

How to Create an Item Type Plug-in

We are going to develop an item type plug-in which allows a user to more easily enter fix width input, where we as a developer would like them to enter the data in a certain format (phone numbers, dates, etc). We are not re-developing this kind of functionality, we are just using the existing jQuery plug-in “Masked Input Plugin” (<http://digitalbush.com/projects/masked-input-plugin/>) and are going to write a PL/SQL wrapper for it.

1. Go to <http://digitalbush.com/projects/masked-input-plugin/> and download the **Minified** version of the javascript code of the jQuery plug-in
2. Open or create the **Sample Application** which comes with APEX
3. Go to **Shared Components\User Interface\Plug-ins**
4. Click **Create >**
5. Enter **Masked Text Field** into the field **Name**
6. Enter **com.yourcompany.apex.masked_text_field** 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.

7. Set **Type** to **Item**
8. Leave everywhere else the default values and click **Create**
9. Go to section **Files**
10. Upload our downloaded **jquery.maskedinput-1.2.2.min.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.

11. Go to section **Attributes**
12. Click **Add Attribute**
 1. Enter the following values

Scope: **Component**
Attribute: **1**
Label: **Mask**
Type: **Text**
Required: **Yes**
Translatable: **No**
Display Length: **40**
Max Length: **40**
Default Value: **(999) 999-9999**

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 Page Item wizard” or on “Edit Page Item”. Attributes extend the existing properties of a page item 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. When would you use the scope “Application”? An example would be to store the name of a mail server or some other URL which will be the same for all usages of the plug-in.

13. Click **Create and Create Another**

14. Enter the following values

Scope: **Component**

Attribute: **2**

Prompt: **Placeholder**

Type: **Text**

Required: **Yes**

Translatable: **No**

Display Length: **1**

Max Length: **1**

Default Value: **_**

15. Click **Create**

16. Go to section **Source**

17. Paste the code which you can find in the file **mask_plugin.sql** into the **PL/SQL Code** field.

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

18. Go to section **Callbacks**

19. Enter **render_mask** into the field **Render Procedure Name** and **validate_mask** into the field **Validation 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_mask` 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.

20. Click **Apply Changes**

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

22. In the Sample Application go to **page 7**

23. Edit **P7_PHONE_NUMBER1**

24. Select **Masked Text Field** in the Display As select list

25. As you can see, our plug-in attribute “Mask” is displayed in the **Element** section

26. The default mask is fine, let's **Apply Changes** and **run** the page.

27. If you are done with testing, go again to **Shared Components\User Interface\Plug-ins\Masked Text Field**

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

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

30. 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