

How to Create a Dynamic Action Plug-in

We are going to develop a dynamic action plug-in which can read some data from the backend and assign it to a page item or DOM object. This plug-in uses an AJAX call to submit the current values in the browser to the server and read data from there. The backend code itself uses dynamic SQL to execute any SQL statement the developer provides. The advantage compared to a “Select List with Redirect” is that a PPR (partial page request) is done, compared to a full load of the page. The handling should also be much simpler for a developer.

1. Go to **Shared Components\User Interface\Plug-ins**
2. Click **Create >**
3. Enter **Get Data** into the field **Name**
4. Enter **com.yourcompany.apex.get_data** into the field **Internal Name**. Replace yourcompany with the name of your company.

Note: The value has to be unique within your application. If you want to contribute your plug-in to the APEX community, it should be unique world wide. That’s why you should use a reverse version of your companies domain name as prefix.

5. Set **Type** to **Dynamic Action**
6. Leave everywhere else the default values and click **Create**
7. Open the file **get_data.js**
8. Replace **yourcompany** with your company name
9. Save the file.

Note: See the code comments for additional information on how such a plug-in javascript function has to look like.

It’s highly recommended that you use a javascript/CSS compressor (eg.: <http://developer.yahoo.com/yui/compressor/>) to reduce the size of your javascript and CSS code when you are done with your development!

10. Go to section **Files**
11. Upload the **get_data.js** file

Note: A plug-in has its own file storage which can store images, css- and javascript files. This concept has the advantage that it allows a single click installation of the plug-in, without having to upload something to a web server, ... A user can still do that to optimize performance, but by default he is done after installing the plug-in.

12. Go to section **Attributes**
13. Click **Add Attribute**
14. Enter the following values
Scope: **Component**
Attribute: **1**
Label: **SQL Statement**

Type: **SQL Query**
Required: **Yes**
Minimum Columns: **1**
Maximum Columns: **1**
Default Value: **select sysdate from dual**

Note: A plug-in can have up to 10 attributes for each scope. These attributes are displayed in the APEX Builder, when a developer selects the plug-in in the “Create Dynamic Action wizard” or on “Edit Dynamic Action”. Attributes extend the existing properties of a dynamic action and allow a plug-in to prompt the developer in a declarative way for additional data the plug-in requires.

There are two scopes available. “Component” means that an attribute can be entered each time when the plug-in is used for a page item. “Application” means that you can just enter the values once for the whole application.

15. Click **Create and Create Another**

16. Enter the following values

Scope: **Component**

Attribute: **2**

Label: **Page Items to Submit**

Type: **Page Items**

Required: **No**

17. Click **Create**

18. Go to section **Source**

19. Paste the code which you can find in the file **get_data_plugin.sql** into the **PL/SQL Code** field. Replace **yourcompany** with your company name.

Note: See the code comments for additional information on how a render and AJAX procedure has to look like.

20. Go to section **Callbacks**

21. Enter **render_get_data** into the field **Render Procedure Name** and **ajax_get_data** into the field **AJAX Procedure Name**.

Note: Because of performance reasons you can also store the plug-in code in a PL/SQL package. In that case you would just use `package_name.render_get_data` instead and leave the PL/SQL code field blank. Especially during development of a plug-in this is a more convenient way than writing the PL/SQL code of the plug-in in the text area of the browser.

22. Click **Apply Changes**

23. Let's test our new plug-in!

24. In the **Sample Application** go to **page 11**

25. Create a **new page item**

Item Type: **Display Only**

Pick Display Only Type: **Display as Text (does not save state)**

Item Name: **P11_ADDRESS**

Sequence: **20**

Region: **Select a Customer**

Label: **Address**

Item Source: **Always Null**

26. Set the existing page item **P11_CUSTOMER_ID** to Display As **Select List**

27. Create a **Dynamic Action** with the following values:

Name: **Show Address**

Sequence: **10**

Element Type: **Item**

Element: **P11_CUSTOMER_ID**

Condition Type: **Always**

28. Click **Create**

29. Go to section **Actions**

30. Click **Add Action**

31. Use the following values

Fire When Event Result is: **True**

Sequence: **10**

Action: **Get Data**

Fire On Page Load: **Yes**

Element Type: **Item**

Elements: **P11_ADDRESS**

32. Click **Apply Changes**

33. Click the **Edit Icon** of the just created Action

34. Set the following values in the **detail view** of the action **Get Data**

SQL Statement:

```
select sys.htf.escape_sc(cust_street_address1) || '<br />' ||
       sys.htf.escape_sc (
         nvl2(cust_street_address2, cust_street_address2, null)||
         cust_city || ', ' ||
         cust_state || ' ' ||
         cust_postal_code )
from demo_customers
where customer_id = :P11_CUSTOMER_ID
```

Page Items to Submit: **P11_CUSTOMER_ID**

Note: It's in the responsibility of the developer that he uses the necessary escape functions. The plug-in could also do that, but it would prevent the usage of any formatting HTML tags like bold or line breaks

35. Click **Apply Changes**

36. **Run** the page

37. If you are done with testing, go again to **Shared Components\User Interface\Plug-ins\Get Data**

38. Click **Export Plug-in** in the Task sidebar

39. Follow the wizard steps to export the plug-in

40. You are done! You can now share and use the plug-in in any application by installing it with the **Install button** in Shared Components\User Interface\Plug-ins