**Plug-In**

**1. What is 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.

**2. When will you go for the plug in?**

* You need to manipulate data before is saved.
* You need to make validations before submitting the operation.
* You want to be able to cancel an operation based on your validations.
* Immediate response to the user is needed.
* You need retrieve values and/or take actions after operation has been completed (i.e. Getting and auto generated id).

**3. How the plug in run?**

The plug-in code runs in CRM server(s), either synchronously or asynchronously.

**4. What are the limitations of plugins?**

* In fact, plugin is flexible way to extend custom business logic. According to me the limitation of plugin is our experience of design and coding. For example, **we should improve performance of plugin is less than 2 minutes in Sanbox mode.**
* one of the limitations of running plugins in sandbox mode for crm online is you do not have **access to local services and folder structures** in the network.

**5. How many ways we can create the plugins?**

We can create plug in two ways

* By using Visual studio
* Developer tool kit

**6. How to create a plug in visual studio?**

First we need to download the SDK form the below link.

<https://www.microsoft.com/en-us/download/details.aspx?id=50032>

* Go to visual studio
* Click on the new
* Select the file
* Click on new 🡪click on project
* Click on the visual C#
* And click on the class library
* Click on browse and select the path for the your making DLL(Microsoft.Xrm.Sdk)
* If solution explore is not appearing on the window then go to view and click on the solution explorer then we will see the solution explorer.
* We need to add the system DLL that is Microsoft.Xrm.Serialization.
* After that click on the solution and click on the properties and add the strong key
* Now we need to add the DLL file. To add the DLL file right click on the references and click on the browse select the SDK🡪Bin🡪and select the"microsoft.xrm.sdk.dll" file and click on ok
* After completion of our script we need to build the code.
* After build succeeded the “DLL” will created on your respective plugin development folder.
* Finally we need to register the plug in.

**7. How to register a plug in?**

***Connecting the Server***

* **Step 1** − Run the **PluginRegistration.exe** from the location specified earlier. Click the Create New Connection button.
* **Step 2** − In the Login window, choose Office 365 since we are using the online version of CRM. Enter your credentials and click Login.
* **Step 3** − The tool will open and look like the following screenshot.

***Registering the Assembly***

* **Step 1** − Go to Register → Register New Assembly.
* **Step 2** − This will open the Register New Assembly window. Click the Navigate icon and locate the Plugin DLL that you created in the last chapter.
* **Step 3** − After navigating the DLL, click Load Assembly. This will populate the SamplePlugins assembly and all its plugin classes. You can see the PostCreateContact plugin class highlighted below. If your plugin assembly had 3 plugin classes, it would have shown three plugins listed there.
* **Step 4** − Select Isolation Mode as Sandbox, Location as Database and click Register Selected Plugins. It will show you a success message, if the registration is successful.

***Registering the Step for Plugin***

* Now we will be registering the specific steps on which the individual plugins will be called.
* **Step 1** − Select the PostCreateContact plugin.
* **Step 2** − Click Register → Register New Step.
* **Step 3** − We will be registering this plugin on the creation of the Contact entity, on postoperation stage and in the synchronous mode.
* **Message** − Create
* **Primary Entity** − Contact
* **Event Pipeline Stage of Execution** − Post-operation
* **Execution Mode** − Synchronous
* Keep the rest of the options by default and click Register New Step.
* You can see a new step added to the plugin.
* Now we will go to CRM and test if our plugin is working correctly. Note that these test steps are specific to our example plugin.

**8. How to debug a plugin?**

* **Step 1**: Download and extract Dynamics 365 SDK  in any folder.
* **Step 2**: Now navigate to SDK\Tools\PluginRegistration
* Here you’ll see pluginRegistration.exe file.
* Double click on it and create a new connection.by clicking “Create New Connection”
* **Step 3**: Click on login. It’ll ask for credentials.
* Provide your dynamics 365 username and password and click sign in.
* **Step 4**: Now it’ll display the available instances of your organization.
* Select the one in which your plugin exist you want to debug.
* It’ll display the list of existing assembly deployed in that  instance.
* **Step 5:** Click on install profiler button on top ribbon of Plugin Registration tool.
* Once it’s done, it’ll look like this:
* Also a solution is created in Dynamics CRM instance as shown below:
* **Step 6:** Now select the step you want to debug in plugin registration tool and click Start Profiling.
* Click ok in next window:
* As you click OK,the plugin step is updated as below:
* **Step 7:** Now perform the operation on which the above step has been registered.
* It’ll generate an exception and a log file which you need to download and save.
* It’ll be used in step 9.
* **Step 8**:  Now open the plugin solution in Visual studio and Add the breakpoint  in plugin code from where you would like to debug.
* Now click on debug and then click attach to process.
* Below window will pop up where you need to select PluginRegistration.exe adn click on attach:
* **Step 9**: Now click on debug in plugin registration tool as shown below:
* Now below window will open where you need to select the log file downloaded in step 7 in Profile, specify plugin dll assembly location in Assembly location and select the step in Plugin as shown in below image and click “Start Execution“:
* **Step 10**: As soon as you click on Start Execution, the debugger stops at the breakpoint inserted in code in step 8 as shown in below screenshot:
* Now you can debug the code line by line further by pressing F10.

**9. What are the parameters of execute method in Plugins?**

**Parameters of Execution method**

* [IService\_Provider](https://msdn.microsoft.com/library/system.iserviceprovider.aspx) : A container for service objects.
* ([IPluginExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IPluginExecutionContext.htm)) :Contains references to the plug-in execution context.
* ([ITracingService](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_ITracingService.htm)) : A reference to a tracing service implementation can be obtained from the service provider passed to plug-in’s [Execute](https://msdn.microsoft.com/en-in/library/microsoft.xrm.sdk.iplugin.execute.aspx) method.
* The **ITracingService** interface provides a way to log plug-in run-time information. This method of logging information is especially useful for sandboxed plug-ins registered with Microsoft Dynamics 365 (online) that cannot otherwise be debugged using a debugger.
* ([IOrganizationServiceFactory](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IOrganizationServiceFactory.htm)) : In Microsoft Dynamics 365, the primary web service that accesses data and metadata for your organization is [IOrganizationService](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.aspx). This web service contains the methods that you use to write code that uses all the data and metadata in Microsoft Dynamics 365.

**10. What is the entity reference?**

* [**CrmEntityReference()**](https://msdn.microsoft.com/en-us/library/gg678477(v=crm.7).aspx) **:** Initializes an instance of the CrmEntityReference class.
* [**CrmEntityReferenceGuid) (String,**](https://msdn.microsoft.com/en-us/library/gg695465(v=crm.7).aspx)[**Guid)**](https://msdn.microsoft.com/en-us/library/gg695465(v=crm.7).aspx)**:** Initializes an instance of the CrmEntityReference class.

**11. How do you handle the errors in plugin?**

**Or**

**What is exception handling in plug in?**

By using try and catch.

# 12. What are the Organization service methods?

## Create

Use the [IOrganizationService](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.aspx).[Create](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.create.aspx) method to create an instance (record) of any entity that supports the **Create** message, including custom entities.

## Retrieve

Use the [IOrganizationService](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.aspx).[Retrieve](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.retrieve.aspx) method to retrieve an instance (record) of an entity.

## Retrieve Multiple

Use the [IOrganizationService](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.aspx).[RetrieveMultiple](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.retrievemultiple.aspx) method to retrieve a collection records. The query can be specified using a query expression or Fetch XML query.

## Update

Use the [IOrganizationService](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.aspx).[Update](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.update.aspx) method to update an existing record.

## Delete

Use the [IOrganizationService](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.aspx).[Delete](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.delete.aspx) method to delete an existing record.

## Associate

Use the [IOrganizationService](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.aspx).[Associate](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.associate.aspx) method to create a link between two records that participate in a relationship.

## Disassociate

Use the [IOrganizationService](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.aspx).[Disassociate](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.disassociate.aspx) method to delete the link between two records.

## Execute

Use the [IOrganizationService](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.aspx).[Execute](https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.execute.aspx) method to execute a message. This includes common processing like create and delete of data records and metadata, or it can be specialized processing such as import or detect duplicates.

**13. How do you get the entity information in plugin?**

**Or**

**What is Iplugin Execution?**

Iplugin execution context obtain the entity information and plugin execution pipeline stages in to our plug in logic via Iservice provider.

Developers can use the IPluginExecutionContext service object to obtain contextual information from the CRM platform which is passed to the plugin at run-time.

The IPluginExecutionContext object includes the following GUID properties:

• PrimaryEntityName  
• PrimaryEntityId

The PrimaryEntityName property returns the schema name of the entity where the triggering action was executed. For example, if a user updates an Account which causes a plugin to fire, the PrimaryEntityNameproperty would return “account”. This property is useful when writing plugins that need to execute on multiple entities.

**14. What is the starting point of the plug in execution?**

Execution method is the starting point of the plug in execution.

# 15. What are the IPluginExecutionContext Properties?

|  |  |
| --- | --- |
| **Name** | **Description** |
| [BusinessUnitId](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_BusinessUnitId.htm) | Gets the GUIDGUID of the business unit that the user making the request, also known as the calling user, belongs to. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [CorrelationId](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_CorrelationId.htm) | Gets the GUID for tracking plug-in or custom workflow activity execution.  (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [Depth](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_Depth.htm) | Gets the current depth of execution in the call stack. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [InitiatingUserId](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_InitiatingUserId.htm) | Gets the GUID of the system user account under which the current pipeline is executing. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [InputParameters](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_InputParameters.htm) | Gets the parameters of the request message that triggered the event that caused the plug-in to execute. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [IsExecutingOffline](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_IsExecutingOffline.htm) | Gets whether the plug-in is executing from the Microsoft Dynamics CRM for Microsoft Office Outlook with Offline Access client while it is offline.  (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [IsInTransaction](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_IsInTransaction.htm) | Gets a value indicating if the plug-in is executing within the database transaction. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [IsOfflinePlayback](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_IsOfflinePlayback.htm) | Gets a value indicating if the plug-in is executing as a result of the Microsoft Dynamics CRM for Microsoft Office Outlook with Offline Access client transitioning from offline to online and synchronizing with the Microsoft Dynamics CRM server. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [IsolationMode](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_IsolationMode.htm) | Gets a value indicating if the plug-in is executing in the sandbox. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [MessageName](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_MessageName.htm) | Gets the name of the Web service message that is being processed by the event execution pipeline. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [Mode](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_Mode.htm) | Gets the mode of plug-in execution. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [OperationCreatedOn](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_OperationCreatedOn.htm) | Gets the date and time that the related **System Job** was created. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [OperationId](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_OperationId.htm) | Gets the GUID of the related **System Job**. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [OrganizationId](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_OrganizationId.htm) | Gets the GUID of the organization that the entity belongs to and the plug-in executes under. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [OrganizationName](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_OrganizationName.htm) | Gets the unique name of the organization that the entity currently being processed belongs to and the plug-in executes under. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [OutputParameters](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_OutputParameters.htm) | Gets the parameters of the response message after the core platform operation has completed. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [OwningExtension](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_OwningExtension.htm) | Gets a reference to the related **SdkMessageProcessingingStep** or **ServiceEndpoint**. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [ParentContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IPluginExecutionContext_ParentContext.htm) | Gets the execution context from the parent pipeline operation. |
| [PostEntityImages](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_PostEntityImages.htm) | Gets the properties of the primary entity after the core platform operation has been completed. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [PreEntityImages](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_PreEntityImages.htm) | Gets the properties of the primary entity before the core platform operation has begins. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [PrimaryEntityId](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_PrimaryEntityId.htm) | Gets the GUID of the primary entity for which the pipeline is processing events. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [PrimaryEntityName](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_PrimaryEntityName.htm) | Gets the name of the primary entity for which the pipeline is processing events. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [RequestId](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_RequestId.htm) | Gets the GUID of the request being processed by the event execution pipeline. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [SecondaryEntityName](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_SecondaryEntityName.htm) | Gets the name of the secondary entity that has a relationship with the primary entity. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [SharedVariables](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_SharedVariables.htm) | Gets the custom properties that are shared between plug-ins. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |
| [Stage](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IPluginExecutionContext_Stage.htm) | Gets the stage in the execution pipeline that a synchronous plug-in is registered for. |
| [UserId](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/P_Microsoft_Xrm_Sdk_IExecutionContext_UserId.htm) | Gets the GUID of the system user for whom the plug-in invokes web service methods on behalf of. (inherited from [IExecutionContext](ms-its:C:\Users\home\Desktop\SDKKK\New%20folder\MicrosoftDynamicsCRM2015SDK\SDK\CrmSdk2015.chm::/html/T_Microsoft_Xrm_Sdk_IExecutionContext.htm)) |

**16. What are the Isolation modes?**

* Sandbox
* None

Whenever you register a plug in for online deployment we need to choose Sandbox.

For on premise we can choose the either sandbox or none.

**17. How many ways we can store a plug in?**

* Database = Plugin.dll uploaded into the database
* Disk = The Plugin.dll is saved on the CRM Servers hard disk
* GAC = The plugin.dll is loaded into the GAC on the CRM Server.

For an on-premises deployment, plug-ins that are not registered in the sandbox can be stored in the Microsoft Dynamics CRM server's database or the *on-disk* file system. We strongly recommend that you store your production-ready plug-ins in the Microsoft Dynamics CRM database, instead of on-disk. Plug-ins stored in the database are automatically distributed across multiple Microsoft Dynamics CRM servers in a data center cluster. On-disk storage of plug-ins is useful for debugging plug-ins using Microsoft Visual Studio but is mostly provided for backward compatibility with callouts. You can debug a plug-in that is stored in the database. For more information, see [Debug a Plug-In](http://msdn.microsoft.com/en-us/library/gg328574.aspx).

Plug-ins registered in the sandbox must be stored in the database regardless of the Microsoft Dynamics CRM deployment (on-premises, IFD/SPLA, or Online).

**Note:** Online plug in always be Database.

**18. I have 3 plug ins with create message on same entity but stages are different. I have created 3 messages and registered on pre validation, Pre operation and post operation. These are all synchronous plugins.**

**Due to some reasons whichever the plug in I have configured on post operation gets failed.**

**What about the remaining two plug-in?**

Pre validation plug in will execute since it is not database transaction. This plug in will execute outside of the database operation.

**19. In the above scenario if you assume synchronous plug in will get failed in pre validation. What about the other two plugins?**

Two plugins will execute.

**Note:** Whenever plug in fails we can see logs in plug in trace logs.

**20. When do you go for the synchronous plug in and asynchronous plug in?**

Both Synchronous and Asynchronous plugins have the same 2 minutes time limit. If you have more depth of custom business logic. We will go with the asynchronous plug in.

**21. What is the syntax of pre image and post image?**

***Suppose you registered the Plugin and added a Image with name “PreImage ”***

***Entity preMessageImage;***

if (context.PreEntityImages.Contains(“PreImage”) && context.PreEntityImages["PreImage"] is Entity)

{

preMessageImage = (Entity)context.PreEntityImages["PreImage"];

accountnumber = (String)preMessageImage.Attributes["accountnumber"];

}

***Suppose you registered the Plugin and added a Image with name “PostImage ”***

***Entity postMessageImage;***

if (context.PostEntityImages.Contains(“PostImage”) && context.PostEntityImages["PostImage"] is Entity)

{

postMessageImage = (Entity)context.PostEntityImages["PostImage"];

accountnumber = (String)postMessageImage.Attributes["accountnumber"];

}

**22. What are the plugin pipeline stages?**

|  |  |  |  |
| --- | --- | --- | --- |
| Stage# | Stage | Stage Name | Description |
| 10 | Pre Event | Pre Validation | - It execute before the main system operation. - It may execute outside the database transaction. - 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. |
| 20 | Pre Event | Pre Operation | - It execute before the main system operation. - It could be executed in the database transaction. |
| 30 | Platform Core Operation | Main Operation | - This stage used for internal use only. - It handles main operation such as create, update, delete etc. - No custom plugins can be registered in this stage. |
| 40 | Post Event | Post Operation | - This Stage execute after the main operation. - It could be executed in the database transaction |
| 50 | Post Event | Post Operation [Deprecated] | - This Stage execute after the main operation. - This stage only supports Microsoft Dynamics CRM 4.0 plugins. |

The Microsoft Dynamics CRM 2011 execution pipeline is organization specific. Server can host multiple organizations. There will be virtual pipeline for every organization. This means a plugin must be registered with each organization execution where it has to execute.

#### 23. What is Impersonation?

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. |

**24. What do you mean by run in user context?**

Plugins are usually set to run as the calling user but sometimes you need to run a plugin with System User privileges.

It’s usually the right choice to use the Calling User because any updates, retrieves or any interaction with CRM data will be done using the calling users identity and privileges.

* The IOrganizationService can only retrieve data the user can retrieve
* When IOrganizationService updates, creates records they are stamped with the calling user
* The plugin cannot do anything the calling user cannot do

**25. What are the permissions or privileges are required to install a plug in?**

The user must have the system administrator role and he must be a member of the administrative groups in active directory.

**26. My plug in should have to trigger each and every user irrespective of the privileges or roles then how do you register a plug in?**

I will register this plug in the system administrator role.

**27. What do you mean by the filtering attributes?**

This is very handy parameter of step registration process. It is equivalent to “IsDirty” function in JavaScript. If you use “Filtering Attributes” on post update events, you don’t need to compare the value of fields using PreEntityImage and PostEntityImage.

The other biggest advantage is that your plugin does not execute on every update event.  
Here is some code. This code creates a note for a contact when **parentcustomerid** field is updated.

**28. When do you go with the execution order?**

Execution order in plugin 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.

**29. I have 3 plugins I have to execute one after another there is a dependency. Then what is your approach?**

In plug in registration tool there is property called execution order there we can give ranking.

**30.When plugin will go infinite loop? How do you recover?**

An infinite plugin loop example is: You have a post-update plugin on an entity 'A' that updates something in entity 'B' and entity 'B' has a post update plugin that updates plugin 'A'.

Or

You have a post-update plugin on an entity 'A' that updates a field in the same entity 'A', when entity 'A' will be updated, it will result in another update of entity 'A' and this will go on and will result in infinite loop.

Or

When I’m updating the account record using plug in I’m updating the primary contact.

Whenever the primary contact is updated I’m updating the company name through another plug in and this will go on and will result in infinite loop.

**How do you recover :**

There is a property called Depth for Context object. You can use this property to check if you are heading to an infinite loop. To avoid this situation we should have to approach ***IpluginExecutioncontext.depth*** property.

**31.What are the images available in the plugins? When will you go with images?**

There are two types of images

1. Pre Image
2. Post Image

* 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.

**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 | No | Yes |
| Update | PRE | Yes | No |
| Update | POST | Yes | Yes |
| Delete | PRE | Yes | No |
| Delete | POST | Yes | No |

**32. How to Retrieve data from more than one entity using plugin,crm 2011?**

There a couple of a different ways to do this, FetchXml, QueryExpression and Linq or OData

In Microsoft Dynamics CRM 2011, you can use the OrganizationServiceProxy class to access the Web services. Alternatively, you can use the OrganizationServiceContext generated by the code generation tool to gain access to additional functionality.

**33. How to Retrieves the user information for the logged on user?**

OrganizationServiceContextExtensions.WhoAmI Method

**WhoAmIRequest Class:**

Contains the data that is needed to retrieve the system user ID for the currently logged on user or the user under whose context the code is running.

You can call the WhoAmIRequest message to find out the organization for the currently logged on or impersonated user.

**34.What is Impersonate a User?**

To impersonate a user, set the **CallerId** property on an instance of **OrganizationServiceProxy** before calling the service’s Web methods.

**35.What is Support Offline Execution?**

You can register plug-ins to execute in online mode, offline mode, or both. Offline mode is only supported on Microsoft Dynamics CRM for Microsoft Office Outlook with Offline Access. Your plug-in code can check whether it is executing in offline mode by checking the [IsExecutingOffline](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iexecutioncontext.isexecutingoffline.aspx) property.

When you design a plug-in that will be registered for both online and offline execution, remember that the plug-in can execute twice. The first time is while Microsoft Dynamics CRM for Microsoft Office Outlook with Offline Access is offline. The plug-in executes again when CRM for Outlook goes online and synchronization between CRM for Outlook and the Microsoft Dynamics CRM server occurs. You can check the [IsOfflinePlayback](http://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iexecutioncontext.isofflineplayback.aspx) property to determine if the plug-in is executing because of this synchronization.

**36.What are the Database Transactions in Plug-Ins?**

Plug-ins may or may not execute within the database transaction.

You can check if the plug-in is executing in-transaction by reading the ‘IsInTransaction‘ property of IPluginExecutionContext.

Stages 20 and 40 are part of the database transaction while stage 10 and 50 may be part of the transaction

If plugin throws an exception, every action done during the transaction will be rollback.

**37. What will happen after two minutes time limit in plug in?**

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 Timeout exception is thrown

If a plug-in needs more processing time than the 2 minute time limit, consider using a workflow or other background process

**38. How to get the Entity Id when the PlugIn is in PreCreate and PreUpdate Stage?**

In Crm 2011 you can read this from the context.PrimaryEntityId property, rather than needing to go through the Target InputParameter.

**39. What are deployment modes MS Crm?**

For on-premises or Internet-facing Microsoft Dynamics CRM installations, when you deploy plug-ins from another computer to the Microsoft Dynamics CRM server disk (on-disk deployment), the plug-in assembly must be manually copied to the server before registration. The assembly must be deployed to the <installdir>\Program Files\Microsoft CRM\server\bin\assembly folder on each server where the plug-in is to execute.

Plug-in registration should be done after the assembly has been copied to the …\bin\assembly folder on the server to prevent the situation where a system user causes an event in Microsoft Dynamics CRM to be raised but the registered plug-in assembly does not yet exist on the server. For server database deployment, the plug-in assembly is automatically copied during plug-in registration so that the earlier situation is not an issue.

**Note:**

Depending on your plug-in’s design, your plug-ins may require other referenced assemblies to run. Regardless of whether you deploy your plug-in to the database or disk, if your plug-in requires other assemblies to run, you must put copies of these assemblies in the global assembly cache on each server where the plug-in is to execute. This does not apply to a Microsoft Dynamics CRM Online server because you do not have access to the global assembly cache on that server.

**40. How to move a plug-in from a development environment to a staging or production server?**

1. On the development computer, build the plug-in code. Do not include debug information. Optimize the plug-in for performance.
2. Register the plug-in in the Microsoft Dynamics CRM server database.
3. Using the Microsoft Dynamics CRM web application, create a solution or use an existing one, and add the plug-in to that solution.
4. After you have added any other desired components to the solution, export the solution.
5. Import the solution on to the staging or production server.

**41. What are the Assembly Versioning and Solutions?**

Plug-in assemblies can be versioned using a number format of major.minor.build.revision defined in the Assembly.info file of the Microsoft Visual Studio 2010 project. Depending on what part of the assembly version number is changed in a newer solution, the following behavior applies when an existing solution is updated through import.

* The build or revision assembly version number is changed.
* This is considered an in-place upgrade. The older version of the assembly is removed when the solution containing the updated assembly is imported. Any pre-existing steps from the older solution are automatically changed to refer to the newer version of the assembly.
* The major or minor assembly version number, except for the build or revision number is changed.  
    
  When an updated solution containing the revised assembly is imported, the assembly is considered a completely different assembly than the previous version of that assembly in the existing solution. Plug-in registration steps in the existing solution will continue to refer to the previous version of the assembly. If you want existing plug-in registration steps for the previous assembly to point to the revised assembly, you will need to use the Plug-in Registration tool to manually change the step configuration to refer to the revised assembly type. This should be done before exporting the updated assembly into a solution for later import.

**42. What are the Security Restrictions?**

There is a security restriction that enables only privileged users to register plug-ins. For plug-ins that are not registered in isolation, the system user account under which the plug-in is being registered must exist in the **Deployment Administrators** group of Deployment Manager. Only the System Administrator user account or any user account included in the **Deployment Administrators** group can run Deployment Manager.

|  |
| --- |
| **ImportantImportant** |
| For non-isolated plug-ins, failure to include the registering user account in the **Deployment Administrators** group results in an exception being thrown during plug-in registration. The exception description states "Not have enough privilege to complete Create operation for an SDK entity." |

The system user account under which the plug-in is being registered must have the following organization-wide security privileges:

* prvCreatePluginAssembly
* prvCreatePluginType
* prvCreateSdkMessageProcessingStep
* prvCreateSdkMessageProcessingStepImage
* prvCreateSdkMessageProcessingStepSecureConfig

For plug-ins registered in the sandbox (isolation mode), the system user account under which the plug-in is being registered must have the System Administrator role. Membership in the **Deployment Administrators** group is not required.

. Multiple plug-ins that are registered in each stage can be further be ordered (ranked) within that stage during plug-in registration.

**43.** [**The server was unable to process the request due to an internal error – Plugin registration tool**](https://rajeevpentyala.com/2013/06/13/the-server-was-unable-to-process-the-request-due-to-an-internal-error-plugin-registration-tool/)**?**

<http://rajeevpentyala.wordpress.com/category/crm/crm-2011/plug-ins/> ---- very very very imp blog

**44. How Many userid’s available in Plugin context?**

Within the Plugin Context, there are two user ids,

**InitiatingUserId:** Which returns the id of the user who actually fired the plugin.

**User id:** which returns the user Id of the user the plugin is actually running under. (this is the user specified when registering the plugin, or the Calling User, if it's registered to run as the calling user).

**45. What are the Sandbox assemblies?**

1. Sandboxed assemblies can only be registered in Database.
2. Only sandboxed assemblies can be registered in CRM Online.

**46. What is the IPluginExecutionContext.ParentContext Property?**

Gets the execution context from the parent pipeline operation.

**47. What is the ServiceProxy.IsAuthenticated Property?**

Gets a value indicating if the WCF channel has been authenticated.

**48. How to add or remove members from the Team via plugin?**

I also faced similar kind of issue when i add or remove members from the Team via plugin. So, I have tried the plugin message -  Add Members / RemoveMembers. but it didnt work.  But if I use Assosiate/Disassociate message then the plugin is triggering.

**49. How to avoid infinite loop in plugin?**

[CRM 2011 Plugins – Avoiding Infinite Loops](http://roscode.wordpress.com/2013/04/24/crm-2011-plugins-avoiding-infinite-loops/)

When developing plugins for Microsoft Dynamics CRM 2011, there are scenarios in which the execution of your custom code can result in an infinite loop. This leads to unnecessary execution of plugins and can potentially impact the performance of CRM, as well as the user experience. In this post, we will take a look at the Depth property of the IPluginExecutionContext service object and see how it can be used to eliminate the risk of infinite loops.

Before we see how to use the Depth property, let’s take a look at a scenario. Assume your CRM implementation consists of the Contact entity and a custom entity called Member. The requirement for the system is to keep data on the Contact and Member entities synchronised, so that the following business rules apply:

1. When a Member is created, a corresponding Contact record is created (if one does not already exist)
2. When certain details on the Member is updated, details on the corresponding Contact record must also be updated
3. When certain details on the Contact record is updated, details on the corresponding Member record must also be updated
4. You can see that this type of business logic will require a plugin to be registered on create of Member, and on update of both Contact and Member. You will also notice that updating a Member triggers an update of the Contact, which again triggers an update of the Member and so on.
5. Firstly, let’s see what happens if we deploy such a plugin without any consideration for handling infinite loops. I have updated the “Full Name” field on the Member, which should try to update the “Full Name” field on the Contact.

The CRM platform throws an error stating that it has identified an infinite loop. This usually happens after the number of iterations reaches a maximum of 8. We can fix this by adding a depth check at the start of our plugin code, just after we initialize each of the service objects.

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

**IOrganizationServiceFactory factory = (IOrganizationServiceFactory)serviceProvider.GetService(typeof(IOrganizationServiceFactory));**

**\_sdk = factory.CreateOrganizationService(context.UserId);**

**ITracingService tracer = (ITracingService)serviceProvider.GetService(typeof(ITracingService));**

**if (context.Depth > 1) { return; }**

Notice that last line which checks if the plugin has run more than once, and if so, executes a return statement to cancel out of the plugin. Now when we run the plugin, we shouldn’t run into an infinite loop.

You must be careful when using the Depth property as there are more complex scenarios that you may run into. For example, a data import may trigger a workflow to update the “Full Name” of a Member, which in turn will execute our plugin. In this case, the Depth will be 3 at the first execution of the plugin. If you left the code snippet above unchanged, the plugin logic will never get executed.

**50. How to obtain entity record information?**

[CRM 2011 Plugins – Obtaining Entity Record Information](http://roscode.wordpress.com/2013/04/22/crm-2011-plugins-obtaining-entity-record-information/)

Developers can use the IPluginExecutionContext service object to obtain contextual information from the CRM platform which is passed to the plugin at run-time. It includes details about the user who triggered the plugin event as well as transactional information handled at the platform layer. In this post, we will see which properties are available to obtain information about the record which triggered a plugin.

The IPluginExecutionContext object includes the following GUID properties:

* PrimaryEntityName
* PrimaryEntityId

The PrimaryEntityName property returns the schema name of the entity where the triggering action was executed. For example, if a user updates an Account which causes a plugin to fire, the PrimaryEntityName property would return “account”. This property is useful when writing plugins that need to execute on multiple entities.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | if (context.PrimaryEntityName.Equals("contact", StringComparison.InvariantCultureIgnoreCase))  {  // do something for contacts  }  else if (context.PrimaryEntityName.Equals("account", StringComparison.InvariantCultureIgnoreCase))  {  // do something else for accounts  } |

* There is also a SecondaryEntityName property which is useful when defining plugin steps on the SetRelated message.
* The PrimaryEntityId property returns the GUID of the record where the triggering action was executed. This property is useful when you need to retrieve the record from the CRM database.
* Entity contact = \_sdk.Retrieve("contact", context.PrimaryEntityId, new ColumnSet("firstname", "lastname", "emailaddress1"));
* As I mentioned above, a plugin could be triggered to run on records for multiple entities. You wouldn’t normally hardcode the entity schema name in Retrieve operations so it is best to use the two properties together.

**For example:**

|  |  |
| --- | --- |
| 1 | Entityrecord =\_sdk.Retrieve(context.PrimaryEntityName, context.PrimaryEntityId, new ColumnSet(true |

**51. How to obtain user information?**

[CRM 2011 Plug-in – Obtaining User Information](http://roscode.wordpress.com/2013/04/14/crm-2011-plugins-obtaining-user-information/)

Developers can use the IPluginExecutionContext service object to obtain contextual information from the CRM platform which is passed to the plugin at run-time. It includes details about the user who triggered the plugin event as well as transactional information handled at the platform layer. In this post, we will see which properties are available to obtain information about a user.

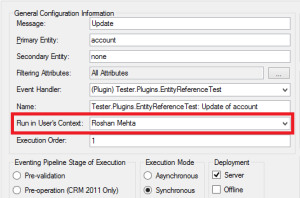
The IPluginExecutionContext object includes the following GUID properties:

* BusinessUnitId
* InitiatingUserId
* UserId

The **BusinessUnitId** property allows developers to obtain the GUID of the Business Unit the calling user belongs to. For example, user Frosty Stevens belongs to the “Sales” Business Unit. When he performs an action that triggers a plugin, the ID of the “Sales” Business Unit will be returned my invoking this property.

The **InitiatingUserId** is the GUID of the user who triggered the execution of the plugin. For example, user Frosty Stevens updates an Account record. When the plugin code is executed, the InitiatingUserId property will return the GUID of Frosty Stevens’ user record.

The **UserId** property is not to be mistaken with the InitiatingUserId property. This property returns the GUID of the user of which custom plugin code is invoked on behalf on. This user is set when defining plugin steps via the Plugin Registration Tool.

[](http://roscode.files.wordpress.com/2013/04/capture.png)

There are many more useful properties available via the IPluginExecutionContext. In my next post, we will take a look at how we can obtain information about the organization from which a plugin was executed.

**52. What is** [**CRM 2011 Plugins – IServiceProvider**](http://roscode.wordpress.com/2013/04/07/crm-2011-plugins-iserviceprovider/)**?**

When a particular event occurs in Microsoft Dynamics CRM, such as “create of a contact” or “update of an account”, the Execute method is invoked for any plugins registered on the event. This method includes a single serviceProvider parameter which provides useful service information about the execution of the plugin. In this post, we will take a look at the information that is made available by the serviceProvider parameter.

The types of service objects available include the following:

* IPluginExecutionContext
* IOrganizationServiceFactory
* ITracingService

The **IPluginExecutionContext** service object is the most useful of the three and provides contextual information to the plugin at run-time. It includes details about the user who triggered the plugin event as well as transactional information handled at the platform layer. The following code can be used to obtain the execution context from the service provider:

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

The IOrganizationServiceFactory service object allows us to create an instance of the organization service which can be used to define and execute various platform requests.

The following code can be used to achieve this:

|  |  |
| --- | --- |
| 1  2 | IOrganizationServiceFactory factory = (IOrganizationServiceFactory)serviceProvider.GetService(typeof(IOrganizationServiceFactory));  IOrganizationService sdk = factory.CreateOrganizationService(context.UserId); |

In the example above, we pass in the GUID of the user who triggered the plugin event, which is obtained from the IPluginExecutionContext. Alternatively, we can pass in the GUID of another CRM user, or pass in a null value to execute the plugin logic under the system context.

Lastly, the ITracingService allows us to trace the plugin execution flow and any variables for debugging purposes. For more information, please read my post on [Debugging Dynamics CRM 2011 Plugins](http://www.magnetismsolutions.com/blog/roshan/roshan-mehtas-blog/2011/11/20/Debugging_Dynamics_CRM_2011_Plugins.aspx).

There you have it, an introduction to the various service objects that we will be working with in our journey to create plugins for Microsoft Dynamics CRM. In my next post, we will take a closer look at the IPluginExecutionContext.

**53.** [**Crm 2011 how to get the stepid in which the plugin is executing**](http://stackoverflow.com/questions/14043482/crm-2011-how-to-get-the-stepid-in-which-the-plugin-is-executing)**?**

In crm 2011, inside the Execute method of a plugin, how can I know the id of the registered step that is executing? For instance, I have two steps for the pre create of an account. The execute method will run two times one for each step. I need to know in the execute method the stepid of the step that is actually running. I can't find it in the context.

You can get the name of the message from the context. Usually, I do something similar to this.

public void Execute(IServiceProvider serviceProvider)

{

IPlugingExecutionContext context

= (IPlugingExecutionContext)serviceProvider

.getService(typeof(IPlugingExecutionContext));

switch(context.MessageName)

{

case "Create" ExecuteCreate(); break;

case "Retrieve" ExecuteCreate(); break;

case "Update" ExecuteCreate(); break;

case "Delete" ExecuteCreate(); break;

default ExecuteFunctionality(Context.MessageName);

}

}

**54. What are the plugin types?**

Synchronous Plug-in: synchronous execution is executed immediately. Synchronous plug-ins that are registered for the event are executed in a well-defined order.

Asynchronous Plug-in: 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 plug-in registered in the sandbox. If the execution of your plug-in logic exceeds the time limit, a [System.TimeoutException](http://msdn.microsoft.com/en-us/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. |

You can check if the plug-in is executing in-transaction by reading the **IsInTransaction** property inherited by **IPluginExecutionContext** 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 and 50 may be part of the transaction.

**55. What is Sand box?**

SandBox: the sandbox is the recommended execution environment for plug-ins as it is more secure, supports run-time monitoring and statistics reporting, and is supported on all Microsoft Dynamics CRM deployments. In addition, Microsoft Dynamics CRM Online only supports execution of custom workflow activities if they are registered in the sandbox.

**Note:** In plug-in code, you can send any message to the web services except those messages that create or update metadata.

The term *offline* applies to the Microsoft Dynamics CRM for Microsoft Office Outlook with Offline Access. The term *client* can apply to either Microsoft Dynamics CRM for Outlook Online or Microsoft Dynamics CRM for Microsoft Office Outlook with Offline Access.

Whether a message is available online or offline can be determined programmatically by inspecting the **SdkMessage.Availability** attribute.

Plug-ins are custom classes that implement the **IPlugin** interface

You must add **Microsoft.Xrm.Sdk.dll** and **Microsoft.Crm.Sdk.Proxy.dll** assembly references to your project. These assemblies can be found in the SDK\Bin folder of the SDK

**56. What is Support Offline Execution in plug ins?**

You can register plug-ins to execute in online mode, offline mode, or both. Offline mode is only supported on Microsoft Dynamics CRM for Microsoft Office Outlook with Offline Access. Your plug-in code can check whether it is executing in offline mode by checking the **IsExecutingOffline** property.

When you design a plug-in that will be registered for both online and offline execution, remember that the plug-in can execute twice. The first time is while Microsoft Dynamics CRM for Microsoft Office Outlook with Offline Access is offline. The plug-in executes again when Microsoft Dynamics CRM for Outlook goes online and synchronization between Microsoft Dynamics CRM for Outlook and the Microsoft Dynamics CRM server occurs. You can check the **IsOfflinePlayback** property to determine if the plug-in is executing because of this synchronization.

**57. What is Access the Plug-in Execution Context?**

**IPluginExecutionContext** contains information that describes the run-time environment that the plug-in is executing in, 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** method.

|  |  |
| --- | --- |
| **C#** | copycodeCopy Code |
| // Obtain the execution context from the service provider.  IPluginExecutionContext context = (IPluginExecutionContext)  serviceProvider.GetService(typeof(IPluginExecutionContext)); | |

When a system event is fired for which a plug-in is registered, 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** 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**.

**58. What is 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.

|  |  |
| --- | --- |
| **C#** | copycodeCopy Code |
| // 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.

## What are the Input and Output Parameters?

The **InputParameters** property contains the data that is in the request message currently being processed by the event execution pipeline.

|  |  |
| --- | --- |
| **C#** | copycodeCopy Code |
| // 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**, so you do have to look at each individual request or response. For example, **DeleteRequest** does have a **Target** property but its type is **EntityReference**. The preceding code example would be changed as follows.

|  |  |
| --- | --- |
| **C#** | copycodeCopy Code |
| // 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** property contains the data that is in the response message, for example **CreateResponse**, currently being passed through the event execution pipeline. However, only synchronous post-event and asynchronous registered plug-ins have **OutputParameters** populated as the response is the result of the core platform operation.

**59. How to Handle Exceptions in Plug-Ins?**

For synchronous plug-ins, whether registered in the sandbox or not, the Microsoft Dynamics CRM platform handles exceptions passed back from a plug-in by displaying an error message in a dialog of the web application user interface. The exception message for asynchronous registered plug-ins is written to a System Job (**AsyncOperation**) record that can be viewed in the System Jobs area of the web application.

For synchronous plug-ins, you can optionally display a custom error message in the web application by having your plug-in throw an **InvalidPluginExecutionException** exception with the custom message string as the exception **Message** property value. If you throw **InvalidPluginExecutionException** and do not provide a custom message, a generic default message is displayed. It is recommended that plug-ins only pass an **InvalidPluginExecutionException** back to the platform.

If a synchronous plug-in returns an exception other than **InvalidPluginExecutionException** back to the platform, the error dialog is displayed to the user and the exception message ([System.Exception.Message](http://msdn.microsoft.com/en-us/library/system.exception.message.aspx)) with stack trace is also written to one of two places. For plug-ins not registered in the sandbox, the information is written to the Application event log on the server that runs the plug-in. The event log can be viewed by using the Event Viewer administrative tool. For plug-ins registered in the sandbox, the exception message and stack trace is written to the Microsoft Dynamics CRM platform trace.

**60. How to Pass Data Between Plug-in?**

The name of the parameter that is used for passing information between plug-ins is **SharedVariables**. This is a collection of key\value pairs. At run time, plug-ins can add, read, or modify properties in the **SharedVariables** collection. This provides a method of information communication among plug-ins.

What is Logging and Tracing in plug in?

An alternative method to debug a plug-in is to use tracing. Tracing assists developers in troubleshooting plug-ins by providing run-time plug-in information as an aid in diagnosing the cause of plug-in failure. Tracing is especially useful to debug Microsoft Dynamics CRM Online registered plug-ins as it is the only supported debugging method for that scenario.

The tracing discussed here is different from ASP.NET tracing. Tracing is implemented in the Microsoft Dynamics CRM SDK through the use of the tracing service **ITracingService**. Developers add **Trace** statements to their plug-in code, then build and deploy the plug-in.

**61. How to Analyze Plug-in Performance?**

The *Plug-in Profiler* is a tool that profiles the execution of plug-ins and custom workflow activities for an enhanced debugging experience in Microsoft Visual Studio

**62. How to get related entities ?**

// Get Related Entities

**EntityReferenceCollection relatedentities = (EntityReferenceCollection)context.InputParameters["RelatedEntities"];**

## 63. How to [Passing configuration data using Plug-in registration tool (Secured vs. Unsecured)](http://rajeevpentyala.wordpress.com/2012/10/22/passing-configuration-data-using-plug-in-registration-tool-secured-vs-unsecured/)?

* CRM plugin registration tool contain “Unsecure & Secure” configuration sections while registering a “Step”
* We can pass configuration data and can be used in the plug-in logic.

Plug-in registration – Secured and Unsecured sections.

**Secured vs. Unsecured**

Below are key differentiations between Secured and Unsecured data configuration

* **Access**
  + Data passed through “Unsecure” section is PUBLIC (i.e., It can be read by any user in CRM).
  + Only users with “System Administrator” role have access to the data passed through “Secure” configuration section
* **Storage**
  + “Unsecure” config data will be stored along with the Plugin ‘Step’ registration information (i.e., In SdkMessageProcessingStep entity)
  + “Secure” config data will be stored in a separate entity named “**SdkMessageProcessingStepSecureConfig**”
  + Only “System Administrator” has Read access on this entity, hence only users with ‘Sys Admin’ role can access this data
  + Both “Secured & Unsecured” configuration data stored as “Plain text” in DB
* **Outlook Sync**
* “Unsecured” configuration data is downloaded to the user’s computer when they go offline making it Unsecure
* “Secured” configuration data is NOT downloaded to User’s Computer when they go Offline.

**64. How to read Configuration data in Plug-in?**

In our plug-in class, we can define a constructor that passes two parameters (i.e., unsecure configuration and secure configuration)

public class AccountCreateHandler: IPlugin{

public AccountCreateHandler(string unsecure, string secure){

// Do something with the parameter strings.

}

public void Execute(IPluginExecutionContext context){

// Do something here.

}

}

**Note** :- If you want to read “Secure” configuration in the plug-in code, either change the user context in plugin registration as “CRM administrator ” or Impersonate to “CRM Administrator” role user in the code

**65. What are the Database Transactions in Plug-Ins?**

* Plug-ins may or may not execute within the database transaction
* You can check if the plug-in is executing in-transaction by reading the ‘IsInTransaction‘ property of IPluginExecutionContext
* Stages 20 and 40 are part of the database transaction while stage 10 and 50 may be part of the transaction
* If plugin throws an exception, every action done during the transaction will be rollback

**Few more Points:**

* 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 Timeout exception is thrown
* If a plug-in needs more processing time than the 2 minute time limit, consider using a workflow or other background process
* ‘Pre-operation’ operations that CRM will do will not be carried out in pre-validation stage.
* If you are deleting a record that has many-to-many relationship records with another entity; these relationship records will still be available in pre-validation stage, but not in pre-operation stage.
* “Target” entity (i.e., pluginContext.InputParameters["Target"])
* It’s the entity on which plug-in registered
* It only contains “dirty” attributes. if you convert to early bound, the value of the unchanged attribute will be null.

**66. How to Handle Exceptions in Plug-Ins?**

For synchronous plug-ins, whether registered in the sandbox or not, the Microsoft Dynamics CRM platform handles exceptions passed back from a plug-in by displaying an error message in a dialog of the web application user interface. The exception message for asynchronous registered plug-ins is written to a System Job (**AsyncOperation**) record that can be viewed in the System Jobs area of the web application.

For synchronous plug-ins, you can optionally display a custom error message in the web application by having your plug-in throw an **InvalidPluginExecutionException** exception with the custom message string as the exception **Message** property value. If you throw **InvalidPluginExecutionException** and do not provide a custom message, a generic default message is displayed. It is recommended that plug-ins only pass an **InvalidPluginExecutionException** back to the platform.

If a synchronous plug-in returns an exception other than **InvalidPluginExecutionException** back to the platform, the error dialog is displayed to the user and the exception message ([System.Exception.Message](http://msdn.microsoft.com/en-us/library/system.exception.message.aspx)) with stack trace is also written to one of two places. For plug-ins not registered in the sandbox, the information is written to the Application event log on the server that runs the plug-in. The event log can be viewed by using the Event Viewer administrative tool. For plug-ins registered in the sandbox, the exception message and stack trace is written to the Microsoft Dynamics CRM platform trace.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **67. What is 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.  **68. How will work Impersonation during plug-in registration?**  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.   |  | | --- | | **noteNote** | | 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 [download](http://go.microsoft.com/fwlink/?LinkID=207027). |   **Impersonation during plug-in execution:**  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** 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** 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.  **InvalidPluginExecutionException Class:**  When thrown by a plug-in, the Microsoft Dynamics CRM platform displays the exception message in a dialog of the Web application.  **69. Give me Prevalidation example?**  Did not write these specific plugins myself but I can describe one and give the justification for using 'PreValidation' rather than 'PreOperation'.  **Entity:** Account  **Event:** Delete  Logic: Plugin runs pre validation. Checks that there are no contacts referencing any of the account's addresses. If any are found, stop execution. If not, delete account.  **Example:**  Account 'Stackoverflow' has address 'Jeff Attwood's House' and Contact 'glosrob'. 'glosrob' is referencing 'Jeff Attwood's House' through a customisation. If a user selects to delete 'StackOverflow', we should detect 'glosrob' is referencing an address and prevent the delete.  The reasoning behind this was the developer found that at the PreOperation stage, some aspects of the delete had already happened, namely the cascade deletes. The logic of the plugin requires us to check all contacts - by registering at PreOperation, contacts under the account had already been deleted, rendering the check obsolete.  In our previous scenario, when the user selected to delete 'StackOverflow' Account, the Contact 'glosrob' would be deleted before the plugin runs. Therefore when the plugin did run afterwards, it would allow the delete.  **Pre-Validation vs Pre-Operation**  In CRM 2011, there is a new operation stage pre-validation. In the article, pre-validation stage is described as:  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. 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.  What is useful to know about this stage is that, the pre-operation operations that CRM will do will not be carried out in pre-validation stage.  **For example:**  If you are deleting a record that has many-to-many relationship records with another entity; these relationship records will still be available in pre-validation stage, but not in pre-operation stage. The same thing happens when deleting a record that has one-to-many relationship with another entity; the lookup field on the other entity will be set to null. So if you query the database to retrieve all records that reference record-to-be-deleted at pre-operation stage, CRM will return 0 result. For many-to-one relationships however, record-to-be-deleted will still have references to them as they are not the ones being deleted.  **70. What are the InputParameters["Target"]?**  Target is the entity record that the plugin is executed against. It is late bound, but you can use ToEntity() to convert it into early bound instance. One thing useful to know about this instance is that it only contains “dirty” attributes. Unchanged fields will simply not be there, if you converted to early bound, the value of the attribute will be null.  Please note that setting an attribute to null (removing the value) will result a null valued attribute in the late bound instance, but there is no way to differentiate this in early bound instance.   |  | | --- | | ***Important Note:*** | | 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 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. |  71.How to call plugin on field on Change? <http://social.microsoft.com/Forums/en-US/a65f10aa-683c-4ab6-98be-0447a6c89f0a/call-plugin-on-field-onchange?forum=crmdevelopment>  **What is security level in Plugin?**   * Workflows triggered automatically will run under the security context of the workflow owner. On the contrary, if executed on demand, the security context of the user who executed the workflow will be used. * Plugins execute under the security context of the CRM Web application pool identity (tipically NETWORK SERVICE). As this account typically maps to generic CRM SYSTEM user this typically works fine.   However, within plugins you can make use of impersonation to work under the credentials of the user who is making the request. For doing so, you just need to pass True to the CreatCrmService method under the context object.If you need to always impersonate with a specific user, you can do that by passing True as above and setting impersonatinguserid attribute while registering the plugin.It is important to know that plugin impersonation does not work offline. The logged on user credentials are always used in this case.  **72. How do you get the credentials of the logged in user while in a plugin?**  When I use **System.Net.CredentialCache.DefaultCredentials** I am getting the credentials of the Network Service account.  **73. I need to get the credentials of the person that is logged in so I can pass them to an external web service. We are using an on premise installation of crm 2011 and using Windows Authentication connected with our active directory. I have impersonation turned on in the web.config file. Any suggestions?**  We have to use InitiatingUserId in IExecutionContext. 74. Difference between Secure / Unsecure Configuration of Plugin Registration tool in CRM 2011? As you all know the plugin registration tool in CRM 2011 contains Secure and Unsecure configurations.   |  |  | | --- | --- | | Unsecure Configuration of  Plugin Registration tool in CRM 2011 | Secure Configuration of Plugin  Registration tool in CRM 2011 | | Unsecure configuration information could be read by any user in CRM. Remember its public information (Eg: Parameter strings to be used in plugin could be supplied here) | The Secure Configuration information could be read only by CRM Administrators.(Eg: Restricted data from normal user could be supplied here) | | Imagine that you include a plugin, plugin steps and activate them in a solution. Later solution was exported as Managed Solution to another environment. In this scenario, the supplied Unsecure configuration values would be available in the new environment. | Imagine that you include a plugin,plugin steps and activate them in asolution. Later solution was exportedas Managed Solution to anotherenvironment. In this scenario, thesupplied Secure configuration  information would NOTbe available in the new environment. | |

**75. The Secured vs. Unsecured data configuration?**

Below are key differentiations between Secured and Unsecured data configuration

* **Access**
* Data passed through “Unsecure” section is PUBLIC (i.e., It can be read by any user in CRM).
* Only users with “System Administrator” role have access to the data passed through “Secure” configuration section
* **Storage**
* “Unsecure” config data will be stored along with the Plugin ‘Step’ registration information (i.e., In ***SdkMessageProcessingStep*** entity)
* “Secure” config data will be stored in a separate entity named “***SdkMessageProcessingStepSecureConfig***”
* Only “System Administrator” has Read access on this entity, hence only users with ‘Sys Admin’ role can access this data
* Both “Secured & Unsecured” configuration data stored as “Plain text” in DB.
* **Outlook Sync**
* “Unsecured” configuration data is downloaded to the user’s computer when they go offline making it Unsecure.
* “Secured” configuration data is NOT downloaded to User’s Computer when they go Offline.

**76. How to read Configuration data in Plug-in?**

In our plug-in class, we can define a constructor that passes two parameters (i.e., unsecure configuration and secure configuration)

public class AccountCreateHandler: IPlugin{

public AccountCreateHandler(string unsecure, string secure){

// Do something with the parameter strings.

}

public void Execute(IPluginExecutionContext context){

// Do something here.

}

}

**Note** :- If you want to read “Secure” configuration in the plug-in code, either change the user context in plugin registration as “CRM administrator ” or Impersonate to “CRM Administrator” role user in the code

**In case of Remote Debugger** :-

* If your CRM server machine is different from your development machine
* Install “Visual Studio Remote Debugger” ([Download](http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=475)) on CRM server machine
* Run it as “Administrator”
* Follow the same steps above except while attaching Process, in the “Attach to Process” window, set “Qualifier” as “CRM server machine”.

**Debug plug-in in outlook offline mode**

* Clean and rebuild the plug-in solution on your machine using visual studio
* Register the plugin on the server
* Synchronize the organization with the outlook
* Go Offline
* Attach the debugger to the process “**Microsoft.Crm.Application.Hoster.exe**”
* Place a breakpoint in the code.
* Run the scenario

**77. How to** [**Configure Tracing and view trace log files in CRM**](http://rajeevpentyala.wordpress.com/2012/08/16/configure-tracing-and-view-trace-log-files-in-crm/)**?**

Tracing is important in CRM especially when your plugin throw exceptions in Test or Production environments.

We can configure and view the trace log file easily using “Crm Trace Log Viewer” tool and it work for both CRM 4.0 and 2011.

**Note** - Don’t forget to disable the tracing after you are done with debugging.

* To disable tracing, open the tool and uncheck the option “Trace Enabled” (Refer image #4)

**Outlook Client Tracing**

* Using CRM 2011 “Diagnostics Tool”, we can enable/disable tracing in outlook client
* Here is the [link](http://www.powerobjects.com/blog/2012/11/08/crm-2011-outlook-client-tracing/)

**78. Where is Secondary Entity used?**

The secondary entity only applies to the SetRelated and RemoveRelated messages.

# 79. [How To Increase Plugin Time Out in Microsoft Dynamics CRM 2011](http://azwaralam.blogspot.com/2013/06/how-to-increase-plugin-time-out-in.html)?

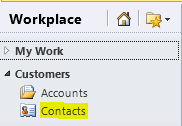
If you want to increase plugin time out for large data processing following is the solution:

* Open Web.config file of Microsoft Dynamics CRM 2011 in notepad.
* Search httpRuntime tag and increase the timeout.
* save the web.config file and then reset iis server.
* Setting default view using Plug-in’s in CRM 2011

We often get the requirement to set a default view based on some criteria (Ex – Could be based on logged in user role, etc…). We can achieve this using a Plug-in.

**Little insight** :-

* When you click on an entity (i.e., ‘Contacts’ in my sample) in CRM application, In the background a “RetreieveMultiple” method call happens on “savedquery” entity.

[](http://rajeevpentyala.files.wordpress.com/2012/01/sitemap-link1.png)

* “savedquery” is a system entity in CRM which stores the meta data of all the views in the system

**Plug-in Logic** :-

* We register a Plug-in for the RetreieveMultiple message on the savedquery entity to be executed in the post stage.
* From “Outputparameters” fetch the “BusinessEntityCollection” (i.e., Views of particular entity)
* Loop through the views and set “isdefault” property to “true” for the desired view.