When to Use Plug-ins

Plug-ins can be used to achieve some of the similar goals as with workflows or client-side code such as JavaScript, Silverlight, or HTML.

Plug-in vs. Client-side Code

The differentiation is clear. Plug-in runs server-side with compiled code and should always be the preferred method of implementing business logic. Client-side code is vulnerable to different client environment, browsers, browser settings, etc, and should not be relied on implementing business critical steps.

On the other hand, anything requiring UI or user interaction, must be done with client side code. You will not be able to modify UI or prompt the user during processing with the plug-in.

Plug-in vs. Processes

These are very similar, in fact workflows are just a skin on an asynchronous system plug-in. The main differences:

- Workflows can only be triggered asynchronously
- Workflows can only be triggered online
- Workflows have limited triggers. Mostly these would be your basic create/update/delete/assign/state change.
- Plugins support different event execution pipelines. Workflows are always post operation.
- Workflow execution logic can be easily modified, this allows encapsulation of code in custom workflow activity while still retaining possibility of frequent changes on execution logic by non-programmers.

Synchronous vs. Asynchronous Plug-ins

Synchronous plug-ins are executed by the CRM Core System. Synchronous execution means that triggering event will wait until the plug-in finishes the execution. For example, if we have synchronous plug-in that triggers on the creation of account and user creates account record in CRM, the form will hang at save until the plugin has finished.

Asynchronous plug-ins are executed by asynchronous service. Asynchronous plug-ins allow triggering event to finish before plug-in code runs. Therefore these can never be used to prevent an action, validate data entry, or provide any error messages back to the user.

Event Pipeline

The event pipeline allows you to configure when in the event the plug-in code will execute. The event pipeline is divided into the following events and stages:

Pre-event/Pre-Validation

This stage executes before anything else, even before basic validation if the triggering action is even allowed based on security. Therefore, it would be possible to trigger the plug-in code even without actually having permission to do so and great consideration must be used when writing a pre-validation plug-in. Also, execution in this stage might not be part of the database transaction.

Example uses:

Some "delete" plug-ins. Deletion cascades happen prior to pre-operation, therefore if you need any information about the child records, the delete plugin must be pre-validation.

Pre-event/Pre-Operation

This stage executes after validation, but before the changes has been committed to database. This is one of the most commonly used stages.

Example uses:

If and "update" plug-in should update the same record, it is best practice to use the pre-operation stage and modify the properties. That way the plug-in update is done within same DB transaction without needing additional web service update call.

Platform Core Operation

This stage is the in-transaction main operation of the system, such as create, update, delete, and so on. No custom plug-ins can be registered in this stage. It is meant for internal use only.

Post-Event

This stage executed after changes have been committed to database. This is one of the most used stages.

Example uses:

Most of the "Create" plugins are post-event. This allows access to the created GUID and creation of relationships to newly created record.

Post-Event/Post-Operation (deprecated)

This stage only supports Microsoft Dynamics CRM 4.0 based plug-ins.

Plug-in Messages

Plug-in Message is the triggering event, such as **Update** or **Create**. There are hundreds of these. The CRM SDK has a list of them all in SDK/message-entity support for plug-ins.xlsx. Here are some of the most commonly used messages:

1. **Create**
   Triggered when the record is created. All set attribute values are included in the target entity.
2. **Update**
   Triggered when the record is updated. You can define which attribute updates trigger the plug-in, by defining the filtering attributes. Also, only the updated attribute values are included in the target entity (see images)
3. **Delete**
   Triggered when the record is deleted.
4. **Retrieve**
   Triggered when record is retrieved, for example when user opens a record on a form.
5. **RetrieveMultiple**
   Triggered when a record set is retrieved using RetrieveMultiple, for example showing a view.

Plug-in Images (Pre vs. Post)

Images are snapshots of the entity's attributes, before and after the core system operation. Following table shows when in the event pipeline different images are available:

Message	Stage	Pre-Image	Post-Image
Create	PRE	No	No
Create	POST	Yes	Yes
Update	PRE	Yes	No
Update	POST	Yes	Yes
Delete	PRE	Yes	No
Delete	POST	Yes	No

The benefits of images

- One of the best uses for this is in update plug-ins. As mentioned before, update plug-in target entity only contains the updated attributes. However, often the plug-in will require information from other attributes as well. Instead of issuing a retrieve, the best practice is to push the required data in an image instead.

- Comparison of data before and after. This allows for various audit-type plugins, that logs what the value was before and after, or calculating the time spent in a stage or status.

The next section of this chapter covers how to start developing a plug-in, plus some examples of plug-ins.