Plug-in

* Plug-in is a .NET class that implements IPlug-in interface.

Example of Syntax:

Public class MyFirstPlugin: IPlugin

{

Public void Execute(IServiceProvider serviceProvider)

{

//Plug-in Logic Code

}

}

IServiceProvider:

* It is a object parameter contains a number of useful objects.
  + IPluginExecutionContext: Information about the event that triggered the plugin (entity, message, stage, user)
  + IOrganizationServiceFactory: Creates IOrganizationService instances for CRUD operations
  + IOrganizationService: Used to interact with CRM data
  + ITracingService: For logging/debugging in Plugin Trace Log
  + IServiceEndpointNotificationService: It allows your plugin to send data to an **Azure Service Bus endpoint** that you’ve configured in Dynamics 365.

Context.Depth:

* It’s an **integer property** in the IPluginExecutionContext.
* It starts at **1** when your plugin is triggered **for the first time** in the current transaction.
* Each time **another plugin or workflow** triggers inside the same pipeline and calls another step, the depth increases by **+1**.
* Used mainly to **prevent infinite loops**.
* Example:

public void Execute(IServiceProvider serviceProvider)

{

var context = (IPluginExecutionContext)serviceProvider.GetService(typeof(IPluginExecutionContext));

var tracing = (ITracingService)serviceProvider.GetService(typeof(ITracingService));

tracing.Trace("Current Depth: {0}", context.Depth);

// Stop if plugin has been triggered more than once in the same transaction

if (context.Depth > 1)

{

tracing.Trace("Plugin exited to prevent recursion.");

return;

}

// Your plugin logic here

}

Input and Output Parameter

* Input Parameter: The data that is in the request message currently being processed by the Execution Pipeline.
  + It will have two keys **Target** and **OptionalParameters**
  + **Target** is, really, the in-memory representation of the data that’s being updated/created/deleted/etc. It’s literally the target of the action.

**Important Point**: Target is not there for every Plug-in execution. It depends on the Plug-in step configuration. Create/Update/Delete will have a Target.

How do you know what you will find in the ‘target’?

* For Create and Update Plug-ins, you will always have a ‘Target’, and it will be of ‘Entity’ type.
* For the ‘Delete’ plug-ins, you will also have a ‘Target’, but it will be of ‘EntityReference’ Type.
  + EntityReference: In Microsoft Dynamics 365, an **EntityReference** is basically a **lightweight pointer** to another record in the system.
  + Instead of containing all the fields of a record, it **just contains the minimum information needed to identify it**.
* Output Parameters: Data that is in the response message. It is populated by the platform and only contains valid data during the After Operation state.

Pre/Post – Event Images:

**1. Pre-Image**

* **What it is**: The values of the record **before** the operation happens.
* **When it’s captured**: Before the core operation executes in the database.
* **Why it’s used**:
  + Compare old and new values for auditing.
  + Use existing field values that are not part of Target (since Target only contains changed fields).
* **Example**:
  + On an Update of a Contact’s email:
    - **Pre-Image**: Email = "old@example.com"
    - **Target**: Email = "new@example.com"

**2. Post-Image**

* **What it is**: The values of the record **after** the operation happens.
* **When it’s captured**: After the core operation has been committed in the database (but before the response is sent back).
* **Why it’s used**:
  + Get the final saved values, including changes from plugins, workflows, or server-side logic.
  + Useful in asynchronous plugins for integration purposes.
* **Example**:
  + On an Update of a Contact’s email:
    - **Post-Image**: Email = "new@example.com" (and possibly other updated fields).

| **Message (Operation)** | **Pre-Image Available** | **Post-Image Available** |
| --- | --- | --- |
| **Create** | ❌ No | ✅ Yes |
| **Update** | ✅ Yes | ✅ Yes |
| **Delete** | ✅ Yes | ❌ No |
| **Retrieve** | ❌ No | ✅ Yes |
| **RetrieveMultiple** | ❌ No | ✅ Yes |

Examples of getting Services:

1. IPluginExecutionContext context = (IPlug-inExecutionContext)ServiceProvider.GetService(typeof(IPlug-inExecutionContext));
2. IOrganizationServiceFactory serviceFactory = (IOrganizationServiceFactory)serviceProvider.GetService(typeof(IOrganizationServiceFactory));
3. IOrganizationService service = serviceFactory.CreateOrganizationService(context.UserId);
4. ITracingService tracingService = (ITracingService)serviceProvider.GetService(typeof(ITracingService));
5. IServiceEndpointNotification azureEndpointService = (IServiceEndpointNotification)serviceProvider.GetService(typeof(IServiceEndpointNotification));

**Practical process:**

First search 'NuGet Package Manager' in main search bar (ctrl+q)

In this select: 'Manage NuGet Packages for Solution'

After select that in browser search:

Microsoft.CrmSdk.CoreAssemblies

and install.

This is the reference sdk assembly. This package added both reference file crm.sdk and xrm.sdk file.

**Post Operation Syntax**

if (context.Stage == 40)

{

Entity updateAccount = new Entity("account");

updateAccount.Id = entity.Id;

updateAccount.Attributes["telephone1"] = "1234567890";

service.Update(updateAccount);

}

**Pre Operation Syntax**

if(context.Stage == 20)

{

entity.Attributes.Add("telephone1", "0987654321");

}

**Plug-in Isolation**

* **Trusts:**
  + Full: plug-in registered outside of sandbox (Means on plugin registration time select none instead of sandbox)
    - Only on-premise
    - Wpw3.exe
  + Partial: Registered id sandbox
    - On-premise and online
    - Microsoft.Crm.Sandbox.WorkerProcess.exe

**Sandbox Architecture:**

* Has its own separate processing service
* Executes in partial trust with restrictions
* Allows for multiple process servers
* Has execution time limit of 2 minutes

**Sandbox Restrictions:**

* Only HTTP and HTTPS protocols are allowed
* Access to localhost is not permitted
* IP Address can’t be used. (must use a named web address that required DNS resolution)
* Anonymous authentication is supported and recommended. No provision for prompting for credentials or saving of credentials.

**Storage Specification for store dll file**

* Database = Plugin.dll uploaded into the database
* Disk = The Plugin.dll is saved on the CRM servers hard disk
* GAC (Global Assembly Casy) = The Plugin.dll is loaded into the GAC on the CRM server.

**Advantage of storing dll file in database:**

* The plugin is backed up when the database is backed up
* For multiple server configurations you only need to deploy once to the database and not individually to each CRM server.
* Plugins in the database can be added to solutions, Disk, GAC plugins can’t.
* Plugins deployed to the GAC or Disk will need an IISRESET to refresh, plugins deployed in the database do not.
* Sandboxed and CRM Online plugins have to be deployed in database.

**Basic Example of Plugin or Structure:**

using System;

using Microsoft.Xrm.Sdk;

namespace pluginProject1

{

    public class myFirstPlugin:IPlugin

    {

        public void Execute(IServiceProvider serviceProvider)

        {

            //ITracingService tracingService = (ITracingService)

            //    serviceProvider.GetService(typeof(ITracingService));

            IPluginExecutionContext context = (IPluginExecutionContext)

                serviceProvider.GetService(typeof(IPluginExecutionContext));

            if (context.MessageName != "Create")

                return;

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

                context.InputParameters["Target"] is Entity)

            {

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

                if (entity.LogicalName != "account")

                    return;

                IOrganizationServiceFactory serviceFactory =

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

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

                try

                {

                    if(context.Stage == 40)

                    {

                    }

                }

                catch (Exception ex)

                {

                    //tracingService.Trace("First Class Plugin: {0}", ex.ToString());

                    throw;

                }

            }

        }

    }

}

**🔹 High-Level Sales Funnel View**

1. **Lead → Capture interest**
2. **Opportunity → Qualify & pursue deal**
3. **Quote → Send proposal**
4. **Order → Confirm purchase**
5. **Invoice → Collect payment**
6. **Case/Service → After-sales support**

**🔹 Example with Real-World Flow**

1. **Customer fills inquiry form → CRM creates Lead.**
2. **Salesperson calls and verifies → Lead is Qualified.**
3. **System creates Contact + Account + Opportunity.**
4. **Sales team prepares Quote → sends to customer.**
5. **Customer accepts → Quote → converted to Order.**
6. **Order fulfilled → CRM creates Invoice.**
7. **After-sale issue → CRM logs a Case.**

** Multiple Lookups = many lookup fields, each to a single table.**

** Polymorphic Lookup = one lookup field, multiple possible tables (system-defined).**

** Multi-Select Lookup = one lookup field, multiple records allowed (new feature).**