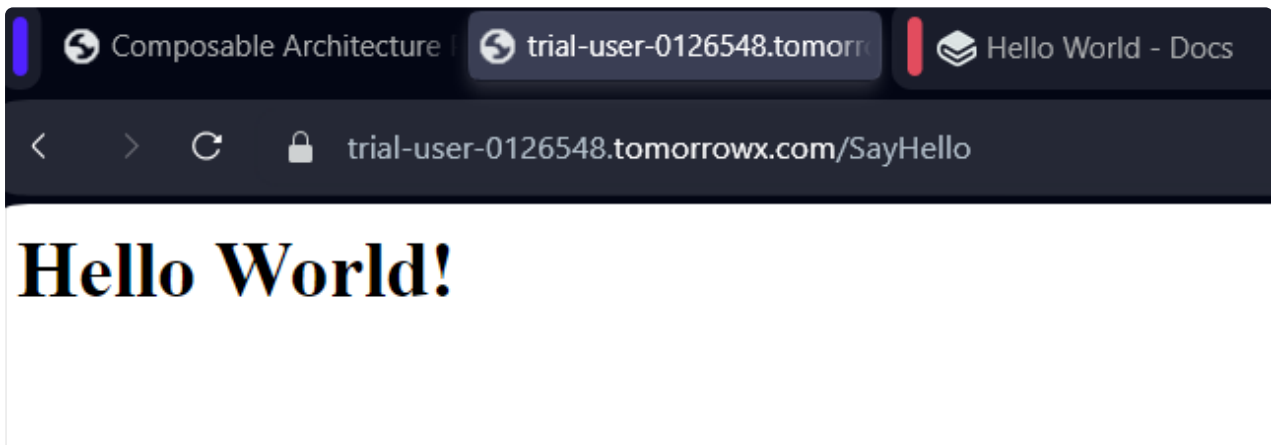
Test

Click the link to open <http://localhost/hello.html> in a new browser tab and refresh the page.



hello world form

Enter **World!** the press the **Say Hello** button, and if successful you'll receive a **Hello World!** response.



SayHello Response

[the crowd erupts into wild applause 🙌🍷]

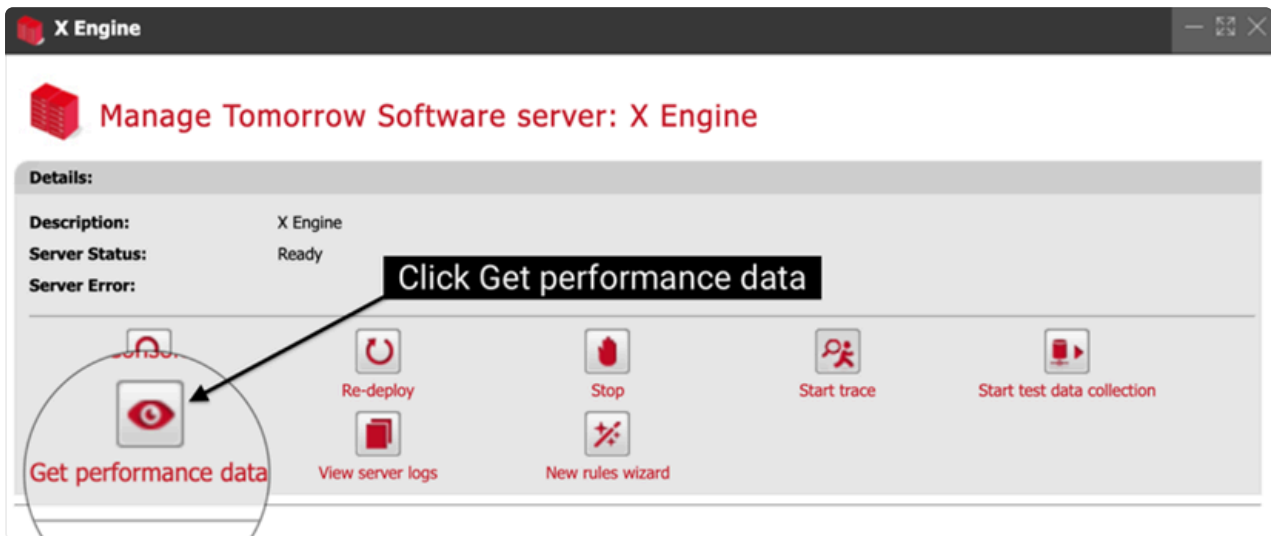
Want some more?

Then read on....

Performance data and live probes

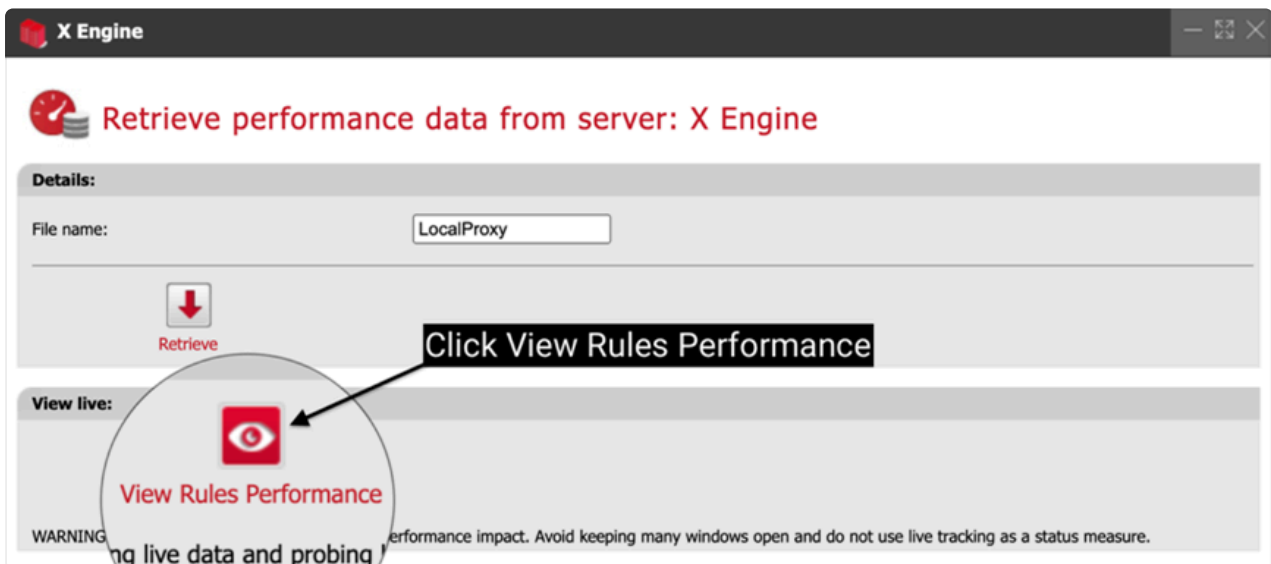
With the Hello, World! example now working successfully, let's give you a glimpse under the hood of the X Engine.

Go back to the console and click **Get performance data** in the X Engine server portal window you have open.



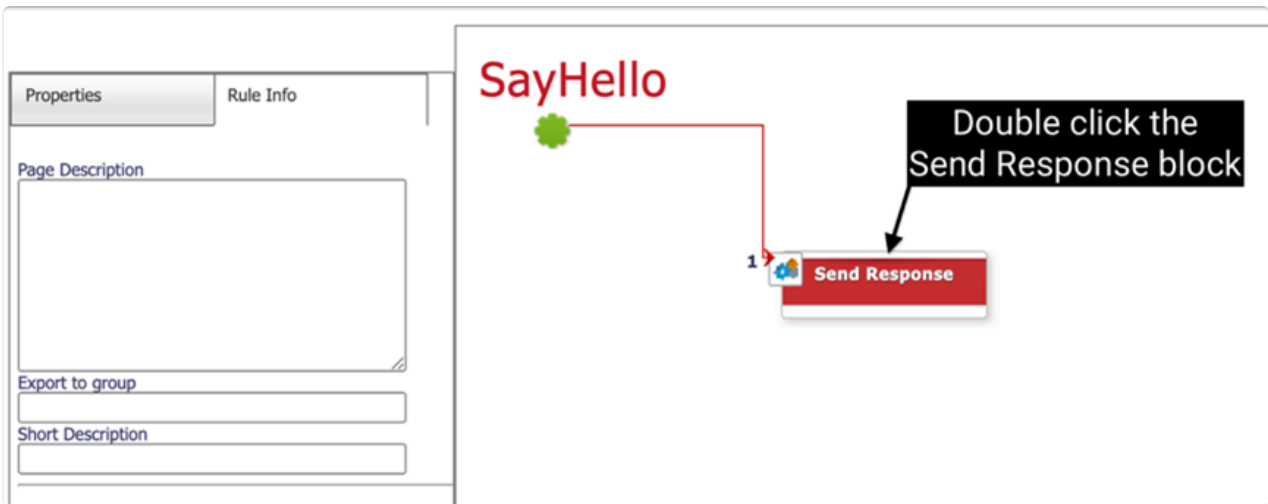
Get Performance Data for X Engine

On the next window click **View Rules Performance**



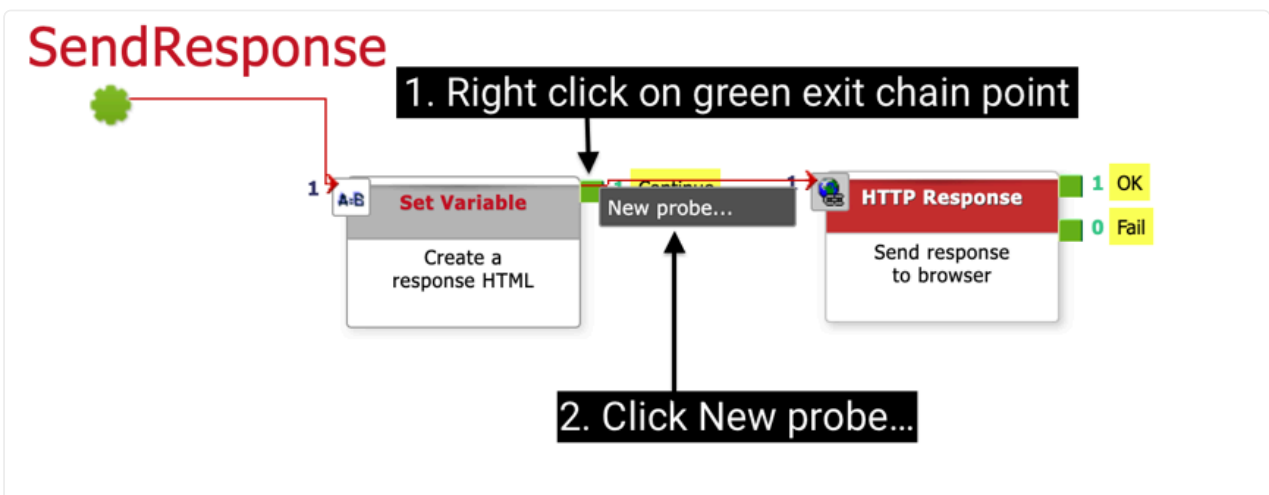
View Rules Performance

The rules editor window opens in a new window. Double click the **Send Response** block.



SayHello Send Response block

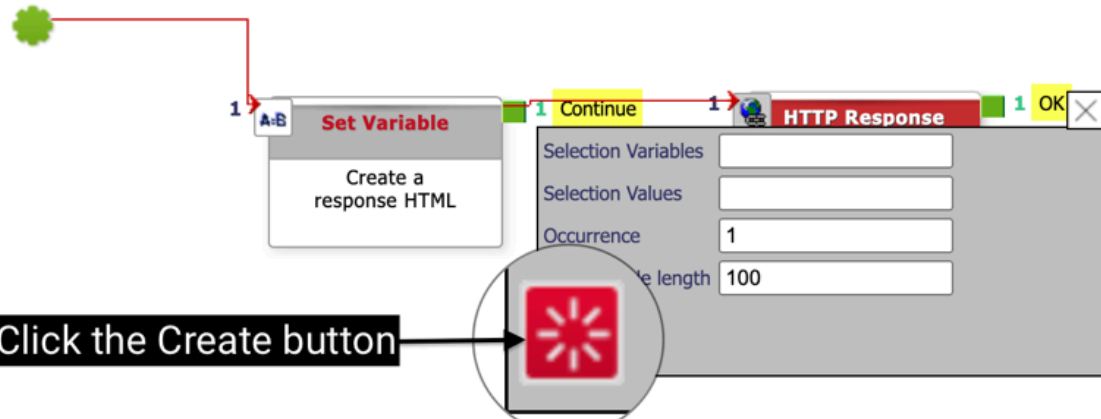
Place a probe on the **Set Variable** block. Right click over the green exit chain point and click **New probe...**



New probe for SendResponse block

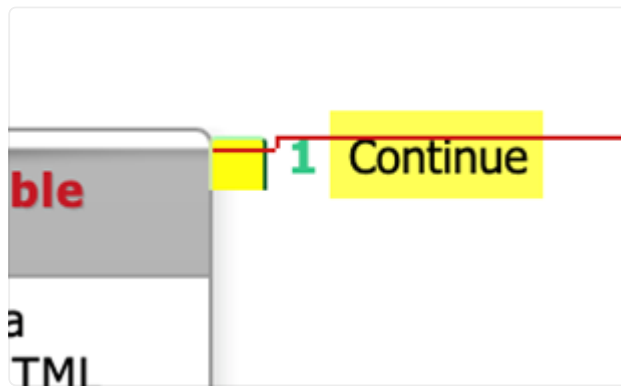
Click the **Create** button. Live probes are triggered by variables and values, and occurrences thereof. We can leave these blank to just trigger on the next transaction.

SendResponse



Create New probe

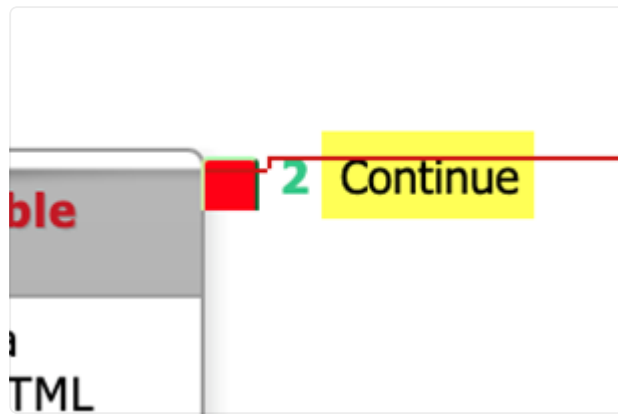
The exit chain point turns yellow to show the probe is set.



Yellow probe

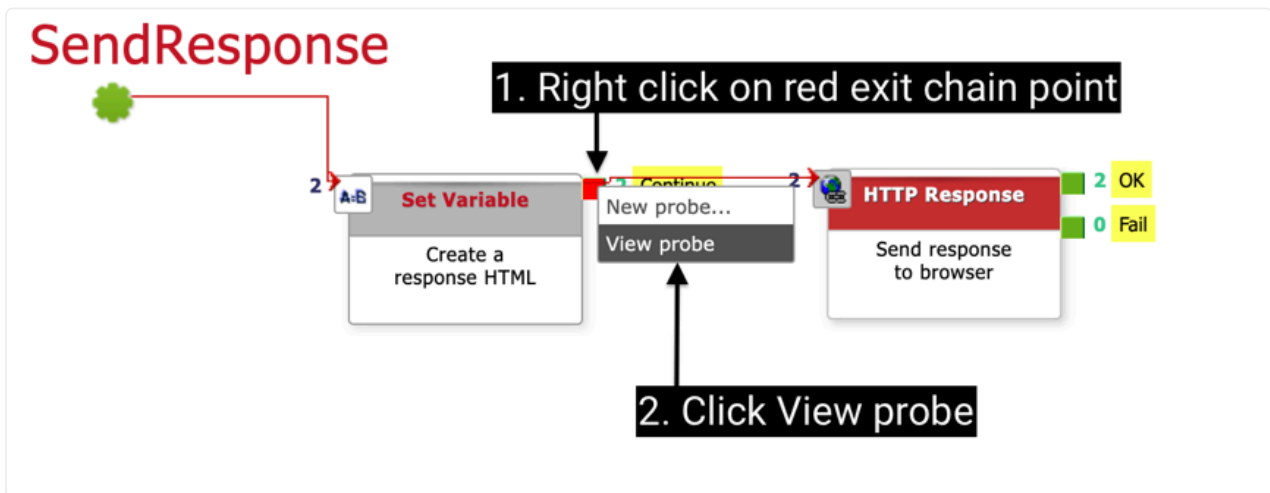
Now go to the browser tab of the demo page showing the SayHello output and click the back button so that the input hello.html page shows.

Input a new name **Probe** into the input field and click **Say Hello**, the page responds as expected with **Hello Probe**. Go back to the rules editor window with the probe set and you'll see the exit chain point has turned red to show the probe has been triggered.



Two probes

Right click on the red exit chain point and click **View probe**.



View a probe

You can now see the transaction data that has just been processed by the X Engine. The contents of the two variables **NAME** and **RESPONSE**.

```
[NAME]=[Probe]
[RESPONSE]=[<html><body><h1>Hello Probe</h1></body></html>]
```



Transaction data with the content of the two variables NAME and RESPONSE

Aside from helping you view live data to assist with composing or troubleshooting your solution, it also provides a superior debugging tool that can even be used on production servers without the need for logging.