**Understanding Plug-ins in MS CRM 2011**

What is a Plug-in??

**A plug-in is custom business logic (code) that you can integrate with Microsoft Dynamics CRM 2011 and Microsoft Dynamics CRM Online to modify or augment the standard behavior of the platform**. Another way to think about plug-ins is that they are handlers for events fired by the Microsoft Dynamics CRM platform. You can subscribe, also known as registering, a plug-in to a known set of events to have your code run when the event occurs.

CRM Event Pipeline

* Calls to the CRM web services execute some message.
* Each message creates a pipeline made up of various stages.
* Plug-ins can be registered on different stages in the pipeline
* A CRM “2011” plug-in is a .NET class that implements**Microsoft.Xrm.Sdk.IPlugin**
* Assemblies registered with CRM and stored in database, GAC or on disk

CRM Event Pipeline Visualization

• **PreEvent**

• Stage 10: Pre-validation

• **PreEvent**

• Stage 20: Pre-operation

• **Core Operation**

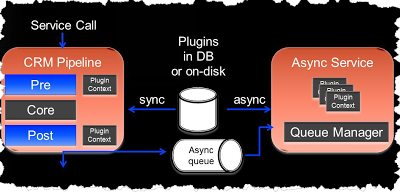
• Stage 30: MainOperation

• **PostEvent**

• Stage 40: Post-operation

• **PostEvent**

• Stage 50: Post-operation (deprecated\*)



*Note:* 20,30,40 are ***Transaction Scope***

Parent/Child Pipelines

* Parent and Child Pipelines no longer exist
* Pre-event Plugin in Parent Pipeline in CRM 4.0

‒ Pre-validation in CRM 2011 (outside transaction)

* Pre-event Plugin in Child Pipeline in CRM 4.0

‒ Pre-operation in CRM 2011 (inside transaction)

Sync/Async Execution

* Plugins can execute

- Synchronously during pipeline execution

- Asynchronously – queued for later execution

Plug-ins in CRM4

* Plug-ins ran in the same process context and not isolated
* Plug-ins could not participate in SQL transactions
* Plug-in registration had to be done by a Deployment Administrator

Plug-ins in CRM 2011

* Able to participate in SQL transactions
* Able to create traces returned with exceptions
* Plug-in assemblies can have 2 isolation modes:

- None or Sandbox

* The addition of the Sandbox isolation mode enables the use of plug-ins in CRM Online.

- Custom workflow activities will not be enabled in CRM Online for CRM 2011

Writing a Plugin

* Implement *IPlugin*

‒ Just the *Execute* method

* Constructor can take two strings

‒ Unsecure config

‒ Secure config

* Work with data from *IPluginExecutionContext*

‒ *Target,* pre-images and post-images etc…

‒ Handed a late bound *Entity*Can transform to early bound type using generic method  
*targetEntity.ToEntity()*

* Ensure plugin code is stateless

‒ Do not assume state is maintained between executions

Pre/Post Event Images

* Snapshot of Entity state at that point in time
* Removes need to call RetrieveRequest to get entity state during execution
* Attributes to be imaged must be specified in registration

*- SdkMessageProcessingStepRegistration.Images*

* Available via properties at runtime

- Pre: *IPluginExecutionContext.PreEntityImages*

- Post: *IPluginExecutionContext.PostEntityImages*

Offline Support

* Offline plugins execute in context of CRM Outlook Add-in
* Plugins can be: Online/Offline/Both
* Check if offline via  
  *%ExecutionContext%.IsExecutingOffline*
* Offline plugins should be idempotent

- May be executed twice

- Use *IsOfflinePlayback* property to check

Transaction Support

* CRM 2011 Stages support plugin execution inside the database transaction

- Stages 20 & 40

* Uncaught exceptions force a rollback
* *IExecutionContext.IsInTransaction*
* Transaction spans CRM DB operations only

- No distributed transaction support

* Plugins registered outside transaction stage **will** participate in transaction if executed as nested pipeline of transactional parent

**Using the Plug-in Registration Tool for Microsoft Dynamics CRM 2011 and Microsoft Dynamics CRM Online**

Do you have a need to extend Microsoft Dynamics CRM 2011 and Microsoft Dynamics CRM Online by writing and integrating custom code? Whether you are writing a plug-in or a custom workflow activity, you will use the Plug-in Registration tool, provided in the Microsoft Dynamics CRM SDK, to register your custom code with the Microsoft Dynamics CRM system. This article complements the documentation in the SDK by providing detailed information on how to use the Plug-in Registration tool to register custom code with Microsoft Dynamics CRM 2011 and Microsoft Dynamics CRM Online. However, registering custom workflow activities with Microsoft Dynamics CRM Online is not supported using the registration tool or any other means.

**Introduction**

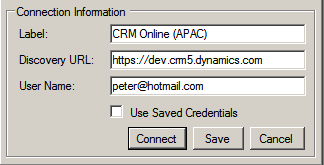
A plug-in is custom business logic that you can integrate with Microsoft Dynamics CRM 2011 and Microsoft Dynamics CRM Online to modify or augment the standard behavior of the system. Plug-ins are event handlers. They execute in response to a particular event being fired by the platform. You can also write custom workflow activities to add new functionality to [processes](http://msdn.microsoft.com/en-us/library/gg328264.aspx) (formerly known as workflows) that you create.

After you write (or purchase) a plug-in or custom workflow activity, you must register it with Microsoft Dynamics CRM. The Plug-in Registration tool is one of three ways you can register plug-ins and custom workflow activities. The other two methods are: by adding a registered plug-in or custom workflow activity to a [solution](http://msdn.microsoft.com/en-us/library/gg334530.aspx) and installing that solution or by writing custom code that uses the SDK plug-in [registration classes](http://msdn.microsoft.com/en-us/library/gg309563.aspx). When developing a plug-in or custom workflow activity, the most common registration method is to use the tool. Later when you are ready to package your code for production deployment, you can make use of solutions.

This article complements the plug-in registration [walkthrough](http://msdn.microsoft.com/en-us/library/gg309580.aspx) provided in the SDK by providing more information about the Plug-in Registration tool itself, where the walkthrough focuses more on the process of plug-in registration. The information in this article applies to all three Microsoft Dynamics CRM deployments: On-premises, IFD, and Online.

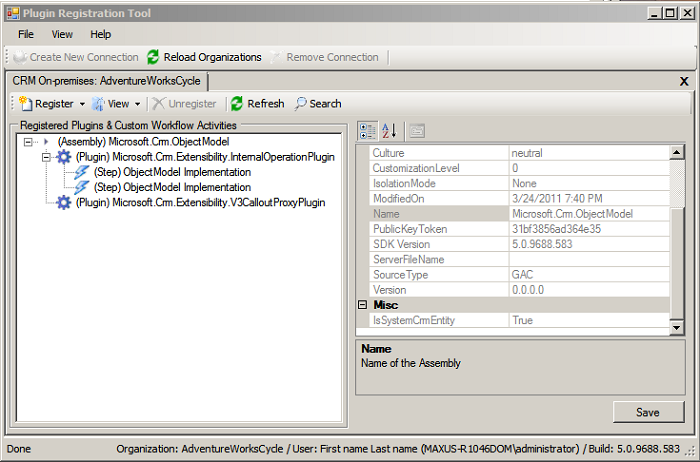
**Connecting to a Microsoft Dynamics CRM Server**

Enter the connection information for your target Microsoft Dynamics CRM server in the **Connection Information** part of the **Connections** pane, and then select **Connect**. Notice that you are just entering the http[s]://server-name[:port] part of the discovery URL. Enter the user name and password of your Microsoft Dynamics CRM user account as appropriate for the type of server you are connecting to.



A list of organizations that you are a member of is shown in the **Connections** pane. Select an organization from the list and click **Connect** or just double-click the organization’s name. A view of registration information related to the selected server and organization is opened in a tabbed pane. Each server and organization you connect to gets its own tab. Select **Save** if you want to save this connection information for when the tool is run again.

**The Main Window**

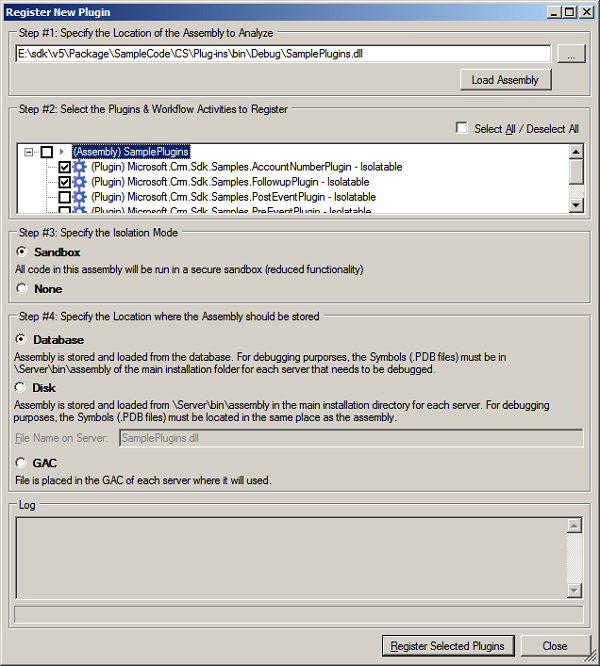


By default, the Microsoft.Crm.ObjectModel assembly and its related plug-ins and steps are registered with the system. The tool does not allow you to unregister these objects because they are required by Microsoft Dynamics CRM.

You can expand the nodes in the **Registered Plugins & Custom Workflow Activities** list and select any list item to view registration information about that selected item. You can change the way the list is displayed, according to assembly, entity, or message, by selecting a view in the **View** menu.

**Registering a Plug-in or Custom Workflow Activity**

To register a plug-in or custom workflow activity, you must first register the assembly containing that compiled code. You register an assembly by selecting **Register New Assembly** in the **Register** menu and then filling out the dialog box that is displayed.

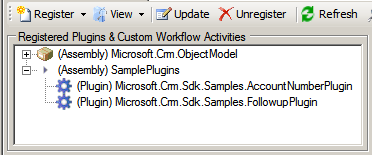


At the top of the figure in **Step #1**, I selected the ellipsis button **…** to the right of the text field and navigated to a plug-in assembly that I built from the SampleCode\CS\Plugins folder of the SDK. Next, in **Step #2**, I checked two plug-ins to register: AccountNumberPlugin and FollowupPlugin. If instead of plug-ins these were custom workflow activities, I could then just click **Register Selected Plug-ins** and be done. However, when registering plug-ins, you should proceed to **Step #3** and **Step #4**.

In **Step #3**, check **Sandbox** to register the selected plug-ins to execute in the [sandbox](http://msdn.microsoft.com/en-us/library/gg334752.aspx), an isolated run-time environment. Otherwise, select **None**. Always specify **None** when registering a custom workflow activity.

In **Step #4**, specify where on the server you want the plug-in assembly [deployed](http://msdn.microsoft.com/en-us/library/gg309620.aspx): in the Microsoft Dynamics CRM database, on the server’s disk under <crmwebroot>\server\bin\assembly, or in the GAC. The most commonly specified option is **Database** while the **Disk** option is maintained mostly for backward compatibility with Microsoft Dynamics CRM 4.0. If registering your plug-in or custom workflow activity in the database, you can still debug it by following the procedure described in the SDK topic [Debug a Plug-in](http://msdn.microsoft.com/en-us/library/gg328574.aspx).

Finally, select **Register Selected Plugins**. The assembly and selected plug-ins or custom workflow activities are displayed in the **Registered Plugins & Custom Workflow Activities** list. However, any registered plug-ins will not execute until you register a step for the plug-in as described in the next topic. On the other hand, custom workflow activities are ready to be included in a workflow.

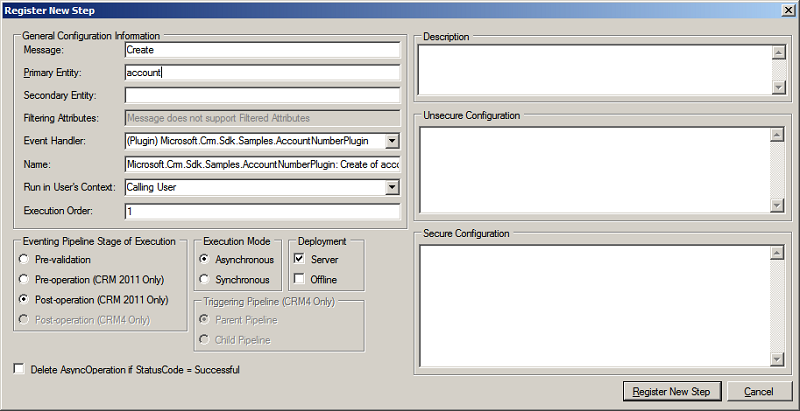


To update or unregister items in the list, other than a system required item, select the item and click **Update** or **Unregister**. Alternately, for an update, simply double-click the item. Notice that to update a plug-in you have to update the registration of its assembly.

**Registering a Step**

After you have registered an assembly and one or more plug-ins, you can register a step for each plug-in. A step defines when and how you want the plug-in to execute. A plug-in will not execute unless there is one or more steps registered for it.

In the view **Display by Assembly**, select a plug-in in the list and then select **Register New Step** in the **Register** menu. The **Register New Step** dialog box is displayed.



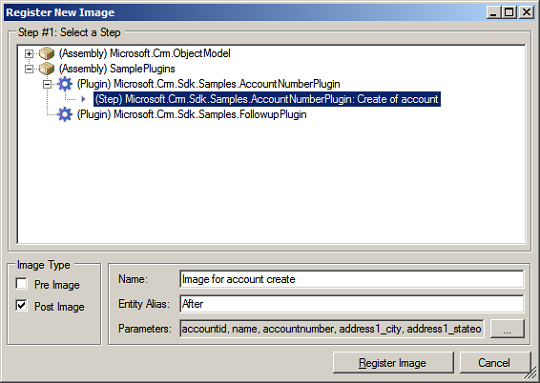
The following tables describe the values you can enter or set in the dialog box.

|  |  |
| --- | --- |
| **Control Label** | **Description** |
| Message | The message that must be processed by the Microsoft Dynamics CRM execution [pipeline](http://msdn.microsoft.com/en-us/library/gg327941.aspx) for the plug-in to execute. See the [list](http://msdn.microsoft.com/en-us/library/gg328576.aspx) of messages that support plug-ins. |
| Primary Entity | The [entity](http://msdn.microsoft.com/en-us/library/gg334400.aspx) that must be processed by the execution pipeline for the plug-in to execute. No entity value in this field indicates any/all entities. |
| Secondary Entity | Some messages require a secondary entity. See the documentation on the specific message for more information. |
| Filtering Attributes | A list of entity attributes that, when changed, cause the plug-in to execute. A value of **null** causes the plug-in to execute if any of the attributes change. When you specify a message that supports filtering attributes, for example [Update](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.messages.updaterequest.aspx), an ellipsis **…** button is displayed that, when clicked, allows you to select attributes from a list. |
| Event Handler | The plug-in that is to be executed. |
| Name | A name for the step. The tool sets a suggested name in the field automatically, but you can change it. |
| Run in User’s Context | Specify the system account that owns any data changes the plug-in makes. The **Calling User** is the logged on user that initiated the message to be processed. For more information, see the SDK topic [Impersonation in Plug-ins](http://msdn.microsoft.com/en-us/library/gg309416.aspx). |
| Execution Order | Specifies the order, also known as rank, that plug-ins are executed within a pipeline [stage](http://msdn.microsoft.com/en-us/library/gg327941.aspx). Plug-ins registered with an order value of 1 are executed first, followed by plug-ins registered with an order of 2, and so on. However, if there is more than one plug-in in a stage with the same order value, then the plug-in with the earliest compilation date is called first. |
| Eventing Pipeline Stage of Execution | Specifies when you want the plug-in to execute: before (Pre) or after (Post) the core operation that processes the message. For more information, see the SDK topic [Event Execution Pipeline](http://msdn.microsoft.com/en-us/library/gg327941.aspx). |
| Execution Mode | For plug-ins registered to execute in a Post stage, specify if you want them to execute immediately (Synchronous) or queued to execute later (Asynchronous). For post operations, specifying asynchronous execution results in improved system performance as compared to synchronous execution. |
| Deployment | Specify where you want the plug-in deployed: on the server, on the Microsoft Dynamics CRM for Outlook with Offline Access, or both. |
| Triggering Pipeline (CRM4 Only) | For plug-ins developed using the Microsoft Dynamics CRM 4.0 SDK, the plug-ins can execute in one of two supported pipelines: parent and child. For more information, see the Microsoft Dynamics CRM 4.0 SDK topics [Parent and Child Pipelines](http://msdn.microsoft.com/en-us/library/cc151083.aspx) and [Execution of Microsoft Dynamics CRM 4.0 Plug-ins](http://msdn.microsoft.com/en-us/library/gg309351.aspx). |
| Delete AsyncOperation if StatusCode = Successful | When any asynchronous operation completes, a **System Job** entity is created to record the completion status. You can view these system jobs in the Web application by selecting **Settings**, and then click **System Job**. Check this option if you want plug-in related system jobs automatically deleted when the status is successful. |
| Description | A description of the step. Typically, you can describe the purpose of the step for other developers or administrators that might run the registration tool at a later date. |
| Unsecure Configuration Secure Configuration | If you have written a constructor for the plug-in that takes one or two string arguments, any string values you specify in these fields are passed to the plug-in constructor at run-time. For more information, see the SDK topic [Write a Plug-in](http://msdn.microsoft.com/en-us/library/gg328263.aspx). |

After you register a step, you will see a new step item under the node of the associated plug-in in the **Registered Plugins & Custom Workflow Activities** list.

**Registering an Image**

To register an image, select **Register New Image** from the **Register** menu. The **Register New Image** dialog box is displayed.



**Follow these steps to register an image with a step.**

1. Select an existing step in the list.
2. Specify an image type. A pre-image is a snapshot of the entity’s attribute before the core operation. A post-image is a snapshot of the entity’s attribute after the core operation.
3. Enter a descriptive name for the image.
4. Enter an entity alias. The entity alias value you specify is used by your plug-in code as the key into the image collection.
5. Click the ellipsis **…** button to select the primary entity attributes that should be included in the image. For improved system performance, only select those attributes that you need.
6. Click **Register Image**.

**Registering a Windows Azure Endpoint**

When you register a service endpoint you are providing Microsoft Dynamics CRM with the information it needs to communicate with a Windows Azure AppFabric solution endpoint. Once established, this communication channel supports posting the execution context of the message currently being processed by Microsoft Dynamics CRM to the AppFabric Service Bus.

Step-by-step instructions about how to register a service endpoint, configure Windows Azure AppFabric ACS, and register a plug-in to post the execution context can be found in the SDK topic [Walkthrough: Register an Azure-aware Plug-in with Plug-in Registration Tool](http://msdn.microsoft.com/en-us/library/gg328524.aspx).

This article provides detailed information about the features of the Plug-in Registration tool and an explanation of the various form fields, settings, and data entry values. The topics covered also provide links to the related SDK documentation for more in-depth research. Try registering some of the sample plug-ins or custom workflow activities provided in the SDK download. Soon, you will be writing and registering your own custom plug-ins or custom workflow activities.

Peter Hecke is a programming writer on the Microsoft Dynamics CRM SDK team and is co-author of the SDK since version 1.3. He has also written numerous MSDN technical articles about Microsoft Dynamics CRM.

**Understand the data context passed to a plug-in**

When a plug-in is run in response to an execution pipeline event for which it is registered, the plug-in’s [Execute](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iplugin.execute.aspx) method is called. That method passes an [IServiceProvider](http://msdn.microsoft.com/en-us/library/system.iserviceprovider.aspx) object as a parameter, which contains a number of useful objects. The following sections describe some of the information that is passed to a plug-in when executed.

**Access the plug-in execution context**

[IPluginExecutionContext](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.ipluginexecutioncontext.aspx) contains information that describes the run-time environment that the plug-in executes, information related to the execution pipeline, and entity business information. The context is contained in the [System.IServiceProvider](http://msdn.microsoft.com/en-us/library/system.iserviceprovider.aspx) parameter that is passed at run time to a plug-in through its [Execute](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iplugin.execute.aspx) method.

// Obtain the execution context from the service provider.

IPluginExecutionContext context = (IPluginExecutionContext)

serviceProvider.GetService(typeof(IPluginExecutionContext));

When a system event is fired that a plug-in is registered for, the system creates and populates the context and passes it to a plug-in through the previously mentioned classes and methods. The execution context is passed to each registered plug-in in the pipeline when they are executed. Each plug-in in the execution pipeline is able to modify writable properties in the context. For example, given a plug-in registered for a pre-event and another plug-in registered for a post-event, the post-event plug-in can receive a context that has been modified by the pre-event plug-in. The same situation applies to plug-ins that are registered within the same stage.

All the properties in [IPluginExecutionContext](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.ipluginexecutioncontext.aspx) are read-only. However, your plug-in can modify the contents of those properties that are collections. For more information about infinite loop prevention, see [Depth](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iexecutioncontext.depth.aspx).

**Access the Organization service**

To access the Microsoft Dynamics CRM organization service, it is required that plug-in code create an instance of the service through the [ServiceProvider.GetService](http://msdn.microsoft.com/en-us/library/bb138962(VS.80).aspx) method.

// Obtain the organization service reference.

IOrganizationServiceFactory serviceFactory = (IOrganizationServiceFactory)serviceProvider.GetService(typeof(IOrganizationServiceFactory));

IOrganizationService service = serviceFactory.CreateOrganizationService(context.UserId);

The platform provides the correct web service URLs and network credentials for you when you use this method. Instantiating your own Web service proxy is not supported as it will create deadlock and authentication issues.

**Access the Notification service**

Synchronous registered plug-ins can post the execution context to the Microsoft Azure Service Bus. The service provider object that is passed to the plug-in contains a reference to [IServiceEndpointNotificationService](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iserviceendpointnotificationservice.aspx). It is through that notification service that synchronous plug-ins can send brokered messages to the Microsoft Azure Service Bus. For more information about Microsoft Azure, see [Introduction to Microsoft Azure integration with Microsoft Dynamics CRM](http://msdn.microsoft.com/en-us/library/gg334766.aspx). For more information about writing a plug-in that can post to the Microsoft Azure Service Bus, see [Write a custom Azure-aware plug-in](http://msdn.microsoft.com/en-us/library/gg328194.aspx).

**Input and output parameters**

The [InputParameters](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iexecutioncontext.inputparameters.aspx) property contains the data that is in the request message currently being processed by the event execution pipeline. Your plug-in code can access this data. The property is of type [ParameterCollection](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.parametercollection.aspx) where the keys to access the request data are the names of the actual public properties in the request. As an example, take a look at [CreateRequest](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.messages.createrequest.aspx). One property of [CreateRequest](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.messages.createrequest.aspx) is named [Target](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.messages.createrequest.target.aspx), which is of type [Entity](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.entity.aspx). This is the entity currently being operated upon by the platform. To access the data of the entity you would use the name “Target” as the key in the input parameter collection. You also need to cast the returned instance.

// The InputParameters collection contains all the data passed in the message request.

if (context.InputParameters.Contains("Target") &&

context.InputParameters["Target"] is Entity)

{

// Obtain the target entity from the input parameters.

Entity entity = (Entity)context.InputParameters["Target"];

Note that not all requests contain a **Target** property that is of type [Entity](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.entity.aspx), so you have to look at each request or response. For example, [DeleteRequest](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.messages.deleterequest.aspx) has a [Target](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.messages.deleterequest.target.aspx) property, but its type is [EntityReference](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.entityreference.aspx). The preceding code example would be changed as follows.

// The InputParameters collection contains all the data passed in the message request.

if (context.InputParameters.Contains("Target") && context.InputParameters["Target"] is EntityReference)

{

// Obtain the target entity from the input parameters.

EntityReference entity = (EntityReference)context.InputParameters["Target"];

}

Similarly, the [OutputParameters](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iexecutioncontext.outputparameters.aspx) property contains the data that is in the response message, for example [CreateResponse](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.messages.createresponse.aspx), currently being passed through the event execution pipeline. However, only synchronous post-event and asynchronous registered plug-ins have [OutputParameters](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iexecutioncontext.outputparameters.aspx) populated as the response is the result of the core platform operation. The property is of type [ParameterCollection](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.parametercollection.aspx) where the keys to access the response data are the names of the actual public properties in the response.

**Pre and post entity images**

[PreEntityImages](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iexecutioncontext.preentityimages.aspx) and [PostEntityImages](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iexecutioncontext.postentityimages.aspx) contain snapshots of the primary entity's attributes before (pre) and after (post) the core platform operation. Microsoft Dynamics CRM populates the pre-entity and post-entity images based on the security privileges of the impersonated system user. Only entity attributes that are set to a value or are **null** are available in the pre or post entity images. You can specify to have the platform populate these **PreEntityImages** and **PostEntityImages** properties when you register your plug-in. The entity alias value you specify during plug-in registration is used as the key into the image collection in your plug-in code.

There are some events where images aren’t available. For example, only synchronous post-event and asynchronous registered plug-ins have [PostEntityImages](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iexecutioncontext.postentityimages.aspx) populated. The create operation doesn’t support a pre-image and a delete operation doesn’t support a post-image. In addition, only a small subset of messages support pre and post images as shown in the following table.

|  |  |  |
| --- | --- | --- |
| **Message Request** | **Property** | **Description** |
| [AssignRequest](http://msdn.microsoft.com/en-us/library/microsoft.crm.sdk.messages.assignrequest.aspx) | Target | The assigned entity. |
| [CreateRequest](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.messages.createrequest.aspx) | Target | The created entity. |
| [DeleteRequest](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.messages.deleterequest.aspx) | Target | The deleted entity. |
| [DeliverIncomingEmailRequest](http://msdn.microsoft.com/en-us/library/microsoft.crm.sdk.messages.deliverincomingemailrequest.aspx) | EmailId | The delivered email ID. |
| [DeliverPromoteEmailRequest](http://msdn.microsoft.com/en-us/library/microsoft.crm.sdk.messages.deliverpromoteemailrequest.aspx) | EmailId | The delivered email ID. |
| [ExecuteWorkflowRequest](http://msdn.microsoft.com/en-us/library/microsoft.crm.sdk.messages.executeworkflowrequest.aspx) | Target | The workflow entity. |
| [MergeRequest](http://msdn.microsoft.com/en-us/library/microsoft.crm.sdk.messages.mergerequest.aspx) | Target | The parent entity, into which the data from the child entity is being merged. |
| [MergeRequest](http://msdn.microsoft.com/en-us/library/microsoft.crm.sdk.messages.mergerequest.aspx) | SubordinateId | The child entity that is being merged into the parent entity. |
| [SendEmailRequest](http://msdn.microsoft.com/en-us/library/microsoft.crm.sdk.messages.sendemailrequest.aspx) | EmailId | The sent entity ID. |
| [SetStateRequest](http://msdn.microsoft.com/en-us/library/microsoft.crm.sdk.messages.setstaterequest.aspx) | EntityMoniker | The entity for which the state is set. |
| [UpdateRequest](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.messages.updaterequest.aspx) | Target | The updated entity. |

Registering for pre or post images to access entity attribute values results in improved plug-in performance as compared to obtaining entity attributes in plug-in code through [RetrieveRequest](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.messages.retrieverequest.aspx) or [RetrieveMultipleRequest](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.messages.retrievemultiplerequest.aspx) requests.

|  |
| --- |
| **Security Note** :A pre-image passed in the execution context to a plug-in or custom workflow activity might contain data that the logged-on user doesn't have the privileges to access. Microsoft Dynamics CRM administrators and other users with high-level permissions can register plug-ins to run under the “system” user account or plug-in code can make calls as a “system” user on behalf of the logged-on user. If this happens, logged-on users can access data that their field level security does not allow access to. |

**Impersonation in plug-ins**

Impersonation is used to execute business logic (custom code) on behalf of a Microsoft Dynamics CRM system user to provide a desired feature or service for that user. Any business logic executed within a plug-in, including Web service method calls and data access, is governed by the security privileges of the impersonated user.

Plug-ins not executed by either the sandbox or asynchronous service execute under the security account that is specified on the **Identity** tab of the **CRMAppPool Properties** dialog box. The dialog box can be accessed by right-clicking the **CRMAppPool** application pool in Internet Information Services (IIS) Manager and then clicking **Properties** in the shortcut menu. By default, CRMAppPool uses the Network Service account identity but this can be changed by a system administrator during installation. If the **CRMAppPool** identity is changed to a system account other than Network Service, the new identity account must be added to the **PrivUserGroup** group in Active Directory. Refer to the “Change a Microsoft Dynamics CRM service account” topic in the Microsoft Dynamics CRM 2011 Implementation Guidefor complete and detailed instructions.

The two methods that can be employed to impersonate a user are discussed below.

[**Impersonation during plug-in registration**](javascript:void(0))

One method to impersonate a system user within a plug-in is by specifying the impersonated user during plug-in registration. When registering a plug-in programmatically, if the **SdkMessageProcessingStep.ImpersonatingUserId** attribute is set to a specific Microsoft Dynamics CRM system user, Web service calls made by the plug-in execute on behalf of the impersonated user. If **ImpersonatingUserId** is set to a value of **null** or **Guid.Empty** during plug-in registration, the calling/logged on user or the standard "system" user is the impersonated user.

Whether the calling/logged on user or "system" user is used for impersonation is dependent on the request being processed by the pipeline and is beyond the scope of the SDK documentation. For more information about the "system" user, refer to the next topic.

|  |
| --- |
| **NOTE**:When you register a plug-in using the sample plug-in registration tool that is provided in the SDK download, service methods invoked by the plug-in execute under the account of the calling or logged on user by default unless you select a different user in the **Run in User’s Context** dropdown menu. For more information about the tool sample code, refer to the tool code under the SDK\Tools\PluginRegistration folder of the SDK package. |

[**Impersonation during plug-in execution**](javascript:void(0)) **:**Impersonation that was defined during plug-in registration can be altered in a plug-in at run time. Even if impersonation was not defined at plug-in registration, plug-in code can still use impersonation. The following discussion identifies the key properties and methods that play a role in impersonation when making Web service method calls in a plug-in.

The platform passes the impersonated user ID to a plug-in at run time through the [UserId](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iexecutioncontext.userid.aspx) property. This property can have one of three different values as shown in the table below.

|  |  |
| --- | --- |
| **UserId Value** | **Condition** |
| Initiating user or "system" user | The **SdkMessageProcessingStep.ImpersonatingUserId** attribute is set to **null** or **Guid.Empty** at plug-in registration. |
| Impersonated user | The **ImpersonatingUserId** property is set to a valid system user ID at plug-in registration. |
| "system" user | The current pipeline was executed by the platform, not in direct response to a service method call. |

The [InitiatingUserId](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iexecutioncontext.initiatinguserid.aspx) property of the execution context contains the ID of the system user that called the service method that ultimately caused the plug-in to execute.

|  |
| --- |
| **ImportantImportant** For plug-ins executing offline, any entities created by the plug-in are owned by the logged on user. Impersonation in plug-ins is not supported while in offline mode. |
|  |

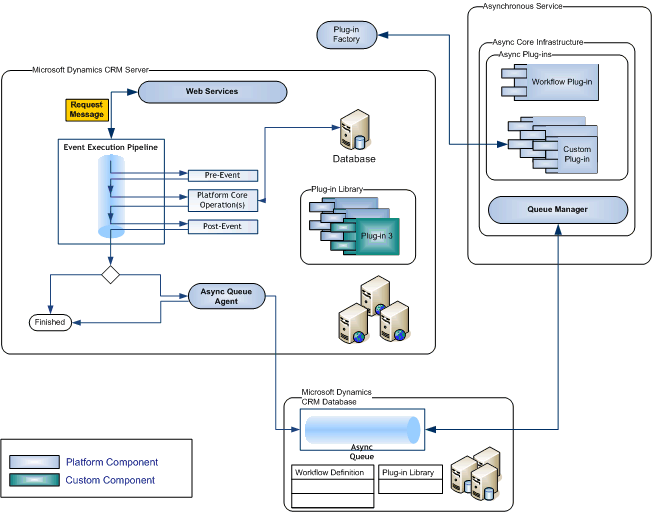
**Event execution pipeline**

The Microsoft Dynamics CRM event processing subsystem executes plug-ins based on a message pipeline execution model. A user action in the Microsoft Dynamics CRM Web application or an SDK method call by a plug-in or other application results in a message being sent to the organization Web service. The message contains business entity information and core operation information. The message is passed through the event execution pipeline where it can be read or modified by the platform core operation and any registered plug-ins.

NOTE: While there are several Web services hosted by the Microsoft Dynamics CRM platform, only events triggered by the organization and OData endpoints can cause plug-ins to execute.

## Architecture and related components

The following figure illustrates the overall architecture of the Microsoft Dynamics CRM platform with respect to both synchronous and asynchronous event processing.



The event execution pipeline processes events either synchronously or asynchronously. The platform core operation and any plug-ins registered for synchronous execution are executed immediately. Synchronous plug-ins that are registered for the event are executed in a well-defined order. Plug-ins registered for asynchronous execution are queued by the Asynchronous Queue Agent and executed at a later time by the asynchronous service.

|  |
| --- |
| **ImportantImportant** |
| Regardless of whether a plug-in executes synchronously or asynchronously, there is a 2 minute time limit imposed on the execution of a (message) request. If the execution of your plug-in logic exceeds the time limit, a [System.TimeoutException](https://msdn.microsoft.com/library/system.timeoutexception.aspx) is thrown. If a plug-in needs more processing time than the 2 minute time limit, consider using a workflow or other background process to accomplish the intended task. |

## Pipeline stages

The event pipeline is divided into multiple stages, of which 4 are available to register custom developed or 3rd party plug-ins. Multiple plug-ins that are registered in each stage can be further be ordered (ranked) within that stage during plug-in registration.

|  |  |  |  |
| --- | --- | --- | --- |
| **Event** | **Stage name** | **Stage number** | **Description** |
| Pre-Event | Pre-validation | 10 | Stage in the pipeline for plug-ins that are to execute before the main system operation. Plug-ins registered in this stage may execute outside the database transaction.   |  | | --- | | **securitySecurity Note** | | The pre-validation stage occurs prior to security checks being performed to verify the calling or logged on user has the correct permissions to perform the intended operation. | |
| Pre-Event | Pre-operation | 20 | Stage in the pipeline for plug-ins that are to execute before the main system operation. Plug-ins registered in this stage are executed within the database transaction. |
| Platform Core Operation | MainOperation | 30 | 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. For internal use only. |
| Post-Event | Post-operation | 40 | Stage in the pipeline for plug-ins which are to execute after the main operation. Plug-ins registered in this stage are executed within the database transaction. |

## Message processing

Whenever application code or a workflow invokes a Microsoft Dynamics CRM Web service method, a state change in the system occurs that raises an event. The information passed as a parameter to the Web service method is internally packaged up into a [OrganizationRequest](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.organizationrequest.aspx) message and processed by the pipeline. The information in the **OrganizationRequest** message is passed to the first plug-in registered for that event where it can be read or modified before being passed to the next registered plug-in for that event and so on. Plug-ins receive the message information in the form of context that is passed to their [Execute](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iplugin.execute.aspx) method. The message is also passed to the platform core operation.

## Plug-in registration

Plug-ins can be registered to execute before or after the core platform operation. Pre-event registered plug-ins receive the **OrganizationRequest** message first and can modify the message information before the message is passed to the core operation. After the core platform operation has completed, the message is then known as the [OrganizationResponse](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.organizationresponse.aspx). The response is passed to the registered post-event plug-ins. Post-event plug-ins have the opportunity to modify the message before a copy of the response is passed to any registered asynchronous plug-ins. Finally, the response is returned to the application or workflow that invoked the original Web service method call.

Because a single Microsoft Dynamics CRM server can host more than one organization, the execution pipeline is organization specific. There is a virtual pipeline for every organization. Plug-ins registered with the pipeline can only process business data for a single organization. A plug-in that is designed to work with multiple organizations must be registered with each organization's execution pipeline.

## Inclusion in database transactions

Plug-ins may or may not execute within the database transaction of the Microsoft Dynamics CRM platform. Whether a plug-in is part of the transaction is dependent on how the message request is processed by the pipeline. You can check if the plug-in is executing in-transaction by reading the [IsInTransaction](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iexecutioncontext.isintransaction.aspx) property inherited by [IPluginExecutionContext](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.ipluginexecutioncontext.aspx) that is passed to the plug-in. If a plug-in is executing in the database transaction and allows an exception to be passed back to the platform, the entire transaction will be rolled back. Stages 20 and 40 are guaranteed to be part of the database transaction while stage 10 may be part of the transaction.

Any registered plug-in that executes during the database transaction and that passes an exception back to the platform cancels the core operation. This results in a rollback of the core operation. In addition, any pre-event or post event registered plug-ins that have not yet executed and any workflow that is triggered by the same event that the plug-in was registered for will not execute.