Ready! API is a  test-driven suite of tools created for the purpose of creating, testing , and automating APIs regardless of development platform.  Products in the suite include but are not limited to:

* [SoapUI Pro:](https://fusion.mastercard.int/confluence/display/NET/SoapUI+Pro)  functional testing for APIs
* [ServiceV Pro:](https://fusion.mastercard.int/confluence/display/NET/ServiceV+Pro%3A+Web+API+Virtualization)  quickly create virtual APIs for user testing and feedback ahead of development
* [LoadUI:](https://fusion.mastercard.int/confluence/display/NET/LoadUI) load testing APIs
* [Secure](https://fusion.mastercard.int/confluence/display/NET/Secure):  provides security tests and functionality to prevent vulnerabilities in APIs.

Please note that there are multiple ways to utilize features in ReadyAPI.  For the purpose of this wiki, utilizing features via the GUI is primary focus; however, scripting languages are often used.

Widely known as a stand-alone application for testing SOAP requests, the old SoapUI has now been incorporated into the ReadyAPI suite of products and can also support more protocols such as REST, IoT, etc.

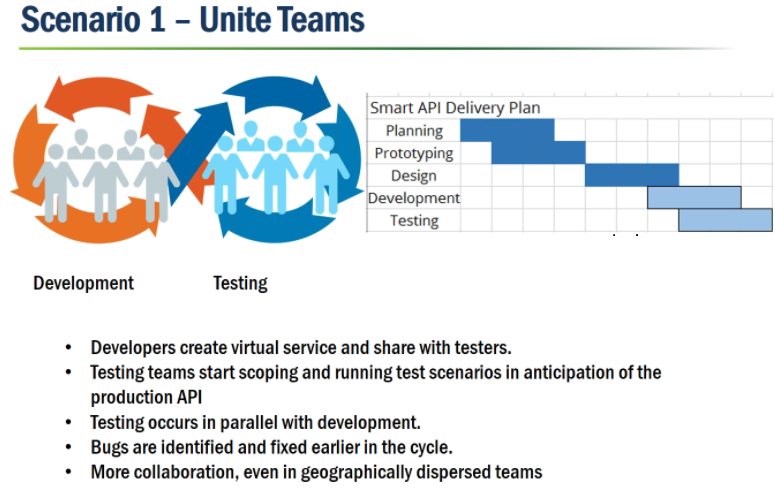
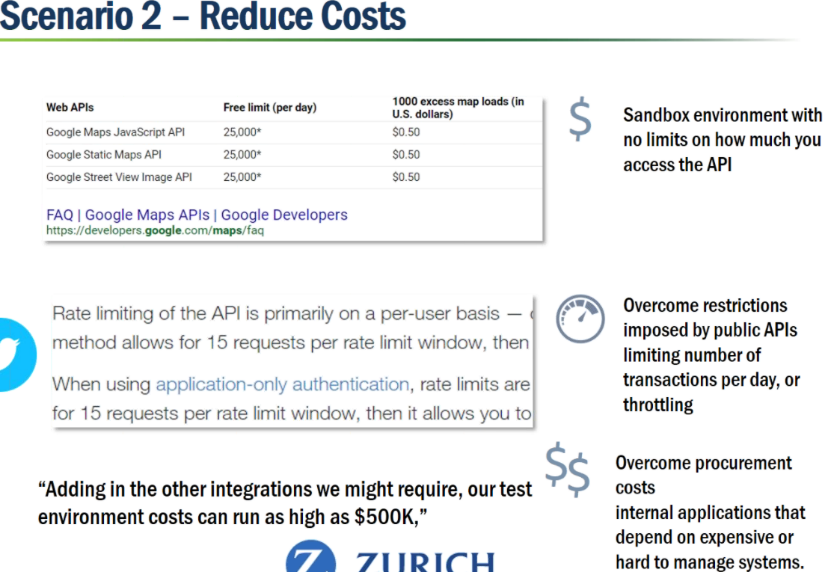
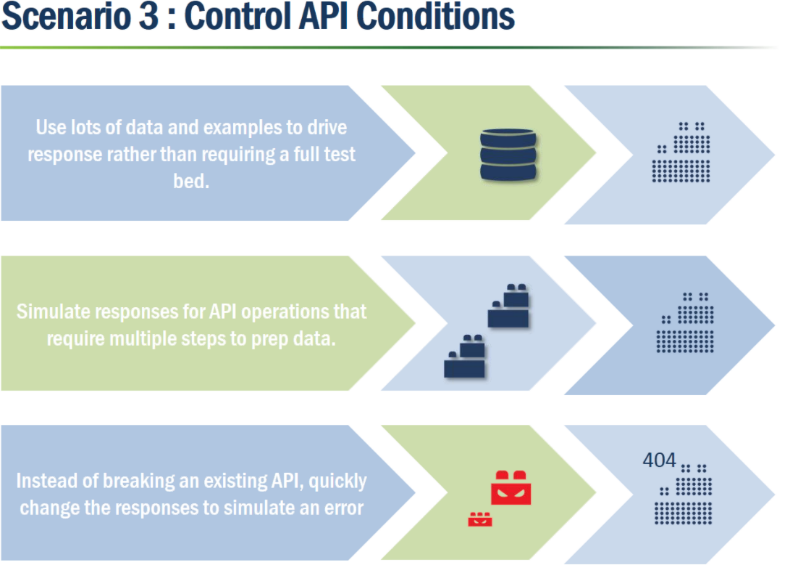
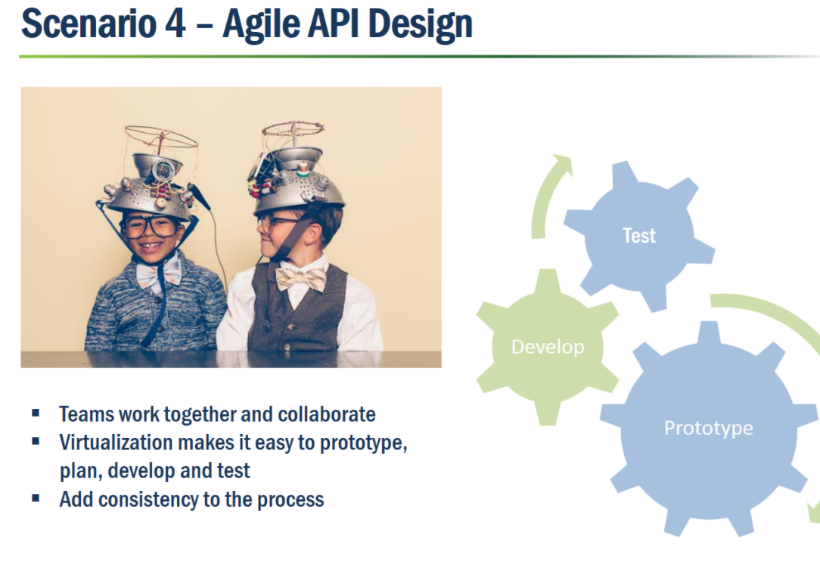
Implementing many options for functional testing, SoapUI structures tests into three levels.  This can be important to keep in mind when considering parameter scoping.

1. A TestSuite is a collection of TestCases that can be used for grouping functional tests into logical units. Any number of TestSuites can be created inside a soapUI project to support massive testing scenarios.
2. A TestCase is a collection of TestSteps that are assembled to test some specific aspect of your service(s). You can add any number of TestCases to a containing TestSuite and even modularize them to call each other for complex testing scenarios.
3. TestSteps are the "building blocks" of functional tests in soapUI. They are added to a TestCase and used control the flow of execution and validate the functionality of the service(s) to be tested.

REFERENCES:

* [Introduction and Features of SoapUI Pro](https://smartbear.com/product/ready-api/soapui/overview/)
* [The structure and setup of SoapUI Tests](https://www.soapui.org/functional-testing/structuring-and-running-tests.html)

**FOUR TENETS OF USING SERVICEV PRO:**

* **Unite Dev and Testing Teams:**
  + Can re-create APIs of others without knowing contract by capturing traffic.
  + Tool will set up a rest service ready for your modifications
  + Can share virts with other teams  
    
* **Reduce Costs:**
  + Can overcome restrictions imposed by public APIs that limit number of transactions per day, or throttling (Google) (Scenario 2)
  + Can test without concerns for limited transactions
  + Can create sandbox environment with no limits on how much you access the API for less $.
  + Can share virtual environments  
    
* **Control API Conditions**
  + Can simulate tests to return 500 error without having them break something to let us get the 500.  
    
* **Agile API Design:  Can easily prototype API at-will.**
  + Ready for users to test before development.  UI developers can develop before API is finished.
  + Integrates Swagger, Postman, Jenkins etc.  
    

**REFERENCES:**

* [Overview](https://smartbear.com/product/ready-api/servicev/overview/)

LoadUI  can reuse functional API tests built in SoapUI Pro to speed up testing and reduce the time it takes you to deploy high performance REST and SOAP web services.

REFERENCES:

* [LoadUI Overview](https://smartbear.com/product/ready-api/loadui/overview/)

Necessary tool that provides security tests and functionality to prevent vulnerabilities in APIs.

REFERENCES:

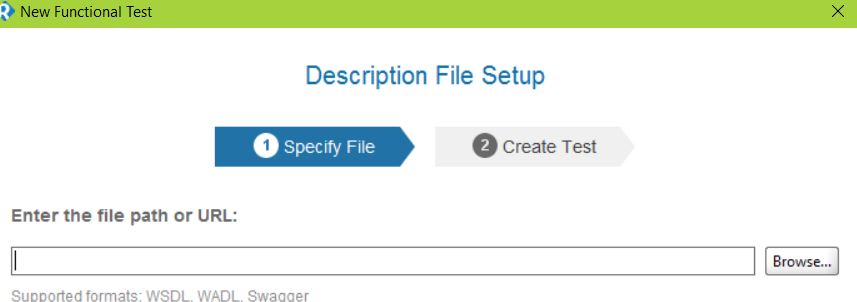
* [Overview](https://support.smartbear.com/readyapi/docs/secure/intro/about.html)

# [SoapUI: Setting up Service Definitions / Importing Swagger or Postman Tests](https://fusion.mastercard.int/confluence/pages/viewpage.action?pageId=227087292)

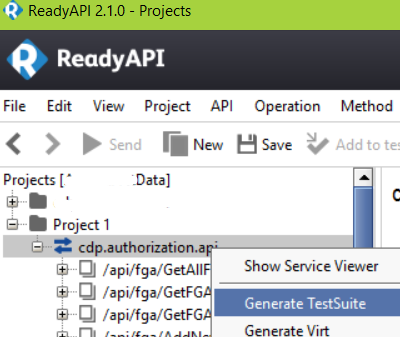
Projects can be created using a wide variety of formats:

|  |  |
| --- | --- |
| * **WSDL** * **WADL** * **URL** * **JSON** * **XML** * **Swagger** | **https://fusion.mastercard.int/confluence/download/attachments/227087292/defs.JPG?version=1&modificationDate=1504211538000&api=v2** |

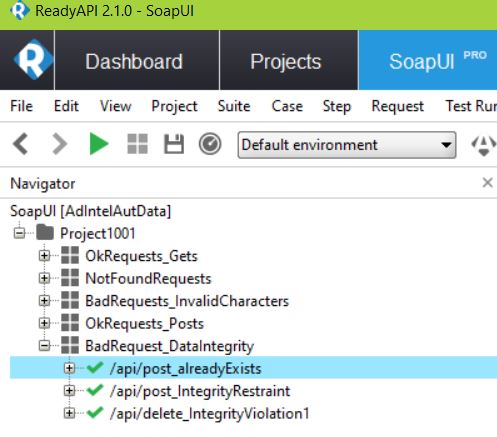
**CREATING A PROJECT FROM SWAGGER**

1. Install the[Swagger plug-in](https://www.soapui.org/extension-plugins/bundled.html) into ReadyAPI.
2. Navigate to toolbar and select File --> New Project.
3. You should then see the image above.  From here you can select to create from a Swagger file.
4. Now you have the option of entering a URL or file path.
   * 

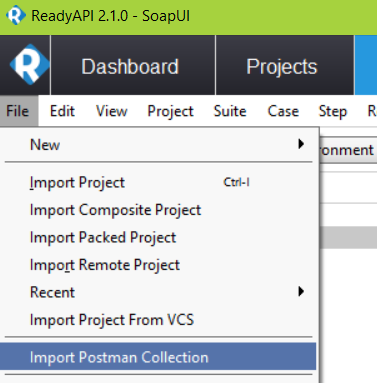
5. Once the document has been imported properly, you will now be able to see all the methods available for you.  These are considered to be the base methods and recommended to keep them as-is.  To create your customized test cases, right-click and select "Generate TestSuite"



6. Now you can create all the test scenarios that you need:



**IMPORTING POSTMAN TESTS**

1. As with Swagger, you will need to ensure that the[Postman plug-in](https://www.soapui.org/extension-plugins/bundled.html)is enabled in ReadyAPI
2. Start Postman and load the desired Postman Collection you wish to use.
3. Select the collection and click "Downlaod" to export.
4. Open ReadyAPI and create a new project.
   * If the plug-in is enabled, you should be able to select File --> Import Postman Collection:  
     

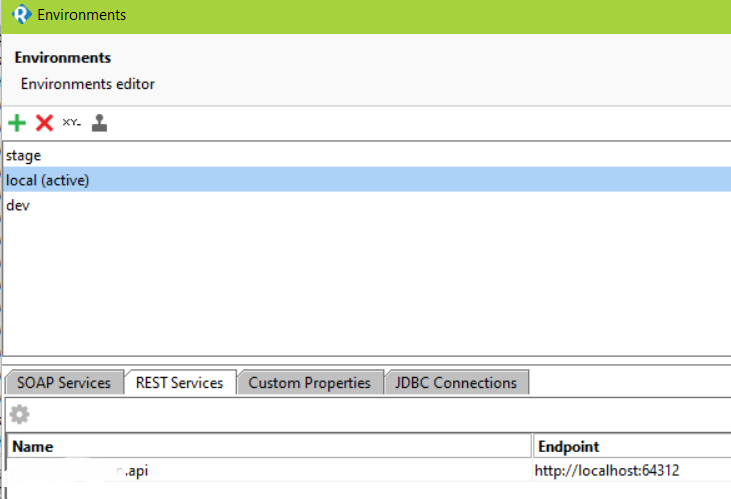
5. From this point, your Postman APIs will be converted to SoapUI APIs.  Please note that you might be prompted to customize your conversion preferences.

**REFERENCES:**

* [Creating Projects](https://support.smartbear.com/readyapi/docs/projects/create/index.html)
* [Creating From Swagger](https://support.smartbear.com/readyapi/docs/projects/create/swagger.html)
* [Creating From Postman Collections](https://www.soapui.org/docs/postman-collections.html)

# [Setting up Environment Properties for Dev and Stage](https://fusion.mastercard.int/confluence/display/NET/Setting+up+Environment+Properties+for+Dev+and+Stage)

ReadyAPI now provides the option to create settings for each of your testing environments.  This will allow you to take the same TestSuite and use it in both dev and test.

1. Open the "Environments" editor
2. From here you can enter endpoints for the environments you use:  
   

### **WARNING:**  Please be cautious when changing environments.  Sometimes, ReadyAPI will cache the current endpoint so that even if it appears that you are running in a development environment, you might still be running in stage.   To prevent this from happening, please close and restart ReadyAPI after changing endpoints.

[Test Data Options](https://fusion.mastercard.int/confluence/display/NET/Data+Options+for+Test+Data)

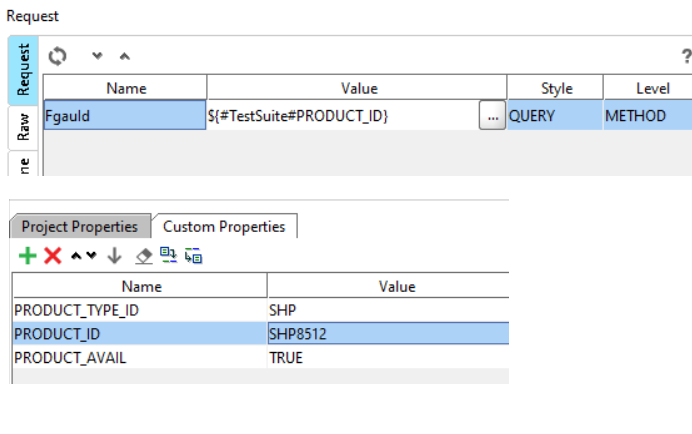
# [Referencing Custom External Properties](https://fusion.mastercard.int/confluence/display/NET/Referencing+Custom+External+Properties)

After setting up external parameters, you will likely want to know how to use these in service requests.  Smartbear provides a common syntax to dynamically insert property values during processing.  Custom properties can be defined and referenced as different levels in SoapUI:

* At the Project, TestSuite and TestCase level in the corresponding Properties tab (see below).
* In a Properties TestStep
* In a DataGen TestStep
* As part of a TestStep configuration:
  + Within a DataSource TestStep
  + Within a DataSink TestStep

A basic syntax is as follows:

${#TestSuite#VARIABLE\_NAME}



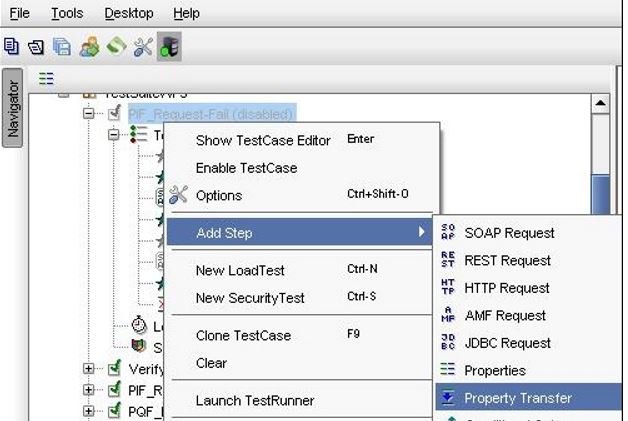
**REFERENCES:**

[Functional Testing](https://fusion.mastercard.int/confluence/download/attachments/227098616/extparam1.png?version=1&modificationDate=1504624236000&api=v2)

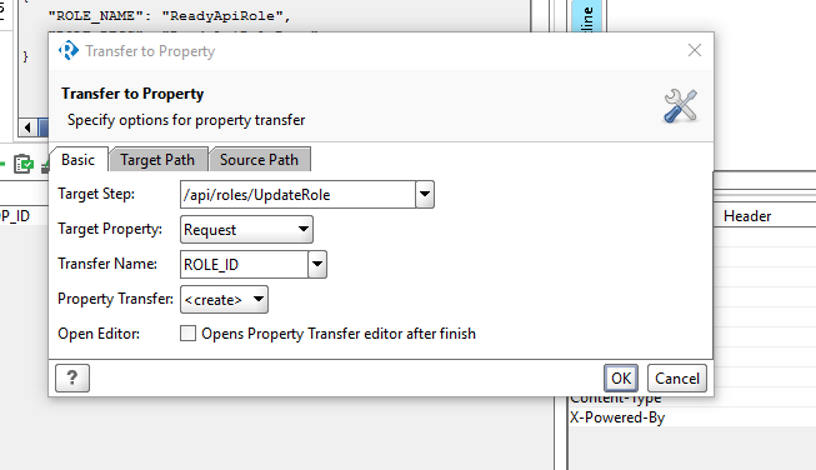
[Working With Properties](https://www.soapui.org/functional-testing/properties/working-with-properties.html)

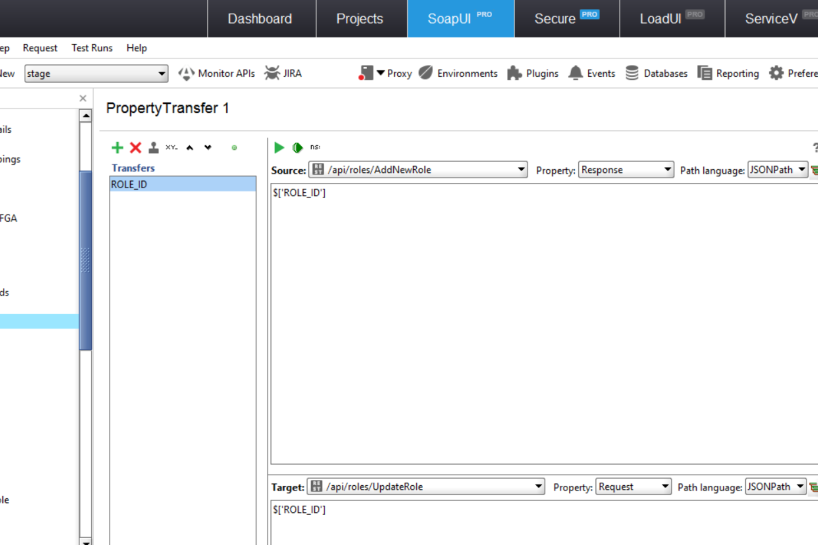
# [Property Transfers: Utilizing Response Data](https://fusion.mastercard.int/confluence/display/NET/Property+Transfers%3A++Utilizing+Response+Data)

Property Transfers is a concept where you can take the response of one request and then use it in a later request.

1. Navigate to your test case --> right-click --> Add Step --> Property Transfer  
   

2.  From here you can continue to walk through the dialog to select your source and destination parameters:





**REFERENCES:**

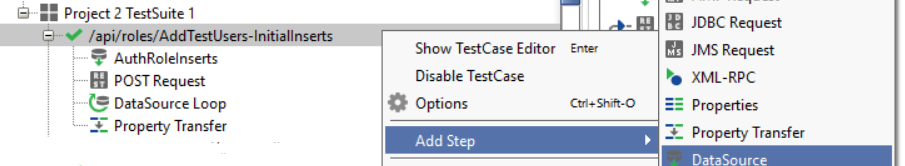
[Property Value Transfers in Functional Testing](https://www.soapui.org/functional-testing/properties/transferring-properties.html)

[Transferring Properties](http://www.softwaretestinghelp.com/soapui-tutorial-8-working-with-soapui-properties/)

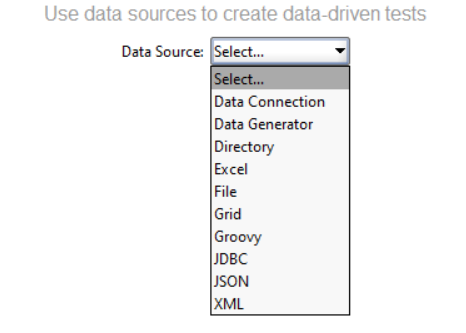
# [Utilizing External Data Sources](https://fusion.mastercard.int/confluence/display/NET/Utilizing+External+Data+Sources)

It can be helpful to load larger amounts of test data for testing.   It is possible to use databases, text files, xml files, Excel files, etc.  Below is a simple example of utilizing an external Excel file to insert data.  For more robust external data usage [see here.](https://support.smartbear.com/testcomplete/docs/testing-with/advanced/working-with-external-data-sources/index.html)

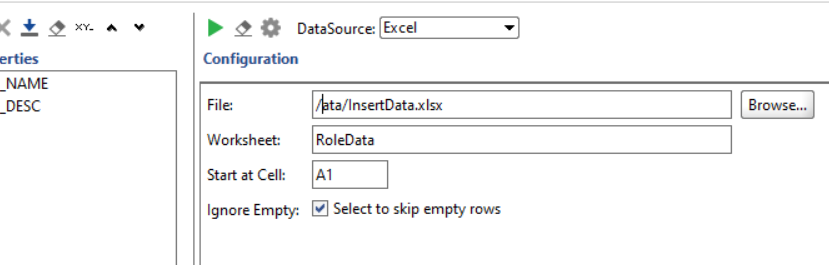
1. Create the test suite and associated cases to be used.
2. Right-click the target test case --> Add Step --> Data Source



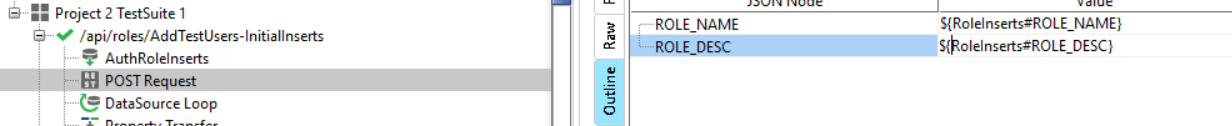
3.  A dialogue window should appear allowing you to select a data source



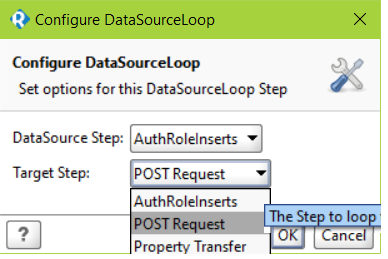
4.  From here, you can choose a data source and navigate down to the details you want:



5.  To assign data rows to data parameters follow the format as referenced below.  For more information on [formatting parameter references see here.](https://fusion.mastercard.int/confluence/display/NET/Referencing+Custom+External+Properties)



6.  Finally, if you want to loop through the data source, add a new test step for "DataSource Loop"

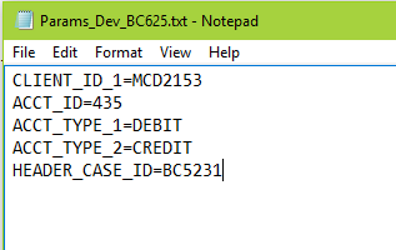


**REFERENCES:**

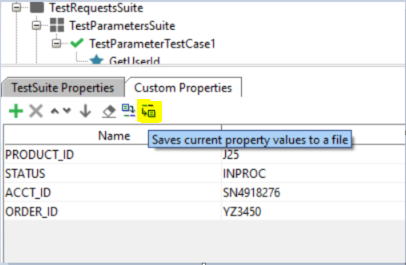
[Working With External Data Sources](https://support.smartbear.com/testcomplete/docs/testing-with/advanced/working-with-external-data-sources/index.html)

# [Creating Re-Usable Parameter Data](https://fusion.mastercard.int/confluence/display/NET/Creating+Re-Usable+Parameter+Data)

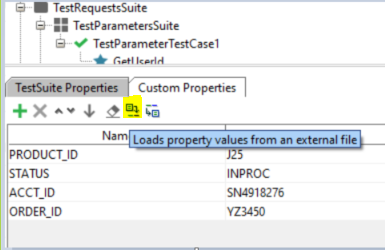
nstead of manually and repeatedly creating header properties and parameters for each test case, you can instead create external files to load data parameters per scenario.  There are multiple ways to do this.  The most simple is using text files.  The text files can be created by hand or via SoapUI; however, either way your parameter file will look something like the below.   To reference your custom parameters see here.



**TO SAVE PARAMETERS ALREADY CREATED:**

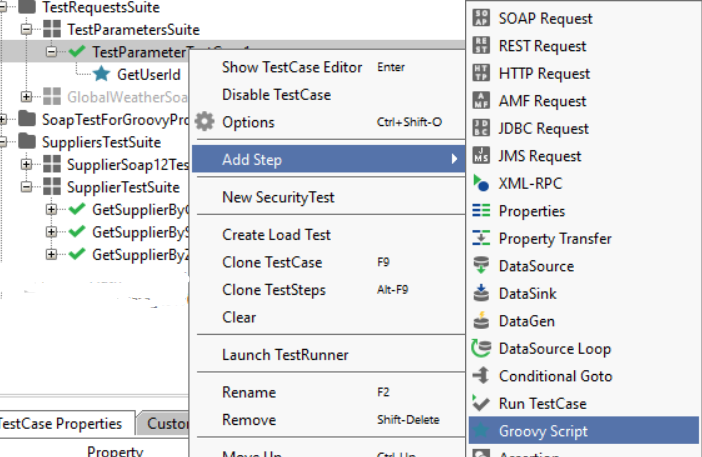


**TO IMPORT SAVED PARAMETERS:**

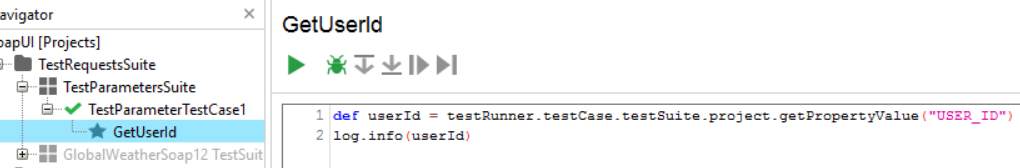


# [Getting Started With Groovy Scripts](https://fusion.mastercard.int/confluence/display/NET/Getting+Started+With+Groovy+Scripts)

Using[Groovy Scripts](http://www.groovy-lang.org/)provides an extremely extensible methodology to extend API development.   An extremely simple example is shown here:

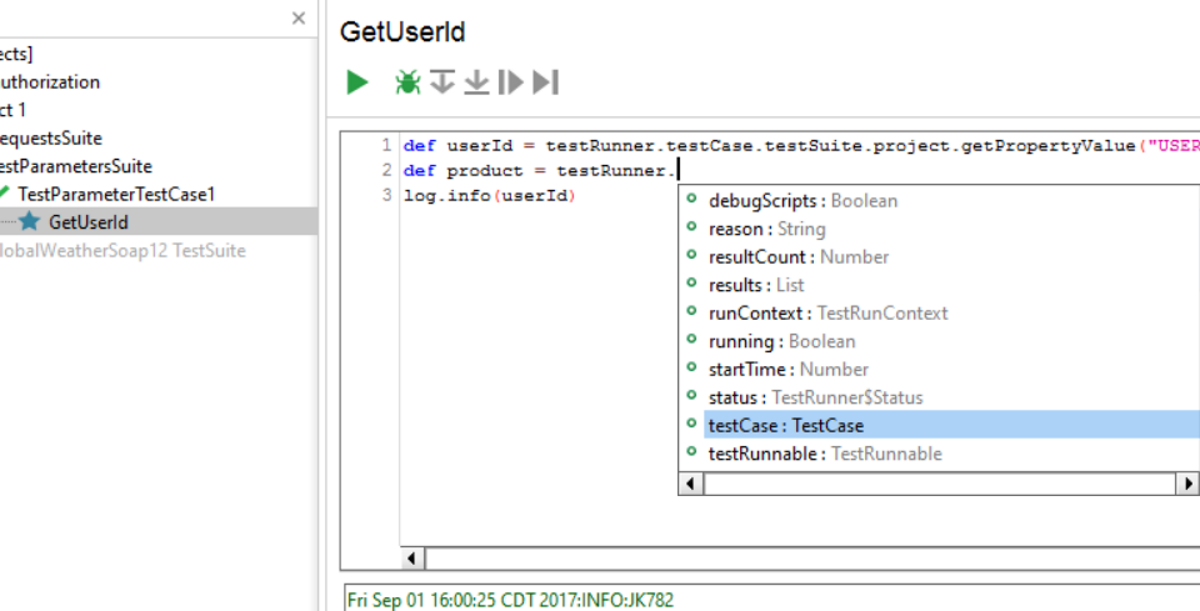
1. Navigate to the desired locale for using a script.  Right click -->  Add Step -->  Groovy Script  
   

2. Double-click on the resulting Groovy Script and an editor will open.  From here you can capture and create re-usable data.



3.  Reading properties from a Groovy Script:

${# Project Name # Value }



**REFERENCES:**

[Scripts and Functional Testing](https://www.soapui.org/functional-testing/working-with-scripts.html)

[SoapUI Groovy Examples](https://www.soapui.org/articles/soapui-groovy-examples.html)

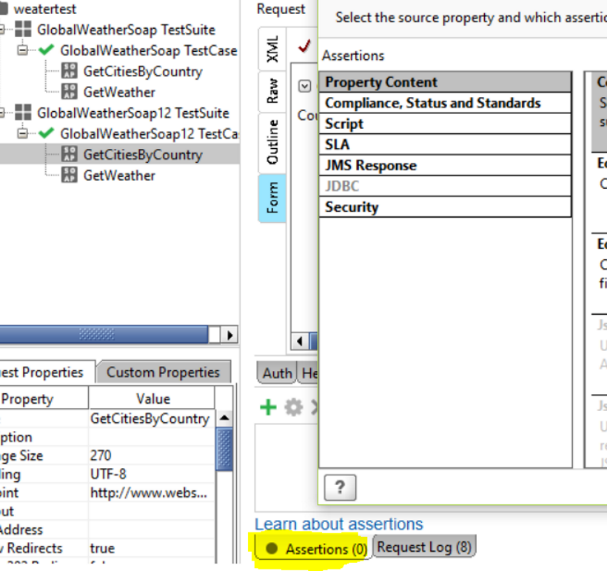
[Scripting and the Script Library](https://www.soapui.org/scripting---properties/scripting-and-the-script-library.html)

# [Creating Assertions](https://fusion.mastercard.int/confluence/display/NET/Creating+Assertions)

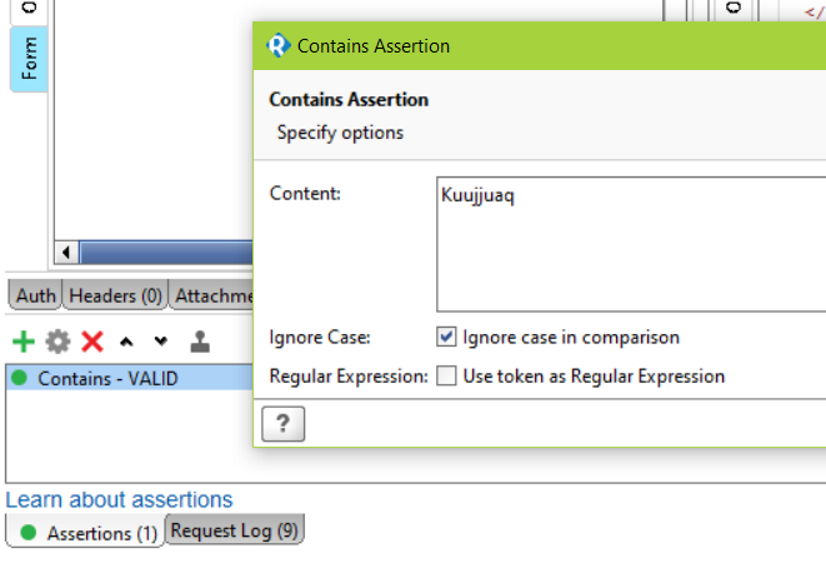
Assertions in SoapUI allow users to compare what the system is supposed to do and what it is actually doing.  There are a variety of assertion options provided by SoapUI; as well as, optional custom assertions.

**TO CREATE AN ASSERTION:**

1. Navigate to test case and click on the "Assertions" tab.



1. Out-of-the box assertions include:  Equals, Contains, XPath match, XQuery match, Schema compliance, Swagger compliance, HTTP Status Codes, script assertions, and more
2. The most basic assertions can easily be handled via the interface:



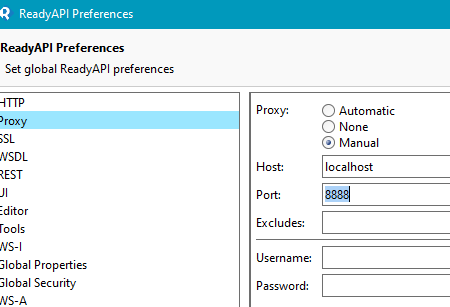
**REFERENCES:**

[Understanding Assertions In SoapUI](http://www.softwaretestinghelp.com/soapui-tutorial-5-soapui-assertions/)

[Functional Testing and Assertions](https://www.soapui.org/functional-testing/validating-messages/getting-started-with-assertions.html)

# [Incorporating Fiddler to Debug API Calls](https://fusion.mastercard.int/confluence/display/NET/Incorporating+Fiddler+to+Debug+API+Calls)

Often it is helpful to trace and debug what SoapUi is actually doing with your calls.  To dig into this information,

1. In ReadyAPI, navigate to 'ReadyAPI Preferences' section.
   1. From here, you can select Proxy -->  Manual --> and enter port "8888"  
      
2. Now, when you open Fiddler, you can narrow down what is captured by selecting "Non Browser" activities and running your test(s).
3. You can also set a filter in Fiddler to capture only SoapUI traffic. Simply drag the “Any Process” icon in Fiddler onto the Soap UI window. Your SoapUI screen will go black to acknowledge the filter setting.

REFERENCES:

* [Fiddler](http://www.telerik.com/fiddler)

# [ReadyAPI Application in Continuous Integration](https://fusion.mastercard.int/confluence/display/NET/ReadyAPI+Application+in+Continuous+Integration)

For now, [see here.](https://support.smartbear.com/readyapi/docs/testing/integrations/ci-systems/jenkins.html)