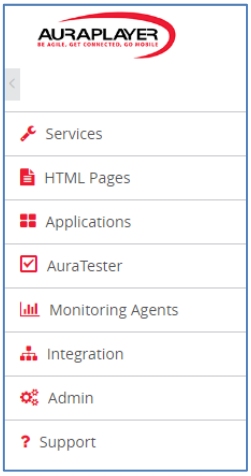
AuraPlayer Server Manager User Guide

Contents

Introduction

AuraPlayer's Service Manager is the tool used to create services and HTML pages as well as store, share and manage them. It also makes it possible to access the files from anywhere.

Main Menu



The Service Manager Menu has the following components:

Services	**Create and manage Services** – View current services, create new services, and perform various operations such as recording, testing and more.
HTML Pages	**Manage HTML pages created by AuraPlayer's Visualizer** – View existing HTML pages, edit and download them.
Applications	Group your Services and Pages to applications **– An application defines a business logic or process, spread across multiple HTML pages.**
AuraTester	**Create, edit and run Test Cases & Test Groups** – Test your services by creating Test Cases. Assemble multiple Test Cases to Test Groups, to help you test complex scenarios.
Monitoring agents	**Create, run and monitor Test Agents** –Create monitoring Agents that run predefined Test Cases or Test Groups in constant intervals. Charts and logs provide you information on the result of each run as well as performance.
Integration	**Export your services to external providers.** Currently, only an export to Oracle Mobile Cloud Service (MCS) and VBCS are supported.
Admin	**Perform administrative tasks and set your preferences**. Backup your data, customize system behavior and manage ORP files.

Services

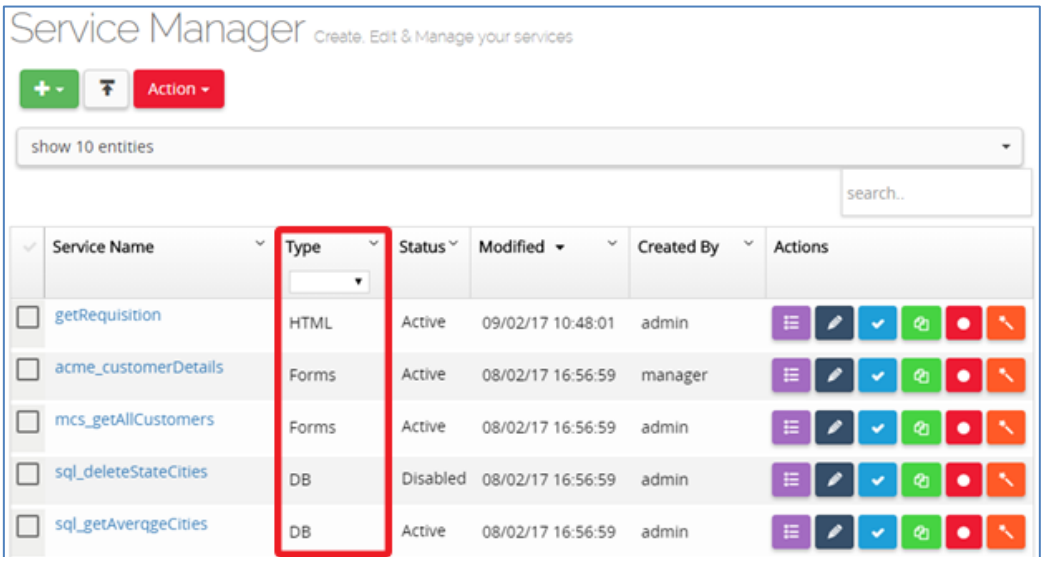
Service Types

AuraPlayer's ServiceManager supports types of services.
All services execute the backend implementation at background, exposing only input and output fields.

- **Oracle Forms service**: executes an Oracle Forms recording.
- **HTML service**: executes a recording of any website, for example: JDE or PeopleSoft.
- **DB service**: executes a database statement or query.
- **JavaScript service**: executes JavaScript code (may call other services and contain custom logic).

Viewing Services

All kinds of services are displayed under the "**Services"** tab from the main menu (left toolbar).
Click on the "Services" tab and the services list will be displayed.

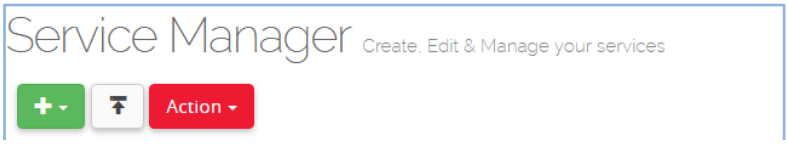


The type of the service is listed under the "**Type**" column.
You may use the dropdown box under the column name to filter and display a particular type only.

Oracle Forms Services

This section introduces services wrapping Oracle Forms/EBS. For other types of services, see the “HTML Services” and “DB Services” sections bellow.

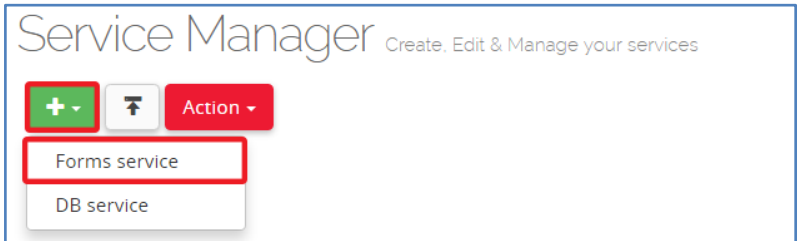
Service Manager Toolbar



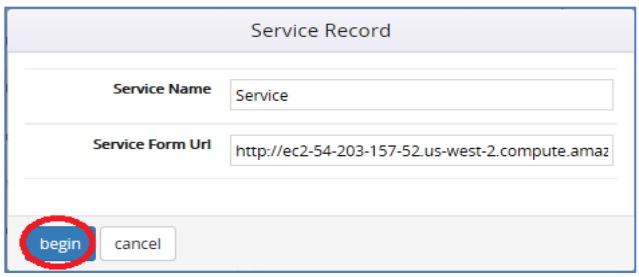
Record/Create	Record/Create a new service
Upload	Upload a service from an existing ORP/HTML file
Action	Perform batch operations on a set of selected services. To select services, check the box next to the service/s
Activate/Disable	Activate or disable the selected services
Export	Export selected services to a csv file
Print	Print a list of the selected services
Delete	Delete the selected services.

Creating a New Service

1. On the Service Manager Toolbar, click on "Create" ⟶ “Forms service”.
   The "Service Record" dialog will be opened.



2. Fill in the service information:



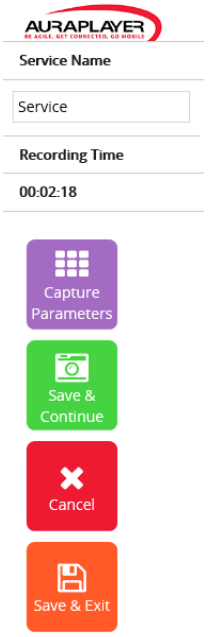
Service Name	A unique name for the new service. Name should reflect the business process.
Form URL	The URL of the Oracle Form you wish to connect and record from. You can only record from a Forms server that has AuraPlayer installed. See the install guide for details.

3. Click “Begin” to start creating your service.

4. The "Recording Toolbar" and the "Oracle Forms" windows will appear.

Recording Toolbar

The Recording Toolbar has the following components:



Service Name	The name you selected for this recording
Recording Time	The duration of the recording
Capture Parameters	Capture all the form's current fields and values as output parameters
Save & Continue	Saves the recorded service, allowing you to continue recording later on. You may use this option to capture a set of values mid-way through recordings.
Cancel	Cancel the current recording
Save & Exit	Stop the recording, and save your new service

5. **Creating Input Parameters**
   To create an input parameter simply input text into the text box, click on a check box, select an option etc. In general, any insertion of values during the Forms session will mark the field and value as an input field.

6. **Creating Output Parameters**
   a. To capture specific output parameters - Click on the desired text fields in the Form, once clicked, they will be captured as output parameters.

b. To capture ALL of the output parameters on a specific form - Click the "Capture Parameters" button on the toolbar. All the fields in the Form will be captured as output parameters.

7. To stop recording without saving the service, click on the “Cancel” button.
8. To save and end the recording, click the “Save & Exit” button. The service is now saved and can be found in the 'Service Manager'. Once pressing the “Save & Exit” button, you will be routed to the 'Edit Service' page to finalize the creation of the Service.

Recording Feature: Capture Parameters

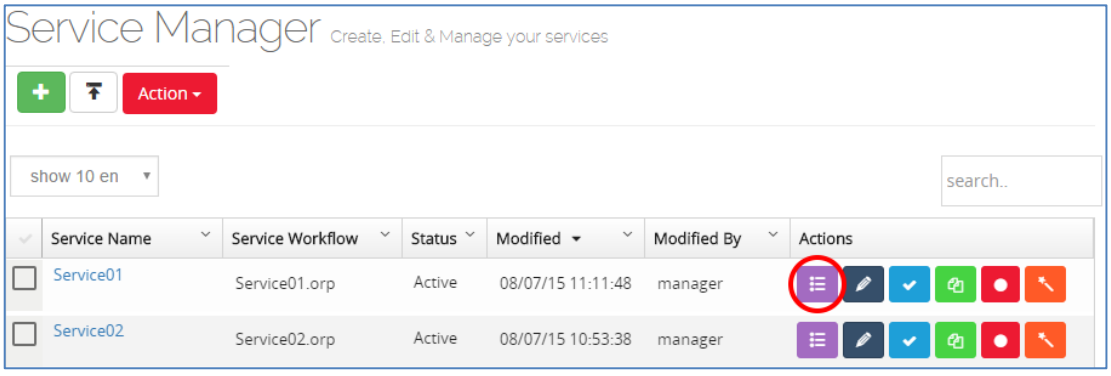
The "Capture Parameters" feature allows you to capture all fields in the Form that is being recorded, marking them as output parameters.

1. Once you reach the Form that has the fields you would like as output fields on your mobilized application, click the “Capture Parameters” button on the Recording Toolbar.
2. Upon success, your recording toolbar will display a success message on the top left corner.

For more details about editing parameters, see the "Edit a Service" Section.

Service Details

To view the service’s details, click on the service name or use the Details button.



The "Service Details" section displays the following information:

Service Name	The name of the service
Description	A brief description of what the service does
Service URL	The RESTful service URL. This URL will be used to create RESTful APi's
WSDL URL	SOAP URL for the service description file (WSDL)
Form URL	The URL of the Oracle Forms server that the service is running against

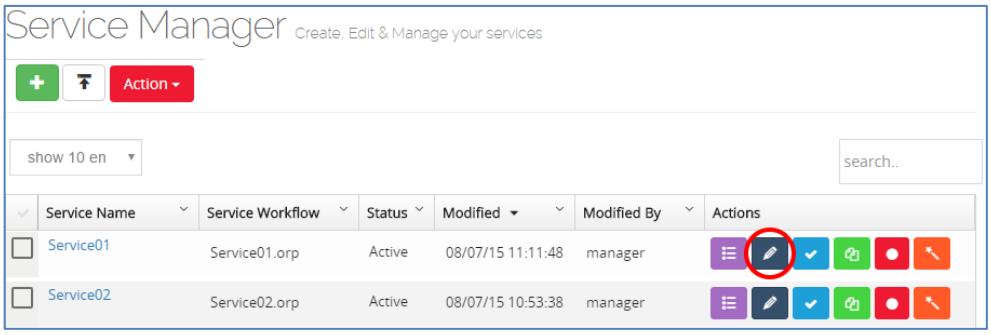
Status	Service status - Active or Disabled

Running AuraPlayer Externally

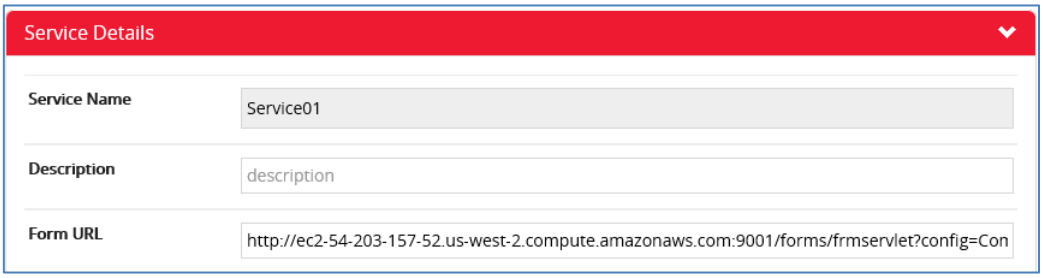
To run the AuraPlayer Service externally, copy the relevant URL (SOAP/REST) from the service details page and create the service client in any tool of your choice. It is also possible to copy the REST URL to your choice of browser.

Editing a Service

To edit an existing service, click on the "Edit" button



Service Details



Service Name	The name of the service
Description	A brief description of what the service does (Optional)
Form URL	The URL of the Form server. You may enter a different URL than the one used at the time of recording.

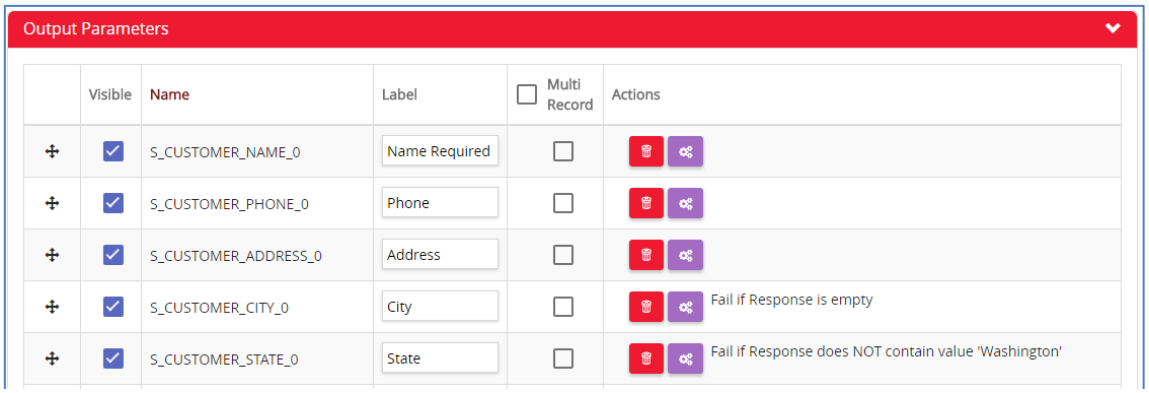
Input Parameters

The 'Input Parameters' are the parameters that the service receives from the end-user.

✣	Drag parameters up or down to change their order
Visible	Choose whether the parameter will be visible in the application or not. For example, you may set a hidden parameter with a fixed predefined value.
Name	Unique name of an input filed, as captured from the Oracle Form
Label	Label to be displayed to the end-user near this field
Default Value	Value to be set as the initial value of this field (Optional).
Move	Copy the current parameter to the Output Parameters list

Output Parameters

The 'Output Parameters' are the parameters the form returns in response to the input parameters.



✣	Drag parameters up or down to change their order
Visible	Whether this parameter will included in the output
Name	Unique name of the input filed, as captured from the Oracle Form
Label	Label to be displayed to the end-user near this field
Multi Record	

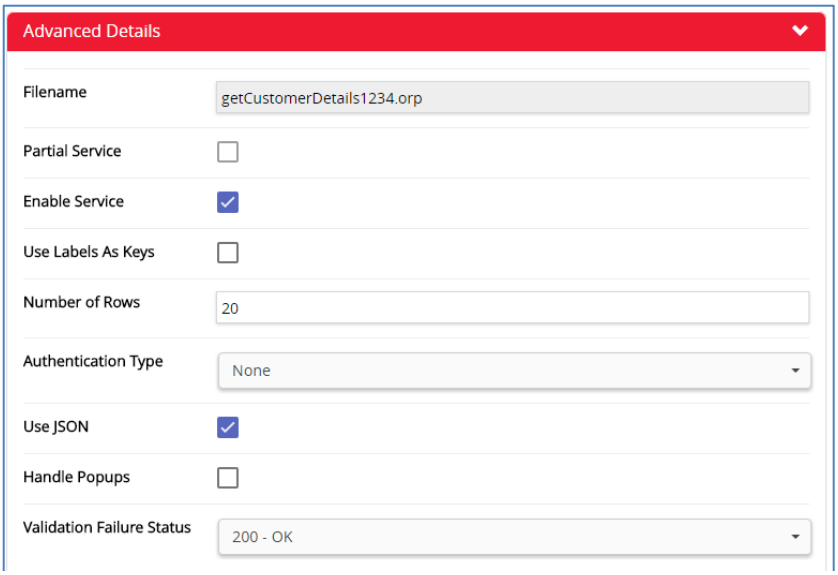
Actions	Delete this output parameter. See “Service validation” below.

Service validation (advanced feature)

You may add validations to the returned values of output parameters.

Click the button next to a parameter, and select one of the validations in the displayed popup. The service will return an error if validation condition is not filled. You may control the returned HTTP code in case of failure in the “Advanced details” section:

Advanced Details



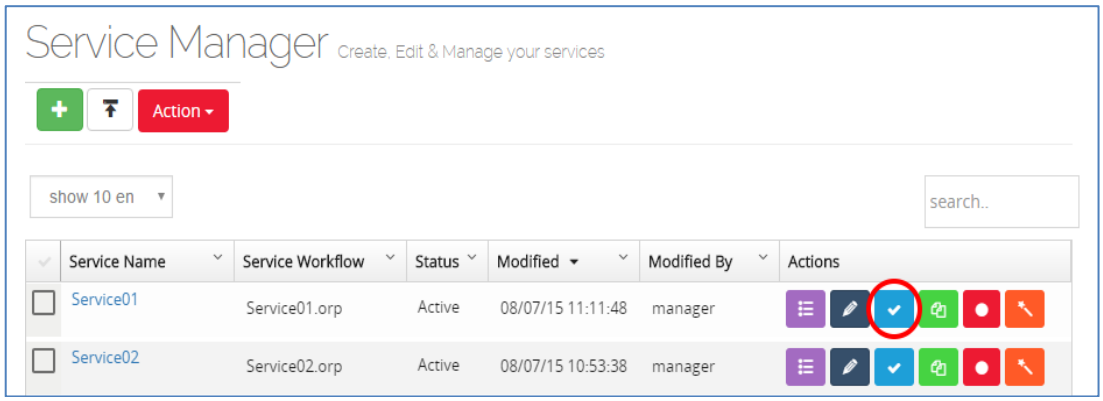
Filename	Name of the ORP file the service is saved from
Partial Service	Partial Service is a recording that does not start from the beginning of the scenario and is not independent. It relies on a previous service to run in order to succeed
Enabled Service	Enable or disable the service
Use Labels As Keys	Call the service with the labels as the parameter names, instead of using the recorded names from the Oracle Form
Number of Rows	Total number of rows you wish to display in the result set
Authentication	**None** – service could be activated simply by accessing its url. **Basic Authentication** – service will require a valid ServiceManager username & password before running (in ‘Authentication’ header).

	Field Encryption - service will require a username & password in 'Authentication' header, passing them to 2 of its input parameters.
Use JSON	Check to receive responses from your REST Service in JSON format
Handle Pop-ups	Deals with pop-ups displayed over the Form during service playback. Check this box if you would like to have the service automatically click on the default button & continue with the playback.
Validation Failure Status	See the "Service validation" section on the previous page.

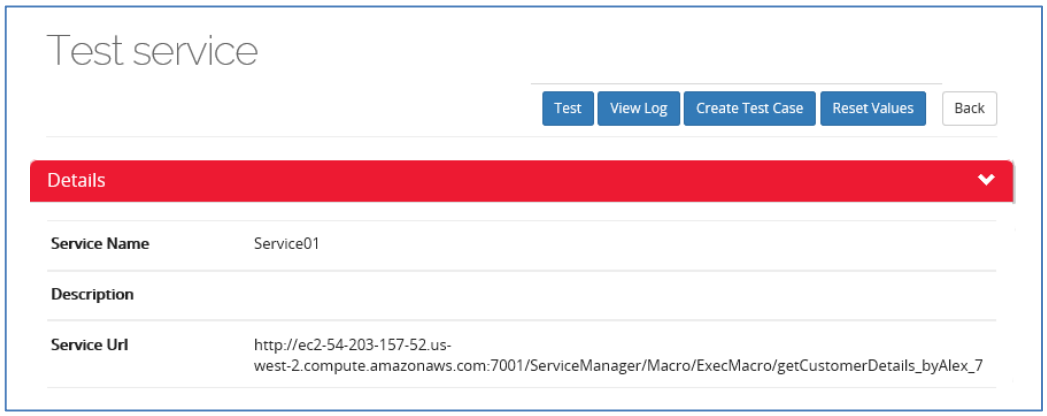
Finally, click 'Update Service' to save your changes to this service.

Testing the Service

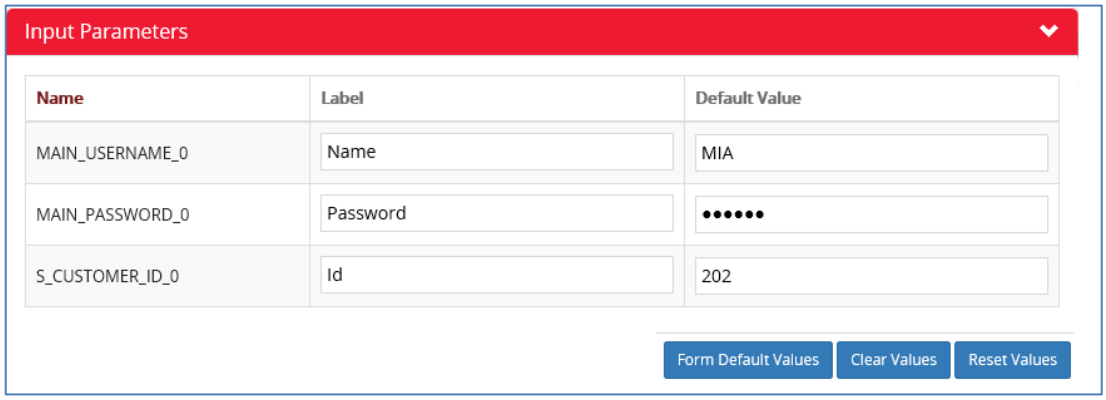
1. To test your service, click the test button in the actions column.



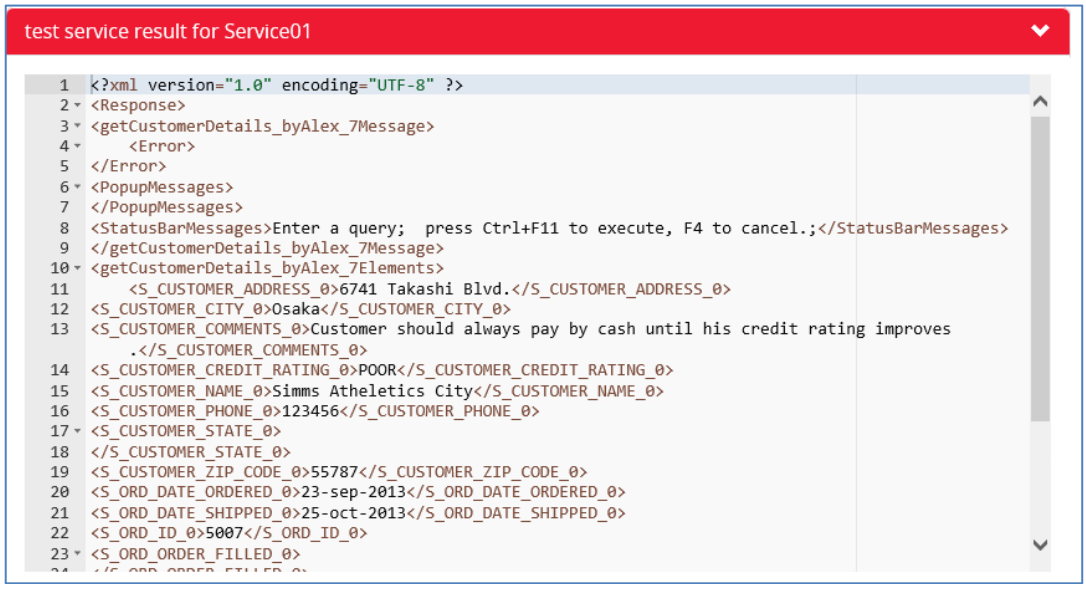
The 'Test Service' page is displayed.



2. To change the input parameters default values, change the values in the "Default Value" column.



3. Once all values are set, click the 'Test' button. The XML response will appear in the text box below



The Forms error-messages, status-bar messages and pop-up messages can be found at the top of the response.

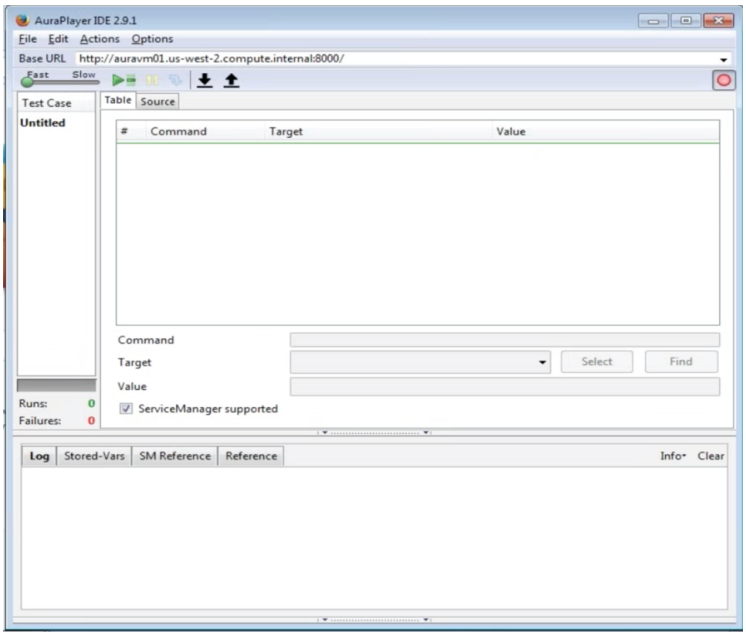
HTML Services

HTML services is our Automation support for web, websites and HTML pages. The 'HTML Service' recorder captures website use cases for automation from our ServiceManager or other integration clients.

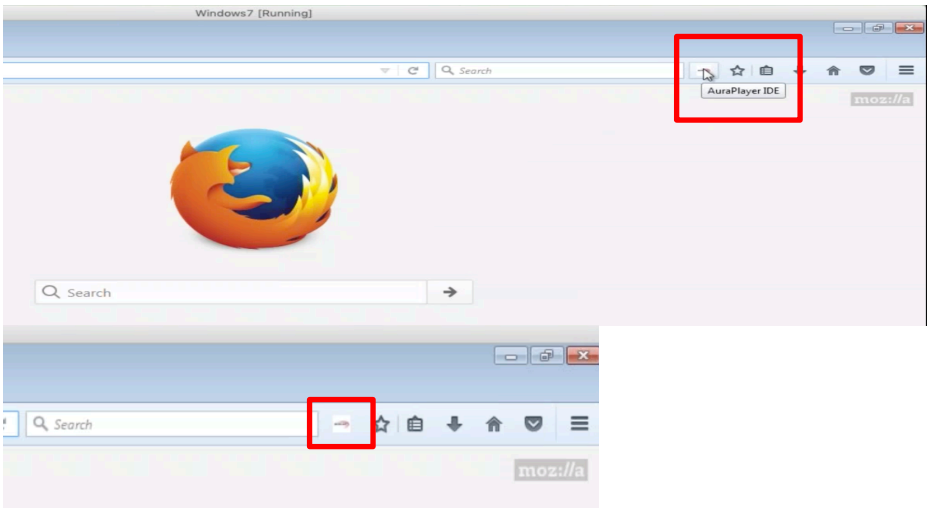
Your ServiceManager operates with HTML Services in a very similar way to Oracle Form Services.

Recording HTML Services

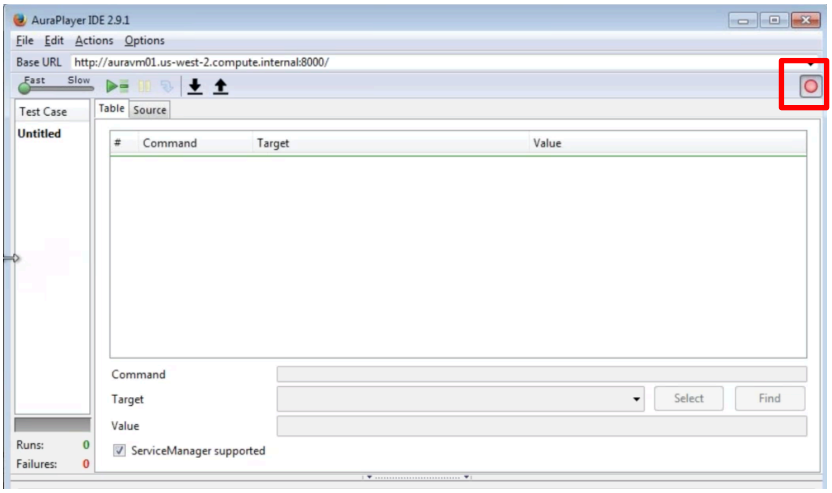
Before beginning, you must get the IDE installation from your AuraPlayer contact.



Once the installation is complete, open the IDE toolbar from the '**Tools'** menu (or press **Ctrl+Alt+S**):



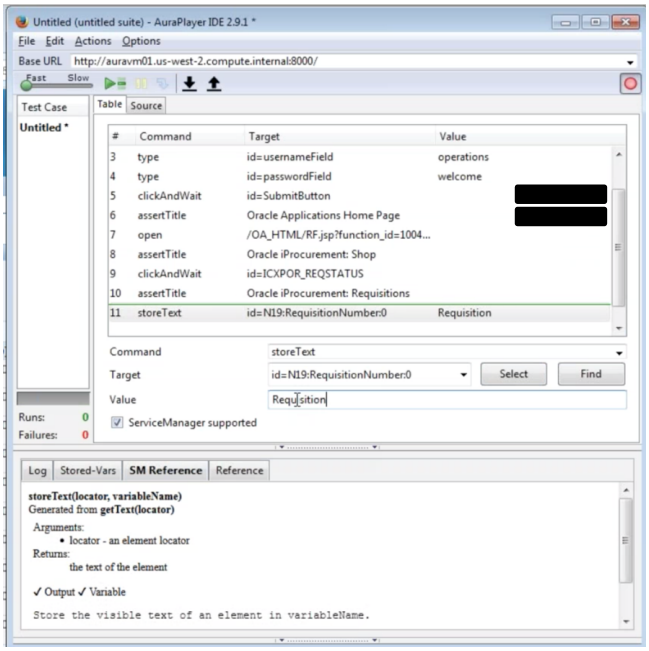
The record button should be pressed automatically so make sure that it is pressed, and you are ready to go with your recording.



In the browser window, perform the actions you would like to record: start by entering a URL to the address bar of the browser, and continue with the rest of your business process (for example – enter username and password, click on a login button, click on some links…).

AuraPlayer Dedicated Recordings

To support our unique functionality, we require some minor adjustments to the recordings generated by the AuraPlayer IDE. They are mandatory to support the service structure, keep robust to changes as well as allow the optimizations we apply for better performance.



This screenshot of an example recording will be referenced from the next paragraphs.

Input parameters

The input parameters of your service are automatically captured from the data you type, links you click, etc. Some types of actions generate input parameters, while other actions do not. Some of the actions that produce input parameters, produce them only when used with particular kinds of **targets**. These targets define how fields will be referred to during playback.

In the example above, the selected action (action #7) – *clickAndWait* – locates the element to click by id (*id=MyReqsTable:RequisitionNumber:2*). By clicking on the down arrow in the 'Target' field, you may see alternative methods to identify the field to be used by the action – all naming elements in the list refer to the same item.
If you require the data to be configurable, always **prefer a simple target**:

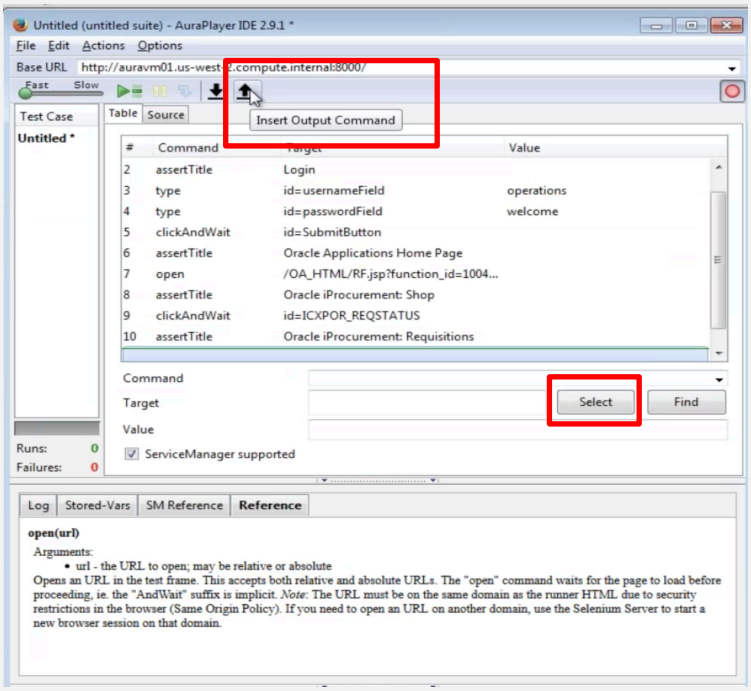
* *"link=14305"*
  Generates an input parameter with default value '14305'.
* *"//span[@id='MyReqsTable']/table[2]/tbody/tr[4]/td/a"*
  Does NOT generate an input parameter since it is too complex and hard to identify the part to be configured. In general, XPath expressions will NOT generate input parameters.

Output parameters

Output parameters cannot be implicitly inferred from the recording, you will have to explicitly define them.

AuraPlayer uses the ***output (storeText)*** action to declare the output parameters of a service.
On the example above, the textual value of the element located by *id=N12:FunctionalAmount:0* is an output parameter of the service.

To add a new action, click on ↑ (up arrow) “Insert Output Command”.

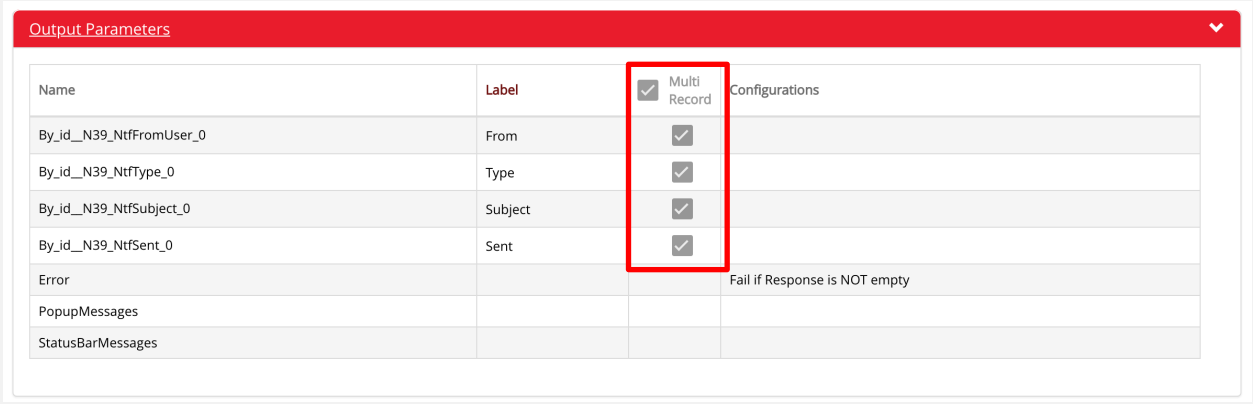


Then, click on ‘*Select*’ button next to the ‘*Target*’ field, and click on the value you wish to capture in the browser window that displays your website.

In the ‘*Value*’ field (variable name) enter a meaningful label for the output parameter to capture the value into.

Capturing **table columns** - capturing the output field is similar to single row recording.

On the Service editing it is required to set in the fields as ‘**Multi Record’**.



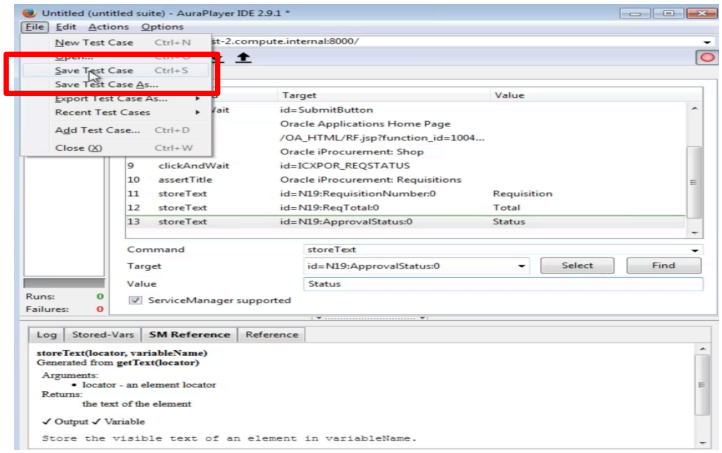
Note that there is also a checkbox '***Table has header***' at the *Advanced* section of the *Service Editor*, which should be checked - if the resulting table returns the column names as the first output line of the table.

A cheat sheet of toolbar commands that your ServiceManager supports is attached at the end of this section.

Finish your recording

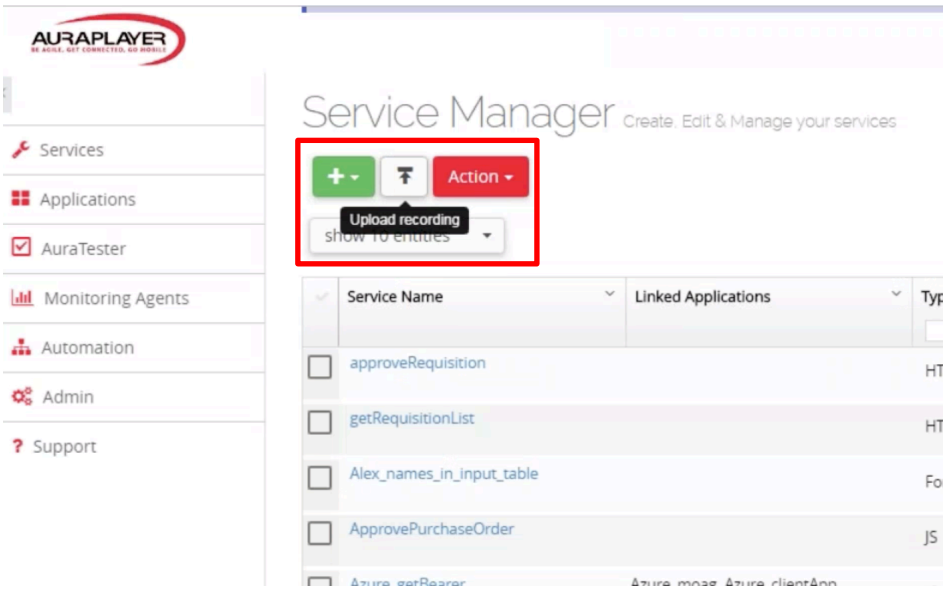
Once you have completed your recording, click on the **Record** button in the toolbar window to stop the recording (see screenshot above).

Open the **File** menu, and click on **Save Test Case…**
Choose a name for your recording (with .html extension), and save it in convenient location. You will need to refer to this from within the ServiceManager to create the playback automation shortly.



Importing a Recoding to the ServiceManager

Open your ServiceManager on the 'Services' tab, and click on the **Upload Recording** button.
Select the recording file you created from the HTML toolbar, and the Service Editor will be displayed.



Viewing recorded HTML services

All services are listed in the 'Services' tab accessible from the left side toolbar.
The 'Type' column in the services list, distinguishes **HTML** Services from other services. And the services can be filtered by clicking on the title and selecting your choice of service type.

Executing, Editing and Using HTML Services

All actions are to be performed in the same manner as with the regular Oracle Form Services (Page 8). Consult this manual for further instructions.

HTML Services output

The HTML Services maintain the same response structure as the Oracle Form Services. They can be tested using the test button from the "Services" detail or edit pages.

Service test result for Recording_OpenRequisition

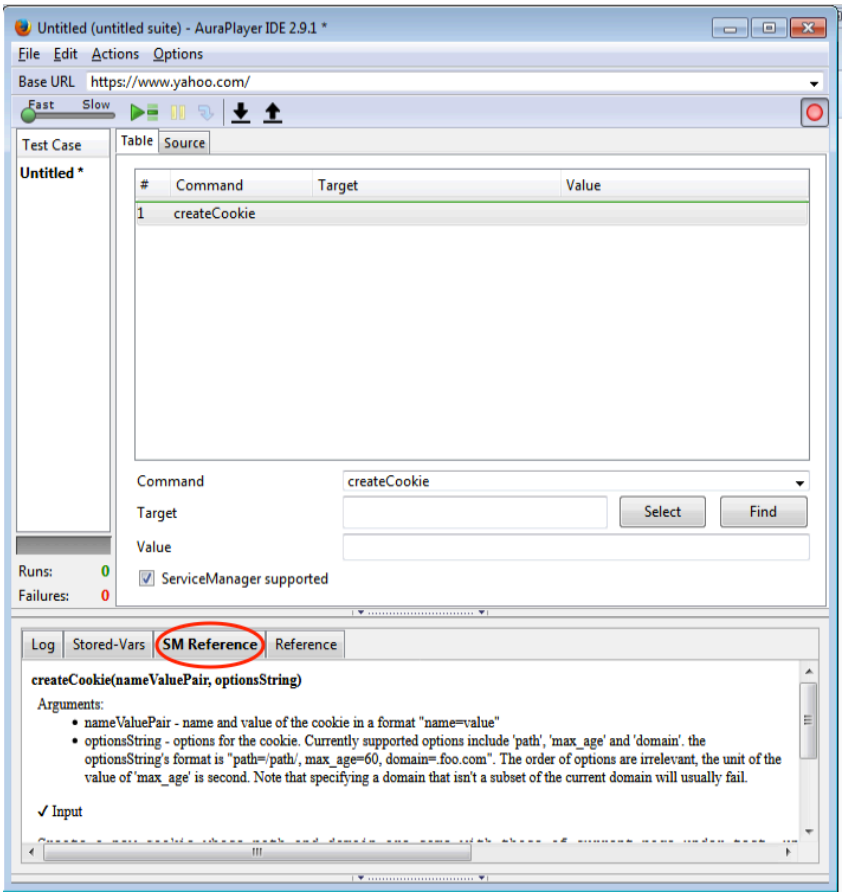

```
{
  "success": true,
  "Response": {
    "Recording_OpenRequisitionSuccess": true,
    "Recording_OpenRequisitionElements": {
      "Amount": 248
    },
    "Recording_OpenRequisitionMessage": {
      "Error": "",
      "PopupMessages": "",
      "StatusBarMessages": "Oracle iProcurement: Requisitions"
    }
  }
}
```


The contents of the 3 special output parameters are:

- Error – HTML playback errors, if any.
- PopupMessages – not relevant for HTML type services (always returns an empty string).
- StatusBarMessages – The page title of the execution of the last command.

Supported Commands

All supported commands, resources and description appear on the HTML Recorder at 'SM Reference' tab at the bottom of the recorder



A few main commands that we support:

Command	Usage
assertLocation	Get the absolute URL of the current page and assert that it matches a regex. If does not match - abort execution.
assertTitle | assertNotTitle	Get the title of the current page and assert that it matches/not-matches a regex. If match result is different from expected - throw an exception, aborting execution.
click | clickAndWait (locator, optionalAction*)	Clicks on a link, button, checkbox or radio button. If the click action causes a new page to load – wait for it to finish. optionalAction – an optional JavaScript code to be injected as the 'onclick' action of the button.
pause (millisToWait)	Wait for the specified amount of time (in milliseconds).
runScript | runScriptAndWait (script, placeholderValue*)	Creates a new "script" tag in the body of the current test window, and adds the specified text into the body of the command. script – if starts by // (meaning comment), the script won't be executed by the IDE, but will be executed by the AuraPlayer ServiceManager. placeholderValue – all ${something} tokens in 'script' will be replaced with 'placeholderValue' before running the script.
select | selectAndWait (selectLocator, optionLocator)	Selects an option on <select> element. Currently we support selection of values identified by label= type only.
selectWindow (name)	Selects a popup/window using a window locator; once a window has been selected, all commands go to that window. To select the main window again, null as the target.
storeChecked (locator, variableName)	Get whether a toggle-button (checkbox/radio) is checked, and store it in an output parameter called 'variableName'.
storeHtmlSource (variableName)	Returns the entire HTML source between the opening and closing "html" tags. It is saved in the 'temp' folder, in a file called 'variableName'.
storeText (locator, variableName)	Get the text of an element, and store it in an output parameter called 'variableName'.
submitAndWait (locator, optionalAction*)	Submit the specified form. optionalAction - an optional JavaScript code to be injected as the 'action' attribute of the form.
type (locator, value)	Sets the value of an input field, as though you typed it in.

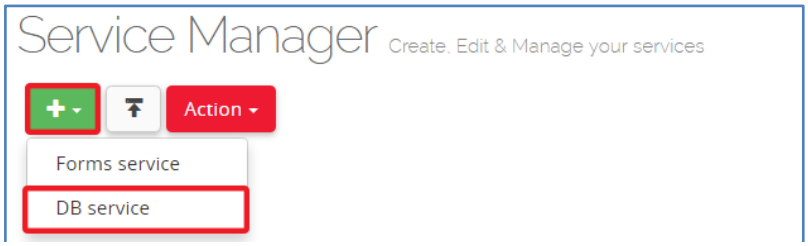
uncheck | uncheckAndWait (locator)	Same as *check* | *checkAndWait* – but the generated input parameter is initialized to *false* value.
waitForElementPresent (locator, shouldWait*)	Waits until the specified element appears, or a timeout is reached. shouldWait – playback will ignore this command if set to 'false' (in order to save performance).
waitForPageToLoad (timeout)	Waits for a new page to load (with max wait time).

DB Services

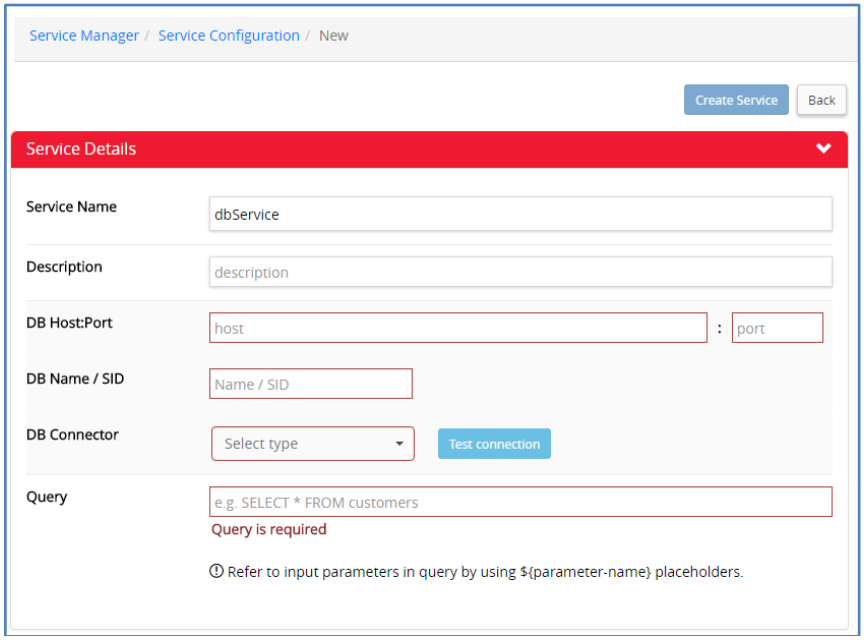
The 'DB Service' executes database queries/operations, and returns the result set.
Your ServiceManager operates with DB Services in a very similar way to Oracle Form Services.

Creating DB Services

1. On the Service Manager page, click on "Add" ⟶ "DB service".



2. The "Service Editor" will be opened, with dedicated database configuration view.

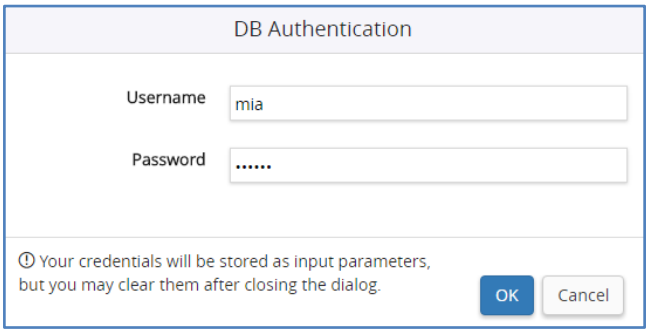


3. Enter a service name and description (optional).

4. Configure the database connection:
   - Fill in the **hostname** of the database server.
   - Fill in the database **port**.
     Default ports are: 1521 for Oracle databases, 3306 for MySQL, and 5432 for PostgreSQL.
   - Enter the database **Name or SID**.
   - Select the database **type**.

5. Click on "**Test connection**" to ensure that your connection is properly configured.

6. A username/password dialog will open, enter your credentials to the database server and confirm.



Your username and password will be automatically stored as the values of the corresponding input parameters. You may clear or hide them if you do not wish them to be exposed in your service.

7. Success/failure status will be displayed in popup message.
   If the connection failed, do not continue any further - check your DB configuration and retry.

8. Enter a “Query” as described in the following section.

DB Queries / Statements

The last field under “Service details” of a Database Service is the “**Query**” field.
Enter a valid SQL statement.
You should enter a single statement (without semicolon). In order to execute multiple commands, wrap them in function or procedure and call it from here.

Examples:

- SELECT STATE FROM CITIES WHERE POPULATION < 1000

- INSERT INTO CITIES (STATE, CITY, POPULATION, WHITE, BLACK, HISPANIC, ASIAN, OTHER) VALUES ('DE', ' Muenchen', 1300000, 0, 0, 0, 0, 0)

Dynamic queries (injecting input parameters)

Your query may vary according to input provided to the service.

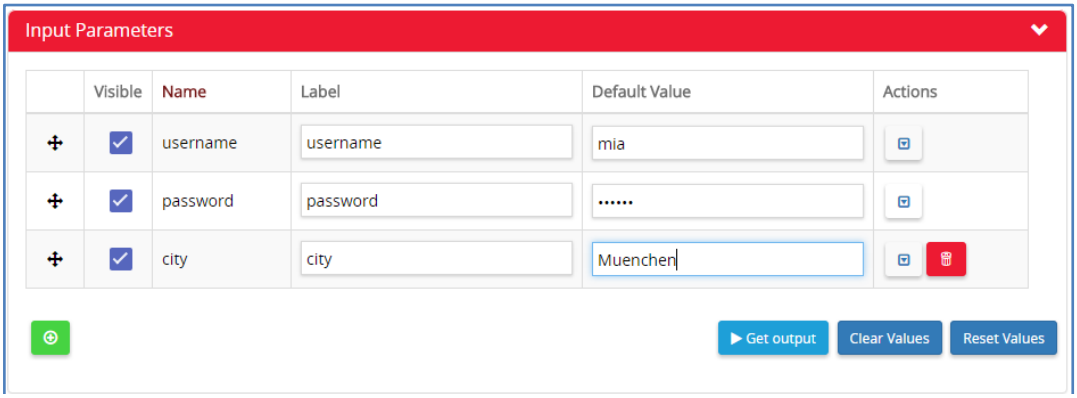
For example, assuming that you are managing a CITIES table, and your service updates the population of the city, then the city name and population are probably input provided by the user executing the service. Meaning, instead of writing:

- UPDATE CITIES SET POPULATION=40000 WHERE CITY='Muenchen'

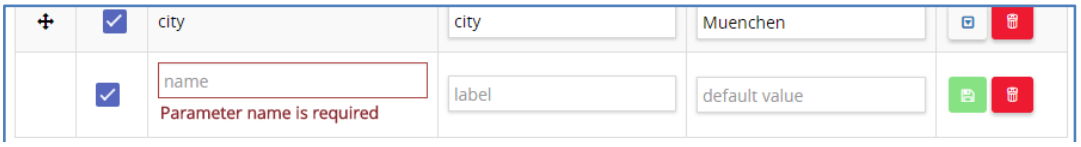
You may write:

- UPDATE CITIES SET POPULATION=**${population}** WHERE CITY='**${city}**'

Now, the **population** and **city** values are expected to be found in the input parameters of the service.
You should add an input parameter named **population** and and input parameter named **city** to the input parameters table:



Click on the add button , to append a new row at the bottom of the input parameters table.



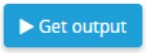
Fill in the name (must match to the query; use **population** in our example), label and default value for parameter. Finally click on the save icon.
You may now add more input parameters – add **city** for this example.

Query examples

SELECT	SELECT * FROM ${table} WHERE POPULATION < ${max_population}
UPDATE	UPDATE CITIES SET POPULATION =0 WHERE CITY='${city}'
INSERT	INSERT INTO CITIES (STATE, CITY, POPULATION, WHITE, BLACK, HISPANIC, ASIAN, OTHER) VALUES ('${state}', '${city}', ${population}, 0, 0, 0, 0, 0)
DELETE	DELETE FROM CITIES WHERE state = '${state}'
Calling a function	SELECT GET_POPULATION('${city}') **as result** FROM dual Since the name of the selected value is dynamic, we have to add "as result" (or any other name) so the output will be returned by that name and we would be able to capture the value with an output parameter named "result".
Executing a procedure	EXECUTE INCREASE_POPULATION('${city}')

Testing Queries and Acquiring Output Parameters

After configuring your database connection and query, click on the "**Get output**" button to execute your query for the first time.
In addition to executing the query, it also adds any table columns returned from the query to the "**Output parameters**" table.



Your query may not return any values, and no output parameters will be added (except from the traditional *Error, PopupMessages,* and *StatusBarMessages*).
The number of new output parameters added to the table will be displayed in a popup message:

You may also manually add, hide, or delete any output parameters.

Finish by saving the service.

Viewing recorded DB services

All services are listed in the 'Services' tab from the left toolbar.
The 'Type' column in the services list, distinguishes **DB** Services from other services.

Executing, Editing and Using DB Services

All actions are to be performed in the same manner as with the regular Oracle Form Services.
Consult this manual for further instructions.

DB Services output

The DB Services maintain the same response structure as the Oracle Form Services:

Service test result for sql_insertCity


```
{
  "success": true,
  "Response": {
    "sql_insertCitySuccess": true,
    "sql_insertCityElements": {},
    "sql_insertCityMessage": {
      "Error": "",
      "StatusBarMessages": "1 rows affected."
    }
  }
}
```


Service test result for sql_getSmallCities


```
{
  "success": true,
  "Response": {
    "sql_getSmallCitiesSuccess": true,
    "sql_getSmallCitiesTableArray": {
      "sql_getSmallCitiesArrayItem": [
        {
          "City": "Vatican City",
          "Population": 800
        },
        {
          "City": "Maza",
          "Population": 5
        }
      ]
    },
    "sql_getSmallCitiesElements": {},
    "sql_getSmallCitiesMessage": {
      "Error": ""
    }
  }
}
```

The contents of the 3 special output parameters are:

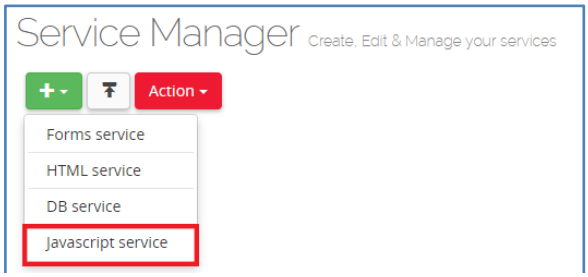
- Error – DB errors, if any.
- StatusBarMessages – Number of affected rows if the executed statement has no output.

JavaScript Services

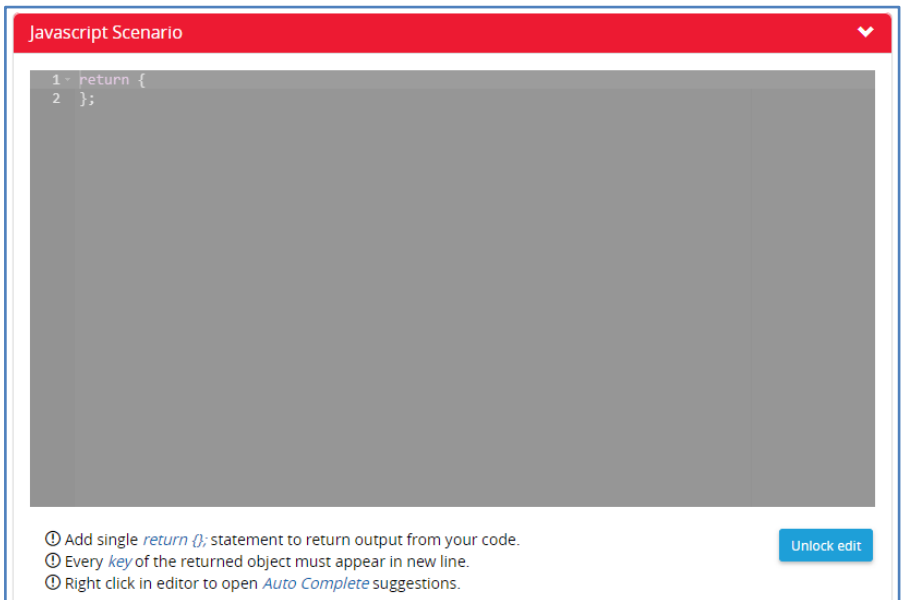
The 'JavaScript Service' executes JavaScript code.
Your ServiceManager operates with JavaScript Services in a very similar way to other types of services.

Creating JavaScript Services

1. On the Service Manager page, click on "Add" ⟶ "Javascript service".



2. The "Service Editor" will be opened.
   In the "JavaScript Scenario" section, you enter your JavaScript code.

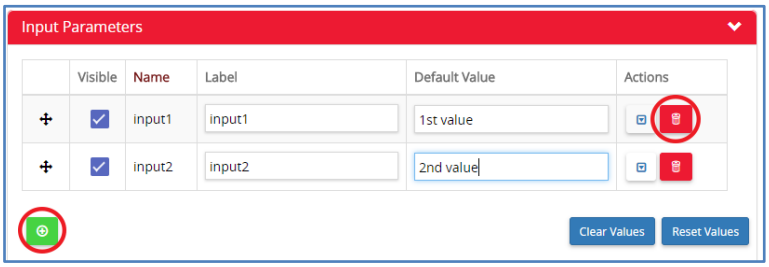


   First, click on the 'Unlock edit' button to open the editor for editing.
   When editing, you cannot add/remove input parameters nor view output parameters.

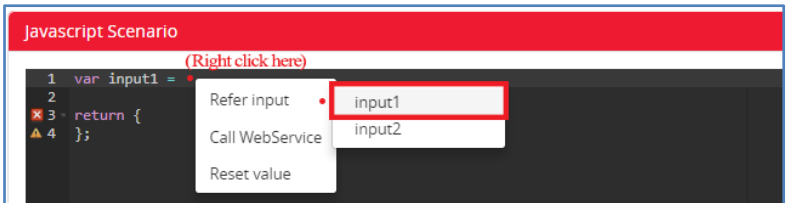
Once you done writing your code, click on the 'Lock edit' button to configure service parameters. The editor will be disabled and grayed out again.

Adding Input Parameters

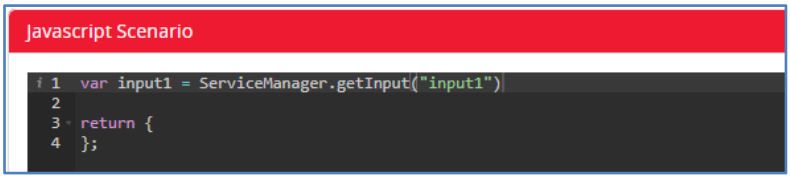
To edit input parameters, make sure that the JavaScript editor is locked.
Use 'Add'/'Delete' buttons in the "Input Parameters" section to create/delete input parameters.



To refer input parameters in the JavaScript code, right click in the editor, and select the input parameter from the context menu.

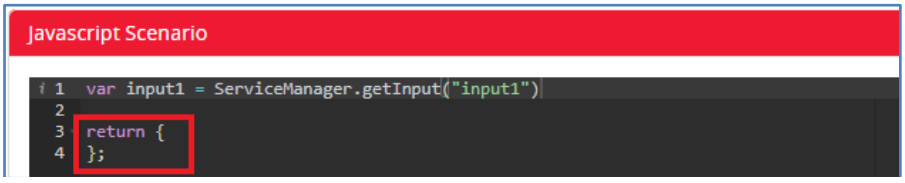


An appropriate expression will be inserted at the current cursor position.
In runtime, this expression will be evaluated as the string value of the parameter.

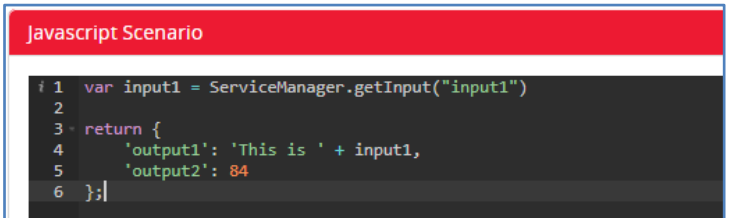


Setting Output Parameters

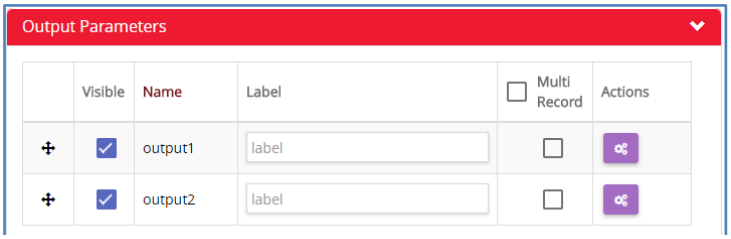
The JavaScript code returns output using a single 'return' statement at the topmost level (i.e. from a single non-nested 'return'). The returned value must always be a valid JavaScript object.



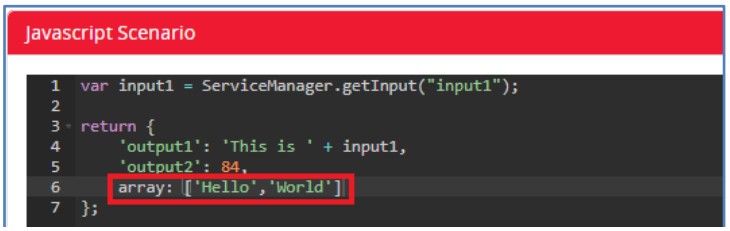
Each property in the returned object evaluates to output parameter: the name of the parameter will be the key of the property, and its value will be assigned accordingly after execution.
Every key in the returned object must appear in new line.
The following code declares two output parameters: output1, output2.



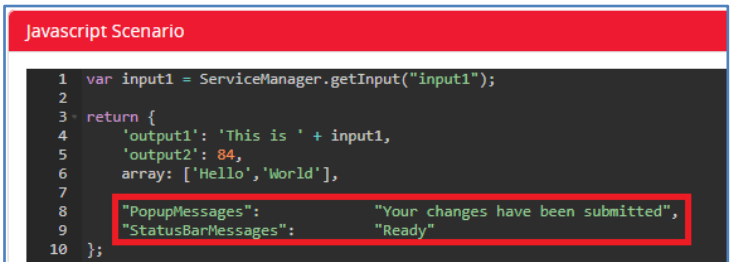
Output parameters are automatically derived from the 'return' statement once you lock the editor.



If you wish to return **array** value, you should mark the generated output parameter as **multi-record**. Returning array value into non-multi-record output parameter will result in assignment of the first array element to the parameter.

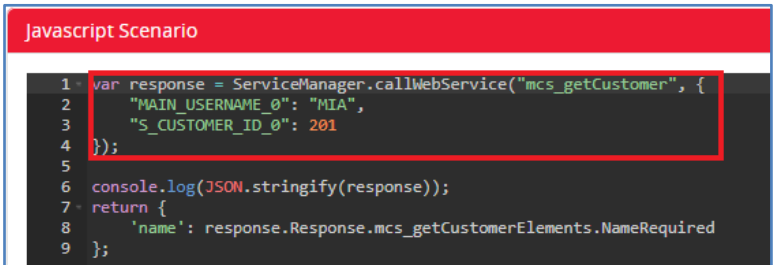


In the same matter you may also return values to the special "StatusBarMessages" and "PopupMessages" output parameters. However, returning value to the "Error" field is done by throwing an error – see "Throwing errors" section bellow.



Calling other services

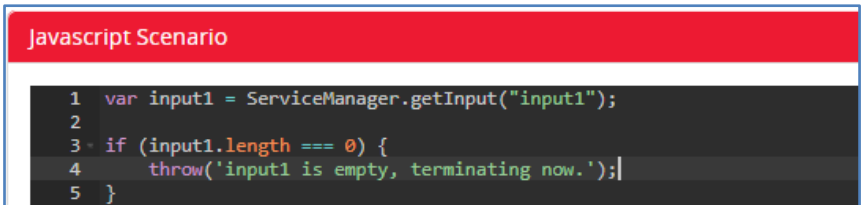
The JavaScript service may invoke other web services on your ServiceManager, and read their result.
Right Click (in the editor) ⟶ **Call WebService**, to insert a template for web service call:



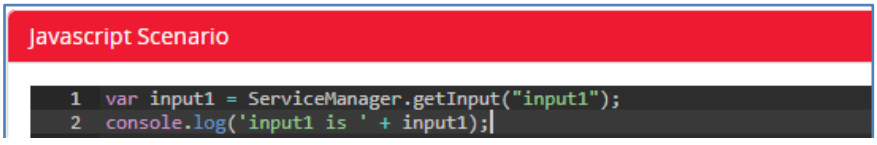
An expression that calls the 'callWebService' method is inserted. Replace the 1st argument with the name of the service to call, and the 2nd argument with an object whose properties are the input names and assigned values.
The returned value is a JSON object, in the exact structure as a service's JSON response.
You may address each field by using the full JSON path, such as:
response.Response.mcs_getCustomerElements.NameRequired.

Throwing Errors

Your JavaScript service may exit abnormally by throwing an error.
Use a 'throw' command to terminate execution and report error in the 'Error' output parameter.

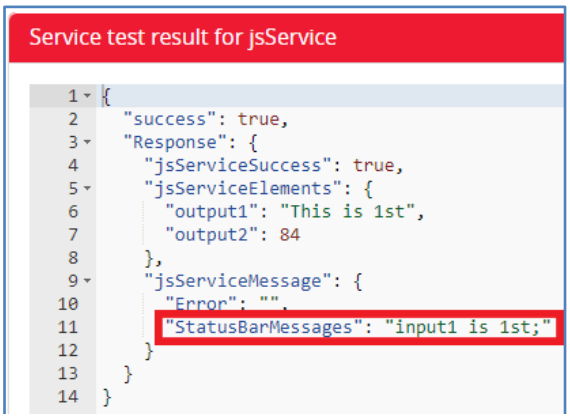


Logging and Debugging



Logging to console using 'console.log()' (or similar), will append the message to the system log (Admin ⟶ View Log).

Additionally, it will append the message to the 'StatusBarMessages' output parameter, if its value is not set in the 'return' statement (see "Setting output parameters" section above). To suppress propagation of console message to 'StatusBarMessages', assign value to it in the 'return' clause – an empty string may be used if necessary.



Supported console commands are: console.log, console.error, console.warn, console.debug, console.info, console.trace.

The system log will preserve log level (console.log will be logged as console.debug). 'StatusBarMessages' (if not overridden), will store all log messages regardless of their severity.

Support

Getting Support

Please feel free to visit: support.auraplayer.com to open a service request.

Or email support@auraplayer.com

Thank you for choosing AuraPlayer.