Plugin Development Cheat Sheet for Dynamics 365 CE with Examples

1. Plugin Class Structure

Basic Plugin Template:

```
public class MyPlugin : IPlugin
{
    public void Execute(IServiceProvider serviceProvider)
    {
        // Plugin logic here
    }
}
```

`IPlugin` → interface that must be implemented

`Execute()` → main method called by platform

`IServiceProvider` → provides access to services

2. Getting Plugin Context

`IPluginExecutionContext` → current execution context

`IOrganizationServiceFactory` → create service instances

`ITracingService` → logging and debugging

```
var context = (IPluginExecutionContext)serviceProv
ider.GetService(typeof(IPluginExecutionContext));

var serviceFactory = (IOrganizationServiceFactory)
serviceProvider.GetService(typeof(IOrganizationSer
viceFactory));

var tracingService = (ITracingService)serviceProvi
der.GetService(typeof(ITracingService));
```

3. Context Properties

`context.MessageName` → current message (Create, Update, etc.)

`context.PrimaryEntityName` → entity being processed

`context.Stage` → execution stage (10, 20, 40)

`context.Mode` → synchronous (0) or asynchronous (1)

`context.Depth` → call depth (prevent infinite loops)

`context.UserId` → user triggering the event

`context.InitiatingUserId` → original user who started the process

4. Input/Output Parameters

Accessing Target Entity (Create/Update):

```
if (context.InputParameters.Contains("Target") && context.InputParameters["Target"] is Entity)
{
    Entity target = (Entity)context.InputParameters["Target"];
}
```

Getting Pre/Post Images:

```
Entity preImage = context.PreEntityImages["PreImage"];
Entity postImage = context.PostEntityImages["PostImage"];
```

Common Input Parameters:

```
// For Delete operations
EntityReference entityRef = (EntityReference)context.InputParameters["Target"];

// For Associate/Disassociate
EntityReference target = (EntityReference)context.InputParameters["Target"];
Relationship relationship = (Relationship)context.InputParameters["Relationship"];
```

5. Organization Service

`service.Create(entity)` → create new record

`service.Update(entity)` → update existing record

`service.Delete(entityName, id)` → delete record

`service.Retrieve(entityName, id, columns)` → get single record

`service.RetrieveMultiple(query)` → get multiple records

```
var service = serviceFactory.CreateOrganizationSer
vice(context.UserId);

// Create record
Entity newEntity = new Entity("contact");
newEntity["firstname"] = "John";
Guid id = service.Create(newEntity);
```

6. Common Checks

Prevent Infinite Loop:

```
if (context.Depth > 1) return;
```

Check Message Type:

```
if (context.MessageName != "Update") return;
```

Check Entity Type:

```
if (context.PrimaryEntityName != "account") retur
n;
```

Check Stage:

```
if (context.Stage != 20) return; // Pre-operation
```

7. Working with Fields

Check if field exists and get value:

```
if (target.Contains("name"))
{
    string name = target.GetAttributeValue<string>
("name");
}
```

Set field values:

```
target["telephone1"] = "555-1234";
target["revenue"] = new Money(100000);
target["ownerid"] = new EntityReference("systemuse
r", userId);
```

Working with OptionSets:

```
target["statecode"] = new OptionSetValue(0); // Ac
tive
int statusCode = target.GetAttributeValue<OptionSe
tValue>("statuscode").Value;
```

8. Error Handling

Throw business error:

```
throw new InvalidPluginExecutionException("Custom
error message");
```

Try-catch pattern:

```
try
{
    // Plugin logic
}
catch (Exception ex)
{
    tracingService.Trace("Error: " + ex.Message);
    throw new InvalidPluginExecutionException(ex.M
essage);
}
```

9. Registration Information

Execution Stages:

`10` → Pre-validation

`20` → Pre-operation

`40` → Post-operation

Common Messages:

`Create` → record creation

`Update` → record update

`Delete` → record deletion

`Retrieve` → record retrieval

`Associate` → relationship creation

`Disassociate` → relationship removal

Execution Mode:

`0` → Synchronous

`1` → Asynchronous

Deployment:

`0` → Server

`1` → Client (deprecated)

`2` → Both (deprecated)

10. Complete Plugin Example

Account Update Plugin:

```
public class AccountUpdatePlugin : IPlugin
{
    public void Execute(IServiceProvider servicePr
ovider)
    {
        var context = (IPluginExecutionContext)ser
viceProvider.GetService(typeof(IPluginExecutionCon
text));
        var serviceFactory = (IOrganizationService
Factory)serviceProvider.GetService(typeof(IOrganiz
ationServiceFactory));
        var tracingService = (ITracingService)serv
iceProvider.GetService(typeof(ITracingService));

        if (context.InputParameters.Contains("Targ
et") && context.InputParameters["Target"] is Entit
y)
        {
            Entity target = (Entity)context.InputP
arameters["Target"];

            if (target.Contains("name"))
            {
                target["description"] = "Updated:
" + DateTime.Now.ToString();
            }
        }
    }
}
```

11. Best Practices

`Check depth` → prevent infinite loops
`Validate input` → check message, entity, stage
`Use tracing` → log important information
`Handle exceptions` → provide meaningful error messages
`Minimize queries` → use pre/post images when possible
`Use early bound` → generate entity classes for IntelliSense
`Stateless design` → don't use static variables

12. Common Data Types

`EntityReference` → lookup fields
`OptionSetValue` → picklist fields
`Money` → currency fields
`DateTime` → date/time fields
`Guid` → unique identifiers
`EntityCollection` → multiple records

```
// EntityReference example
var owner = new EntityReference("systemuser", user
Id);

// OptionSetValue example
var status = new OptionSetValue(1);

// Money example
var revenue = new Money(150000.00m);
```

13. Debugging Tips

`Plugin Registration Tool` → download from NuGet
`Tracing Service` → use for logging
`Plugin Profiler` → replay plugin execution locally
`Exception details` → check system jobs for async plugins

```
tracingService.Trace("Plugin started");
tracingService.Trace("Target entity: " + target.Lo
gicalName);
tracingService.Trace("Message: " + context.Message
Name);
```