Best Practices & Community

As with all programming languages and frameworks, there are some best practices to help developers create solid and robust plug-ins. These best practices are just recommendations and can vary by organization depending on your individual requirements. You may find that some of the best practices described in this chapter are not worded exactly as they are in your organization. That’s perfectly fine, as they are meant to be a guideline. Adopt them as necessary to suit your team’s needs.

APEX has a vibrant and vocal development community. When plug-ins were first introduced, various plug-in–specific web sites were created to help share plug-ins and demonstrate different techniques. Reach out to this community when you need support. Give back when you are able to provide support for others.

---

**Note** Special thanks to Patrick Wolf and Dan McGhan for providing some of their thoughts for this chapter.

---

When to Create a Plug-in

So, when exactly do you create a plug-in? This is a very common question that developers ask about plug-ins. The answer is, as Oracle guru Tom Kyte would say, “It depends.” There’s no set answer, but the following are some things to consider when responding to the question:

- **Time**: i.e., do you have the time to write this plug-in? If you’re in a time crunch, you may not be able to spend the time writing a full-blown plug-in. Alternatively, if you have time restrictions, you may consider writing a plug-in with the minimal set of features to get you going. When you have more time, then you can go back and add additional features as required.

- **Cost of not writing plug-in**: What would be the cost of not writing a plug-in? Sometimes, writing a plug-in may seem like extra work and unnecessary when you’re pressed for time. If you’re doing a bunch of hacks just to avoid writing a plug-in, then it will probably cost you more in the long run. This is similar to the idiom “one step forward, two steps back.”
**Reusability:** As with code modularization, if you plan to reuse certain functionality and it makes sense to do so, write a plug-in. If you’re unsure in a particular situation, hold off and wait to see if you reuse the same code somewhere else. If so, then you should probably write a plug-in. The main thing to remember is not to start creating your own custom framework for a workaround when one already exists.

**Moving parts:** If without a plug-in your code has to make a lot of assumptions and has a lot of *moving parts*, you may want to write a plug-in. Encapsulating all of the code in a single location will allow you to better control how things.

It is also important not to get carried away by developing plug-ins for things that are already supported. For instance, you should not write a dynamic action plug-in for a browser alert message.

---

**Be Aware About Security**

One side effect of creating your own plug-ins is that you may introduce various security exploits in your application without knowing it. The following subsections discuss some common security mistakes and how to avoid them when creating a plug-in.

**Cross-Site Scripting Attacks**

Cross-site scripting, commonly referred to as XSS, is when a user puts malicious JavaScript code on your website to silently steal information. A good opportunity for this is an application which allows users to enter comments at the bottom of the page. When other users view the page, they’ll see the comments posted by all users. If the comment values are not escaped, a malicious user can put code into a comment which can send the malicious user private information about any user currently logged in.

To prevent users from entering code into text fields and having that code executed, you can escape user input when it’s being displayed. To do this you can use the `APEX_PLUGIN_UTIL.ESCAPE` function. Instead of always forcing escaped values, you should use the plug-in `escape` variable. For example, in item type plug-ins, APEX developers can choose to escape the values as shown in Figure 7-. The value of the `escape_special_characters` checkbox in the figure is reflected in the corresponding `p_item.escape_output` variable accessible from within the plug-in functions.

---