**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-in/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-in/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-in/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-in/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-in/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-in/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-in/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-in/library/gg328194.aspx).

**Input and output parameters**

The [InputParameters](http://msdn.microsoft.com/en-in/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-in/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-in/library/microsoft.xrm.sdk.messages.createrequest.aspx). One property of [CreateRequest](http://msdn.microsoft.com/en-in/library/microsoft.xrm.sdk.messages.createrequest.aspx) is named [Target](http://msdn.microsoft.com/en-in/library/microsoft.xrm.sdk.messages.createrequest.target.aspx), which is of type [Entity](http://msdn.microsoft.com/en-in/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-in/library/microsoft.xrm.sdk.entity.aspx), so you have to look at each request or response. For example, [DeleteRequest](http://msdn.microsoft.com/en-in/library/microsoft.xrm.sdk.messages.deleterequest.aspx) has a [Target](http://msdn.microsoft.com/en-in/library/microsoft.xrm.sdk.messages.deleterequest.target.aspx) property, but its type is [EntityReference](http://msdn.microsoft.com/en-in/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-in/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-in/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-in/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-in/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-in/library/microsoft.xrm.sdk.iexecutioncontext.preentityimages.aspx) and [PostEntityImages](http://msdn.microsoft.com/en-in/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-in/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-in/library/microsoft.crm.sdk.messages.assignrequest.aspx) | Target | The assigned entity. |
| [CreateRequest](http://msdn.microsoft.com/en-in/library/microsoft.xrm.sdk.messages.createrequest.aspx) | Target | The created entity. |
| [DeleteRequest](http://msdn.microsoft.com/en-in/library/microsoft.xrm.sdk.messages.deleterequest.aspx) | Target | The deleted entity. |
| [DeliverIncomingEmailRequest](http://msdn.microsoft.com/en-in/library/microsoft.crm.sdk.messages.deliverincomingemailrequest.aspx) | EmailId | The delivered email ID. |
| [DeliverPromoteEmailRequest](http://msdn.microsoft.com/en-in/library/microsoft.crm.sdk.messages.deliverpromoteemailrequest.aspx) | EmailId | The delivered email ID. |
| [ExecuteWorkflowRequest](http://msdn.microsoft.com/en-in/library/microsoft.crm.sdk.messages.executeworkflowrequest.aspx) | Target | The workflow entity. |
| [MergeRequest](http://msdn.microsoft.com/en-in/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-in/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-in/library/microsoft.crm.sdk.messages.sendemailrequest.aspx) | EmailId | The sent entity ID. |
| [SetStateRequest](http://msdn.microsoft.com/en-in/library/microsoft.crm.sdk.messages.setstaterequest.aspx) | EntityMoniker | The entity for which the state is set. |
| [UpdateRequest](http://msdn.microsoft.com/en-in/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-in/library/microsoft.xrm.sdk.messages.retrieverequest.aspx) or [RetrieveMultipleRequest](http://msdn.microsoft.com/en-in/library/microsoft.xrm.sdk.messages.retrievemultiplerequest.aspx) requests.

|  |
| --- |
| **securitySecurity 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. More information:[Impersonation in plug-ins](http://msdn.microsoft.com/en-in/library/gg309416.aspx) |

**Event execution pipeline**

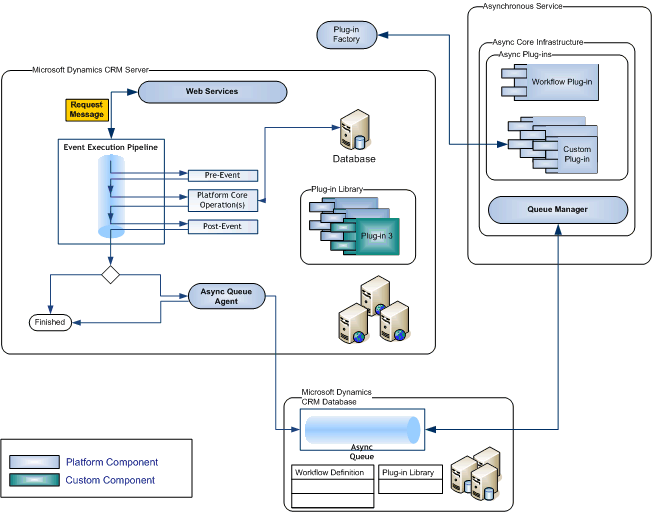
The Microsoft Dynamics CRM event processing subsystem executes plug-ins based on a message pipeline execution model. A user action in the Microsoft Dynamics CRM Web application or an SDK method call by a plug-in or other application results in a message being sent to the organization Web service. The 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.

Completed till here



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.