Plugin Development for Dynamics 365

by Alex Shlega

Email: ashlega@yahoo.com

Linkedin: https://www.linkedin.com/in/alexandershlega/

Web: <u>www.itaintboring.com</u>

Contents

Pre-	-Requisites	4
1.	Overview and Introduction	5
2.	Finding Help When You Need It	6
3.	Setting up the Dev Environment	7
4.	Setting up the Project	9
5.	Configure your Dynamics 365 trial instance	12
6.	Check Point 1	14
7.	Plugin Registration Tool	15
8.	First plugin: let's add some validation	20
9.	Why using a plugin in the validation scenario above?	26
10.	Check Point 2	27
11.	Context, Target	28
12.	First plugin updated: let's make it smarter	31
13.	OrganizationService	41
14.	Check Point 3	44
15.	Quiz	45
16.	Exercise	49
17.	Check Point 4	50
18.	Entity, EntityReference, Attribute	51
19.	Plugin step attribute - first plugin fine tuning	54
20.	Working with different attribute types	57
21.	Check Point 5	60
22.	Sample Customization Requirements	61
23.	Plugin #1: Pushing priority field updates to the related company contacts	63
24.	Plugin #1: Using Queries with OrganizationService	66
25.	Record Count, Paging, and Total Record Count	69
26.	Exercise	73
27.	Check Point 6	74
28.	Plugin #1: Using a QueryExpression	75
29.	Plugin #2: Validating the credit limit	77
29.1	1 Setting it up	79
29.2	Pre-Operation, Post-Operation, Pre-Image and Post-Image	83

29.3	Adding a Pre Image	87
29.4	Implementing the validations	92
30	Check Point 7	98
31	Exercise: Plugin #3	99
32	Plugin #3: Q & A and Working Sample	. 101
33	Remaining Theory	. 103
34.1 S	andboxed plugins vs Non-Sandboxed plugins	. 104
34.1	Synchronous vs Asynchronous	. 106
34.2	Transactions	. 108
34.3	Pre-Validation	. 110
34.4	Plugin Execution Order	. 112
34.5	Plugins Recursion	. 114
34.6	Early Bound vs Late Bound	. 117
35	Debugging the plugins	. 119
35.1	(Coming Soon) Raising errors: InvalidPluginExecutionException	. 121
35.2	(Coming Soon) Tracing Service	. 122
35.3	(Coming Soon) Plugin Profiler	. 123
36	Quiz	. 124
36.	1 You are registering a plugin step on Create message. Which "images" do you have access t 124	ο?
36. cod	2 In your update plugin, you just got Target entity from the plugin context, and you have thi le there:	
36. the	3 You are adding a Pre Image to the step using the Plugin Registration Tool, and you are get error message below:	_
36.	4 You have a QueryExpression, and you want to retrieve the first record from the result:	. 125
36 cod	, ,	this
36.	6 Have a look at the plugin code below – do you see any problems there?	. 125
36. get	7 You have compiled a plugin assembly, and you are trying to register it in Dynamics. You ar ting the error below – what do you still need to do?	

Pre-Requisites

- You have a workstation with VS 2017 installed on it (for this course, Community Edition should work just fine)
- You have access to Dynamics 365 online. You can open a trial instance if you don't have one provided by your organization
- You must be familiar with the basic Dynamics concepts: entities, attributes, form configuration screens, importing solutions, etc
- You must be familiar with .Net development you don't have to be an expert, but you should feel comfortable with the basic C# syntax. From there, the more you know the better. Although, truth be told, I'm not a .Net development expert myself

1. Overview and Introduction

If you are used to the training courses where you listen a lot and do little, you are up for a different experience. This time around, you will have to do quite a bit of work, so think of it more as of a bootcamp. After all, practice makes perfect, and that's exactly what we are going to do – we are going to practice the skills you will need to become a Dynamics 365 plugin developer.

I have been doing Dynamics 365 development for a while, and, if there is anything I learned, it's that there are no shortcuts. I can talk all day long about plugins development, but, in the end, it's only when you try it yourself you will start to realize what I was talking about.

That's why this is going to be a very practical course. Maybe at the expense of some of the finer details – you can easily find those online once you know what you are looking for.

We will set up the environment, I will give you a bit of an overview, and, then, we will start digging into it together. You will be writing the code, you will be compiling it, and you will be deploying those plugins in the actual Dynamics 365 environment. What you will, hopefully, take away from this course is the knowledge of not only how to do this kind of development, but, also, that you, personally, can do it.

But, before we proceed, let's do the introductions.

I've been working as a Dynamics developer/consultant/solution architect since 2010. The very first Dynamics project I worked on probably defined my future career in Dynamics. I could have become a functional consultant, but, since that project turned out to require quite a lot of code customizations, that's where it all started. Fast forward, I now have a blog where you will, hopefully, find some useful information on Dynamics:

http://www.itaintboring.com/

Back in July 2017, I was awarded a Microsoft Community Contributor badge. If you are not aware of the community forums, there is a great online community out there – should you have a question about Dynamics, it would be a great place to ask:

https://community.dynamics.com/crm/f/117

And, most recently, I received an MVP award from Microsoft.. which, quite frankly, I am still finding hard to believe at myself:



So, let's get to work. And, if there are any questions, just ask!

2. Finding Help When You Need It

You may not know anything about plugin development yet, so it may look strange that I am not starting this course with all the details on how the plugins work, what they are, etc. Truth is, you will not become an expert even once you have completed the course. I am hoping it will help you do your first most important steps, but, from there, you will have to take it further on your own.

How much further really depends on the role you are working in, on your personal preferences, on your curiosity, and on a lot of other things.

What I am sure about is that you will need to know where to find all the additional information you may need and where to ask all the questions you may have along the way. Even as soon as later today, while reflecting on the training material.

So, here is the list of my favourite links:

- a) Dynamics Community Forum: https://community.dynamics.com/crm/f/117
- b) Developer Guide: https://docs.microsoft.com/en-ca/dynamics365/customer-engagement/developer
- c) Andrii Butenko blog: http://butenko.pro/blog/
- d) Guido Preite blog: http://www.crmanswers.net/
- e) Neil Parkhurst blog: https://neilparkhurst.com/
- f) My own blog? http://itaintboring.com
- g) There are lots of other blogs out there I simply cannot mention all of them

And, of course, there is this one:

http://lmgtfy.com/?q=how+to+write+a+plugin+for+dynamics+365

You may find that Andrii (see above) will sometimes refer aspiring Dynamics developers to that last link when answering questions in the community forums.

Now that you have some idea about the online resources available to you, let's set up the development environment.

3. Setting up the Dev Environment

- 1. Make sure you have Visual Studio 2017 installed (Community Edition, at least)
- 2. Create Dev folder on the C drive
- Download ScriptsAndTools.zip from http://itaintboring.com/downloads/training/ScriptAndTools.zip
- 4. Create SDK subfolder
- 5. From the zip file you just downloaded, copy downloadtools.ps1 file directly into the SDK subfolder
- 6. Run the script, wait till it's completed, and have a look at the folders structure

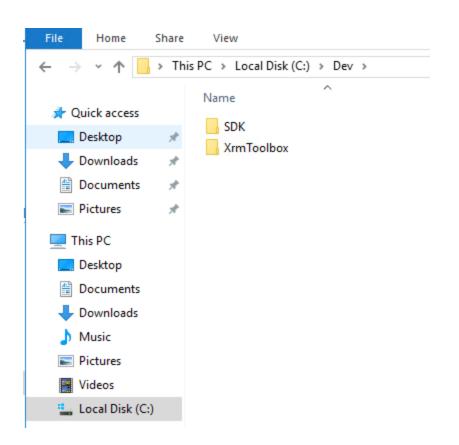
By running that script, you just downloaded a few SDK tools from Microsoft. We will look at some of them later, but, if you had any experience with pre-V9 SDK, you may have noticed the difference. Here is why:

https://docs.microsoft.com/en-ca/dynamics365/customer-engagement/developer/download-tools-nuget

Starting with the V9 version of Dynamics 365, this is how you are supposed to download the SDK. For better or worse, it's not distributed as a single downloadable package anymore.

One other tool you may need to know about is XrmToolBox. Actually, this is not a single tool. It's, really, a host application for a lot of other tools.

- 7. Download XrmToolBox from https://www.xrmtoolbox.com/
- 8. Unzip to c:\dev

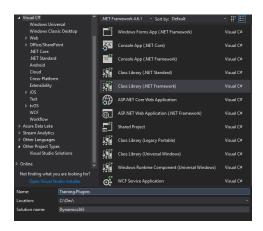


4. Setting up the Project

Now that we have the environment ready, let's set up our first plugin project

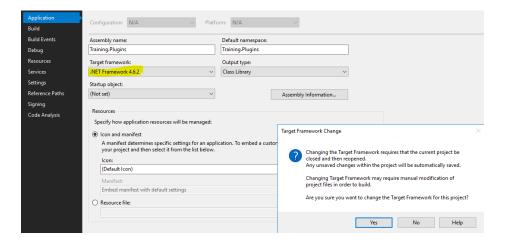
1. Create a new Class Library project in the C:\Dev folder. Call the solution "Dynamics365", and call the class library "Training.Plugins".

We will keep building on top of this initial solution later in the course.



2. Make sure target framework version is set to 4.6.2

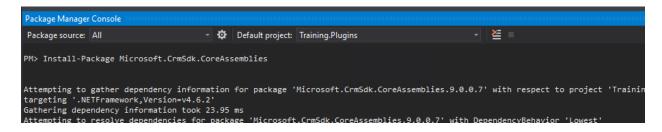
NOTE! If you don't see .NET Framework 4.6.2 in the list of target frameworks, you will need to install the developer pack for that version from the url below: https://www.microsoft.com/net/download/visual-studio-sdks



"You should build any custom client applications using Microsoft .NET Framework 4.6.2 or later. Starting with the Dynamics 365 (online), version 9.0, only applications using Transport Level Security (TLS) 1.2 or better security will be allowed to connect. TLS 1.2 is not the default protocol used by .NET Framework 4.5.2, but it is in .NET Framework 4.6.2."

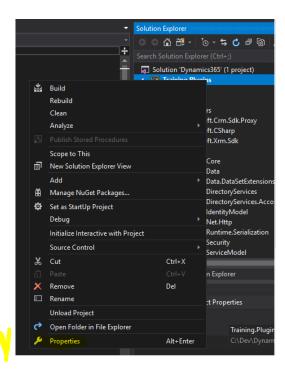
https://docs.microsoft.com/en-ca/dynamics365/customer-engagement/developer/visual-studio-dot-net-framework

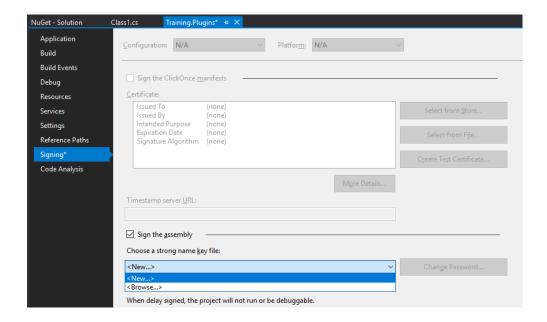
- 3. Download Dynamics SDK NuGet packages
 - From the Tools menu, select NuGet Package Manager->Package Manager Console
 - Use the following command to install the packages:
 Install-Package Microsoft.CrmSdk.CoreAssemblies

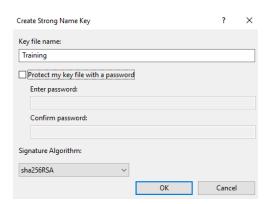


Those NuGet packages contain core SDK reference assemblies you'll be using to build the plugins.

4. Sign your project in the Visual Studio – right click on the project in the solution explorer, choose properties, then "Signing", then create a new key:







5. Build the project, make sure it's all good so far.

5. Configure your Dynamics 365 trial instance

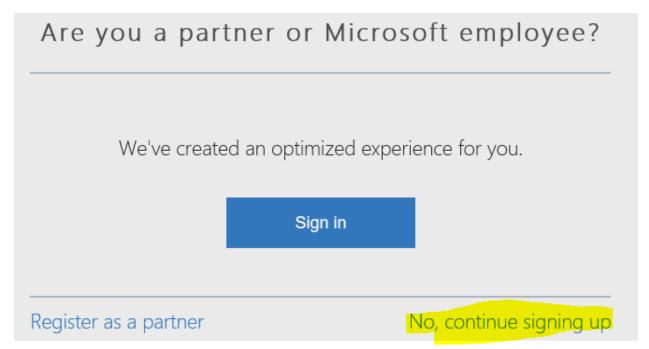
To create a trial of Dynamics, go to this url first:

https://trials.dynamics.com/Dynamics365/Signup#

And choose the option highlighted below:

Are you signing up on behalf of a customer or using this trial for development purposes? Sign up here.

Then choose "continue signing up"



Proceed through the signup wizard, and choose "all applications" when asked if you need it for sales/services/etc.

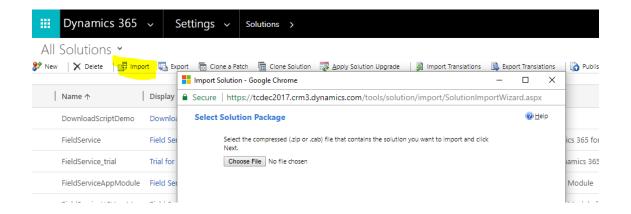
Once you have your trial instance ready, open up a browser and navigate to your trial instance. You will need to deploy the training solution now.

First, download the training solution:

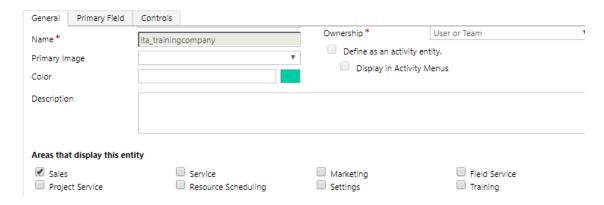
http://itaintboring.com/downloads/training/Training 1 0 0 0.zip

There are a few custom entities/other components in that solution which we will be using in the plugins throughout the course. That's an unmanaged solution which will install a few custom entities we'll be using throughout the course.

In Dynamics, navigate to the Settings->Solutions, click Import, and deploy the solution file:



Once the solution has been imported, open it in Dynamics, and, for each entity there, add it to the sales area:



This is to make the navigation easier. Then use Publish All Customizations, and you should be good to go.

6. Check Point 1

- There is a Virtual Machine with Visual Studio 2017 with a plugin project created in it
- You have downloaded all the basic SDK tools and XrmToolBox
- You have access to the trial instance of Dynamics 365
- You have installed the training solution in that instance of Dynamics 365



What's Next?

We will start looking at the actual plugins right after that. For each of those plugins, I will provide the source code and, then, will walk you through the steps required to get that plugin deployed to Dynamics. Once everyone has the plugin registered, we will all test the results, and, then, will have a quick Q&A session before we proceed to the next plugin. Along the way, I will be introducing a few tools you will need to become familiar with, and, also, we will be discussing different aspects of Dynamics 365 architecture – this will help you understand what's happening to your plugins behind the scene.

7. Plugin Registration Tool

As the title above implies, Dynamics plugins must be registered. That brings up at least a few questions:

- How do you register a plugin?
- Where do you register a plugin and why do you need to register it?
- And what the heck is a plugin, then?

You will get better understanding as you progress through the course, but I'll explain at least some of that now.

A plugin is, really, just a .NET class which implements an IPlugin interface:

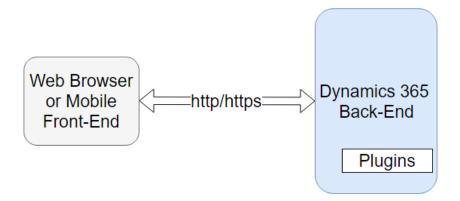
```
using Microsoft.Xrm.Sdk;
namespace ItAintBoring.Dynamics.Samples.Plugins
{
   public class SamplePlugin: IPlugin
   {
     public void Execute(IServiceProvider serviceProvider)
     {
        ...
     }
   }
}
```

Plugins are organized into the class libraries, which, in turn, are compiled into dll files. You can actually have more than one plugin in each of those dlls.

However, you can certainly compile a plugin in the Visual Studio, you can get a dll, but what do you do with it? This is where we need to talk about the plugins architecture in Dynamics.

What can a plugin do? How does it start? Where does it run? Who is supposed to run it?

We need to step back for a second and think of how Dynamics works, at least on the high level (don't think of this diagram in terms of servers yet):



Dynamics users would open a browser, they would navigate to a certain url, and, then, they would start working with Dynamics.

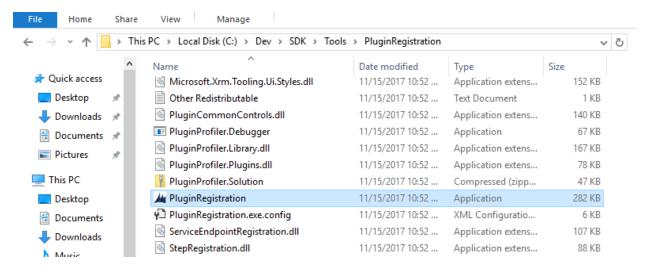
As they keep working, they will be taking certain actions. They will be creating records, updating records, deleting records, assigning those records to other users, and so on.

From this standpoint, here is what you need to know about the plugins:

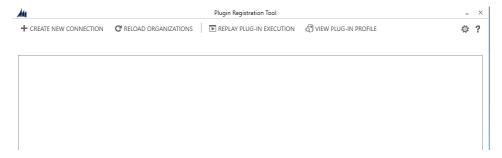
- Plugins can only run on the back-end, never on the front-end.
- For every plugin, you need to tell Dynamics that there is a plugin that needs to run for a certain action (what is called a "message"). This process is called plugin registration, and you only need to do it once. Unless, of course, you need to change some of the registration details. Actually, it's a bit more complicated since there are a few other things you have to configure when registering a plugin, but that's the idea. We'll get to the details shortly.

While setting up the environment earlier, we've used a powershell script to download the SDK tools. One of those tools is called Plugin Registration—please go ahead and find it in your environment now:

C:\Dev\SDK\Tools\PluginRegistration

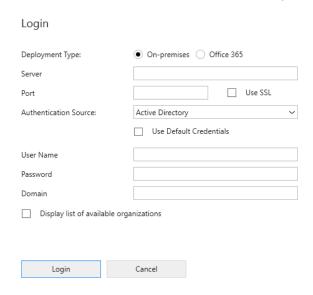


Then, start the tool:

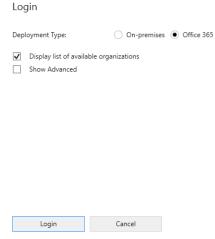


This is not much so far, but keep going.

Click Create New Connection button at the top:

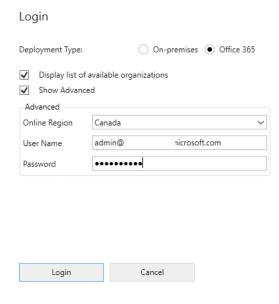


As you can see, there are a few options there, but we will need to choose Office 365. Also, enable "Display list of available organizations" checkbox:



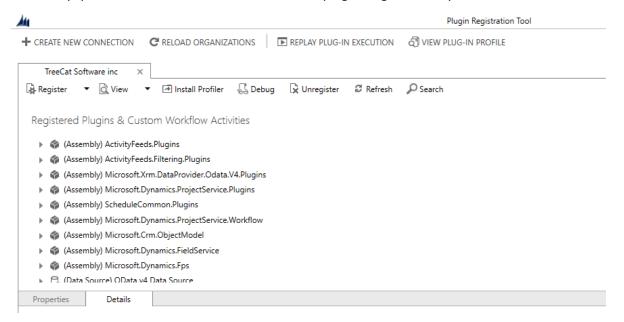
And click login. You will be asked for the user credentials – make sure to provide the credentials for your Dynamics 365 trial instance.

This is where you may or may not get an error even if you have provided correct credentials. Unless you have selected "Show Advanced" option above, Plugin Registration utility will try to choose the region automatically, and, depending on where you are trying to run it from, it may or may not be able to do so. If an error happens, you should cancel the login, and, unless Plugin Registration utility logs in automatically after that, try restartig it and following the same steps again, but choose "Show Advanced" option this time. Then you should see this kind of screen:



Choose your online region and login.

Eventually, you should see the list of out-of-the-box plugins registered in your trial instance:



This may look unexpected, but yes, there are plugins there, already, even though you have not yet started to develop anything. Well, there is, actually, nothing you should do with those out of the box plugins. First, you'll need to create your own plugin, which is exactly what we are going to do. But keep the plugin registration utility open for now – you'll need it again soon.

8. First plugin: let's add some validation

Scenario:

As part of the training solution, you have deployed a "Training Company" entity. For the Training Company entity, you want to make sure that your Dynamics users stop using "Test" as a company name. And, if they try doing that, you want Dynamics to display an error message.

Plugin overview:

It's really easy to implement this kind of validations using plugins. Remember we talked quickly about actions/messages before? What we need here is a plugin that runs whenever a Training Company record is created and/or whenever it is updated. The plugin should check the value entered into the "name" field, and, if there is "Test" there, the plugin should throw an error.

Now, if you have never developed a plugin for Dynamics, all of the above may sound a little cryptic to say the least. Don't you worry - we'll get through.

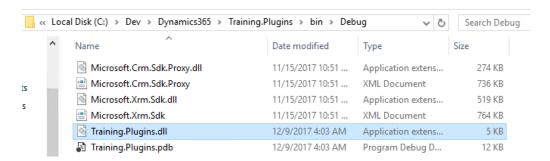
Creating a plugin:

- 1. Open Dynamics 365 solution that we created before in the the Visual Studio
- 2. Delete Class1.cs file (permanently)
- 3. Add a new C# class, call it ValidationPlugin:

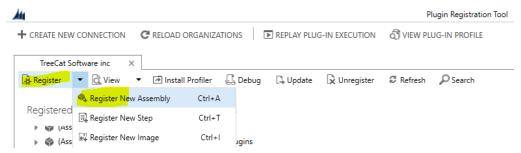
4. Replace the code in that file with the following code:

```
using System;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;
namespace Training.Plugins
{
    public class ValidationPlugin : IPlugin
        public void Execute(IServiceProvider serviceProvider)
            IPluginExecutionContext context = (IPluginExecutionContext)
                serviceProvider.GetService(typeof(IPluginExecutionContext));
            Entity entity = (Entity)context.InputParameters["Target"];
            string name = (string)entity["ita_name"];
            if ("Test".Equals(name))
                throw new InvalidPluginExecutionException("Cannot use this name");
            }
        }
    }
}
```

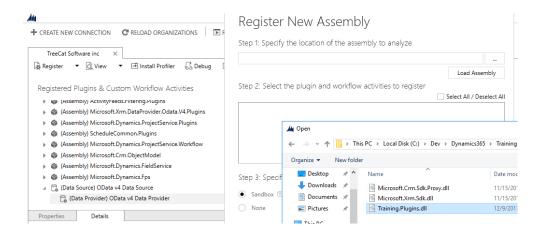
5. Build the project – once Visual Studio has finished building the project, you will find Training.Plugins.dll file in the bin\Debug subfolder:



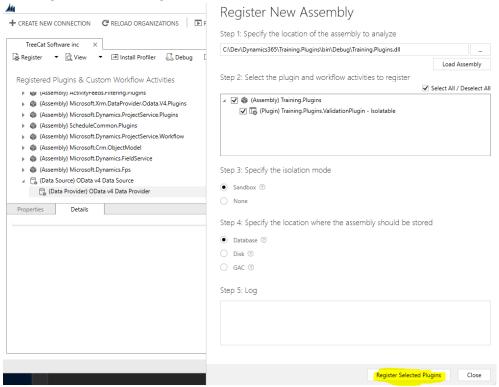
6. Use the PluginRegistration tool (which you should still have open there) to register plugin assembly first



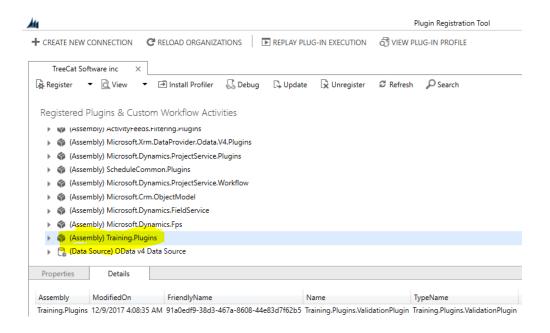
7. Choose Register New Assembly, then select plugin dll:



8. Click Register Selected Plugins button



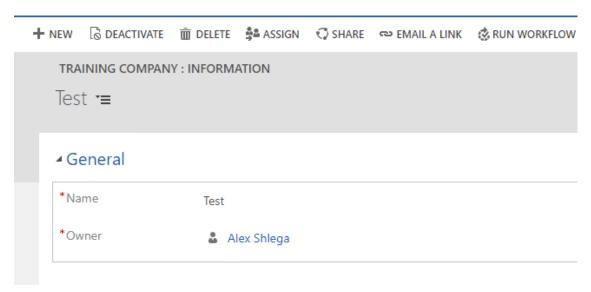
9. If all goes well, you should see your plugin added to Dynamics:



Let's take a pause and talk about what just happened. You have created a plugin dll. You have registered it in Dynamics. Is there anything else you need to do? Remember I mentioned those actions/messages earlier? So far, even though the dll has been deployed to Dynamics, we still have not specified which plugin we are going to use and when.

There are hundreds of entities in Dynamics – will this same plugin be running on the update/create/delete of each and every record or can it be configured somehow?

To start with, let's go to Dynamics and create a new "Training Company" record – just make sure to put "Test" in the name:

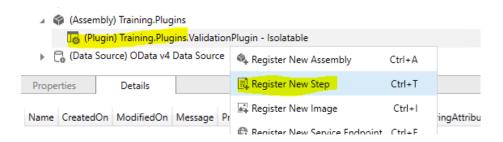


See? We have a plugin in Dynamics that's supposed to validate the value you put in the "Name" field, but, apparently, something is missing since it's not doing anything yet.

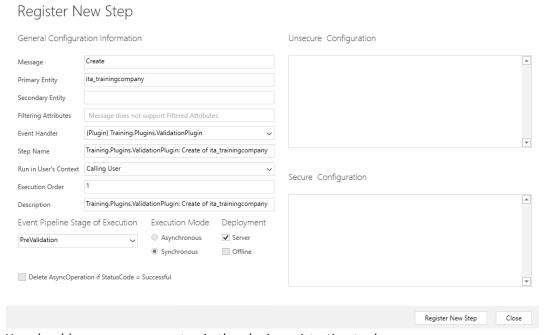
What's missing it an SDK Message Processing Step. This is where we will specify when the plugin will run, for which entity, for which message, if it will run before or after the "action", etc.

Let's go back to the Plugin Registration Tool and continue the exercise:

10. In the Plugin Registration Tool, expand Training. Plugins node, select the Validation Plugin, right click there, and, from the popup menu, choose "Register New Step" option:



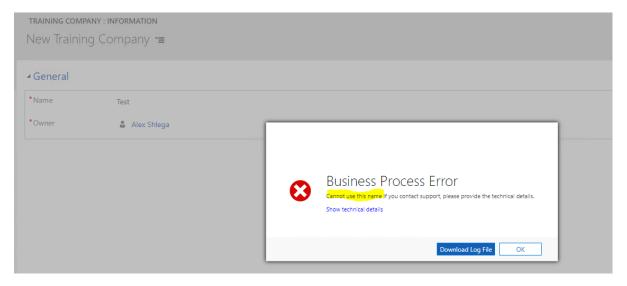
- 11. Set up the step as per the screenshot below
 - Type in "Create" into the Message field
 - Type in ita_trainingcompany into the Primary Entity field
 - Leave the rest as is
 - Then click "Register New Step"



12. You should, now, see a new step in the plugin registration tool

- ✓ (Assembly) Training.Plugins
 ✓ (Plugin) Training.Plugins.ValidationPlugin
 ✓ (Step) Training.Plugins.ValidationPlugin: Create of ita_trainingcompany
- (Assembly) Microsoft. Dynamics. Field Service

Let's do the same test again – go back to the browser and open Dynamics 365. If you have it open, you don't need to refresh the screen. Just start creating a new Training Company record, type in "Test" in the name field, click "Save".. and you should see this:



The error message might not look too user friendly, there are those additional buttons, and so on. However, the plugin just worked!

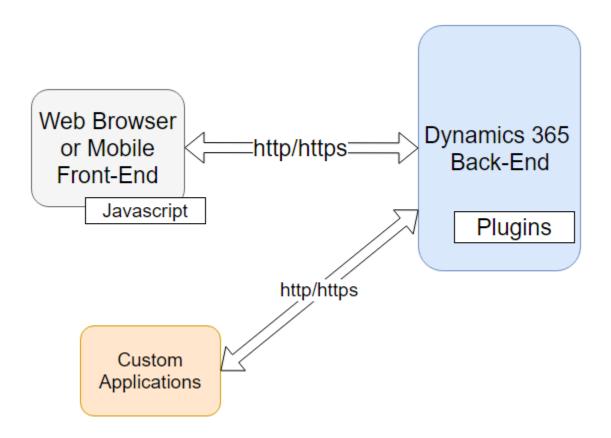
Click "Ok" in that popup window, and try using "New" instead of "Test" for the name field. There will be no error messages this time:



9. Why using a plugin in the validation scenario above?

Could we use a javascript? Or a workflow? Or a business rule?

We could, but have a look at the diagram below:



Javascripts will run on the client side only. So, even if we had validations there, they would not work for the custom applications.

It's a different story with the business rules and workflows – you can use those for server-side validations, too. However, business rules can be configured and cannot be customized through development. There is only so much you can configure there, and you certainly can't do complex calculations or queries.

Workflows can be configured and customized (using custom workflow activities). From that standpoint, the difference between plugins and workflows may not be that huge. Plugins are still somewhat more flexible when it comes to configuring the trigger conditions, pre-images, post-images, messages. That's a lot of terminology you may not be familiar with yet, though, so we should probably be moving on.

But, just out of curiosiy, can you come up with some scenarios where using a plugin for validation won't be good enough?

10. Check Point 2

- You just created your first plugin, and you registered it in Dynamics 365
- You have learned that you need to use Plugin Registration Tool to register the plugins
- You have also learned that plugins will never run on the front-end, they will only run on the back-end. It's not enough to simply register a plugin assembly – you need to register a Message Processing Step for the plugin to work
- We did discuss, at least on the high level, why using a plugin may be more suitable in the data validation scenarios when compared to javascripts. And we also looked at why, in some cases, it may work better the other way around



What's Next?

We need to start digging into the details. Why and how did that plugin work? What if you wanted to make a different plugin? What are the various registration options you have probably seen on the screen? Moving on.

11. Context, Target

Plugins are not WCF/ASP.NET/Web applications. They are just not – when thinking about the plugins, don't think about the WCF/ASP.NET. Those are two completely different architectures, and Dynamics plugins don't have anything to do with web development.

Which means, for example, that you cannot access request parameters in the plugins the way you would do it in WCF.

Still, when developing a plugin you need to know the exectuion context – what was happening on the client, what is the record being processed, what are the fields being updated, what is the action being taken.



What is the action?

But, also:

- What are the fields?
- What is the entity name?
- What is the entity id?
- Etc

And, besides, Dynamics won't necessarily create a new instance of your plugin for every call. It can be the same plugin instance for create/update/delete, for different records, etc.

How do you identify the context of each of those calls, and how do you identify the entity/fields for which the plugin is running?

Every plugin is expected to implement "Execute" method of the IPlugin interface, and there is a single parameter passed to that call:

```
public void Execute(IServiceProvider serviceProvider)
```

IServiceProvide is a common interface not specific to Dynamics that defined just one method:

```
object GetService(
          Type serviceType
)
```

Using the serviceProvide, you can get access to the contextual information specific to that plugin execution:

IPluginExecutionContext gives you access to pretty much all the contextual details you may get access to in the plugin:

```
    context.InputParameters["Target"]
    context.PreEntityImages
    context.PostEntityImages
    context.MessageName
    context.Stage
```

You might want to take a look at the MSDN page describing IPluginExecutionContext in more details:

https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.ipluginexecutioncontext.aspx

But, for now, let's get back to the code you used for the first plugin:

```
using System;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;
namespace Training.Plugins
{
    public class ValidationPlugin : IPlugin
        public void Execute(IServiceProvider serviceProvider)
            IPluginExecutionContext context = (IPluginExecutionContext)
                serviceProvider.GetService(typeof(IPluginExecutionContext));
            Entity entity = (Entity)context.InputParameters["Target"];
            string name = (string)entity["ita_name"];
            if ("Test".Equals(name))
                throw new InvalidPluginExecutionException("Cannot use this name");
            }
        }
    }
}
```

Do you recognize anything there now? How about these lines?

First, we are getting the context in exactly the way we just discussed. And, then, we are getting the "Target" from the context.InputParameters.

We will talk about Entities, EntityReferences, and Attributes shortly, but, for now, just keep in mind that "Target" is, really, the in-memory representation of the data that's being updated/created/deleted/etc. It's literally the target of the action.

In other words, if, in the user interface, you are updating the account record, then, in the plugin, "Target" will represent exactly that account.

Let's explore this a bit more, though.

Q1: Will the context be there for every plugin execution?

Yes, it will.

Q2: Will the Target be there for every plugin execution?

No, not necessarily. It depends on the plugin step configuration. Remember we can register a plugin for different messages? Not all of them will have a target. Although, messages like "Create"/"Update"/"Delete" will have a Target.

Q2.1: How do you know what you will find in the "Target"?

You just learn those things as you keep developing plugins. For example, for the "Create" and "Update" plugins, you will always have a "Target", and it will be of "Entity" type. For the "Delete" plugins, you will also have a "Target", but it will be of "EntityReference" type.

With that bit of theory in mind, let's make some changes to the first plugin.

12. First plugin updated: let's make it smarter

To start with, let's see why the code we wrote so far is not very reliable yet. Let's register the plugin on "Update" first –give it a try on your own first. Although, no worries if you are not sure how to do it yet.

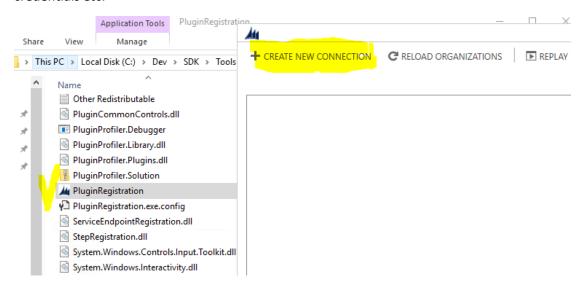
(Hint: you will certainly need to use the Plugin Registration Tool)

So, let's see if you can recall the steps we went over when registring the plugin on "Create".

If you were able to register the plugin on "Update" of the Training Company entity, great! Still, have a look at the step-by-step instructions below to see if you did it right. If you were not able to do it, just follow the instructions below.

Bring up the Plugin Registration Utility and connect to your trial instance of Dynamics

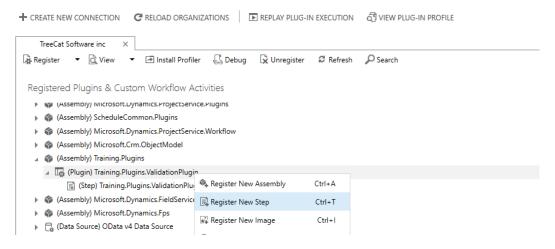
This time, the tool should remember how you did it last time, so you may not have to type in your credentials etc.



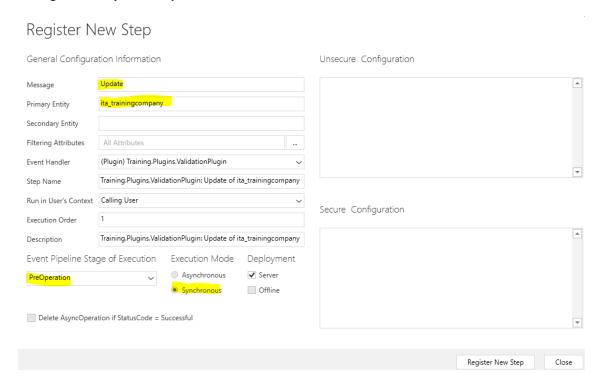
Login



Select Training.Plugins assembly, right-click on the Training.Plugins.ValidationPlugin, and choose "Register new step" item



Configure the Update step



Here is what you should see once the step has been registered

(Assembly) Training.Plugins

(Plugin) Training.Plugins.ValidationPlugin

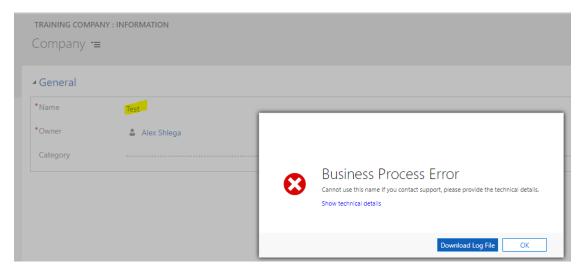
(Step) Training.Plugins.ValidationPlugin: Create of ita_trainingcompany

(Step) Training.Plugins.ValidationPlugin: Update of ita_trainingcompany

Now that we have the plugin registered on Update, we can try a couple of things:

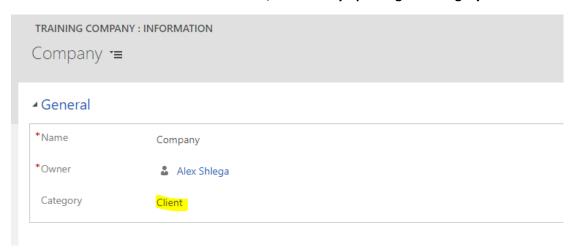
- Let's try updating the name field for an existing Training Company
- And, then, let's try updating Category field for an existing Training Company while keeping the name as is

First, let's try updating the Name field



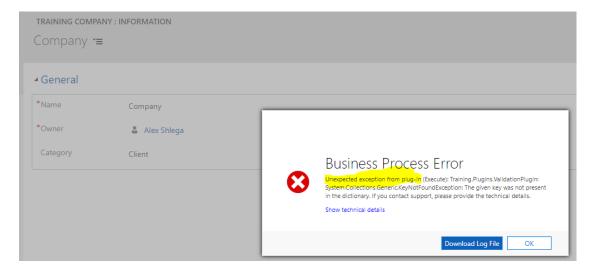
Everything looks good – still getting the error message as expected

Now let's leave the Name field untouched, but let's try updating the category field instead



Can you guess what's going to happen?

And click "Save"



It's an unexpected error!

If you look at the error type, which is GenericKeyNotFoundException, you can probably guess what went wrong. The problem is that, unless a field is populated on the client side for that particular plugin call, that field will not be added to the Attributes collection of the Target entity.

For example, in the above scenario, we've only updated the Category field. But the plugin is still expecting some value in the "Name" attribute:

```
string name = (string)entity["ita_name"];
```

This does not work because ita_name is not there for this update.

Well, you need to test if it's there, then, and there are at least a few ways you can do it:

- You can use Contains method:

You can re-write the code using GetAttributeValue method:

```
string name = entity.GetAttributeValue<string>("ita_name");
    if ("Test".Equals(name))
    {
        throw new InvalidPluginExecutionException("Cannot use this name");
    }
```

The difference between those two is that GetAttributeValue will not tell you if Name field was populated or not. It will give you a value or null. However, null can also be a valid value when the field is

being cleared. You might not be able to clear the "Name" field since it's a required field, but, if we were using an optional field, you might just make it empty on the update. In the plugin, you would have that attribute in the Target, though the value you'd find null value there.

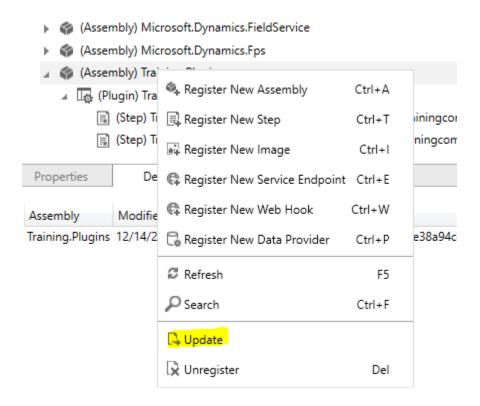
So, for our current scenario either of those methods above would work. But, in general, whenever you need to check if a field has been modified as part of the Update/provided as part of the Create, you should be using "Contains".

Let's re-write the plugin using Contains:

```
using System;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;
namespace Training.Plugins
{
    public class ValidationPlugin : IPlugin
        public void Execute(IServiceProvider serviceProvider)
            IPluginExecutionContext context = (IPluginExecutionContext)
                 serviceProvider.GetService(typeof(IPluginExecutionContext));
            Entity entity = (Entity)context.InputParameters["Target"];
             if (entity.Contains("ita_name"))
                 string name = (string)entity["ita_name"];
                 if ("Test".Equals(name))
                   throw new InvalidPluginExecutionException("Cannot use this name");
             }
        }
}
```

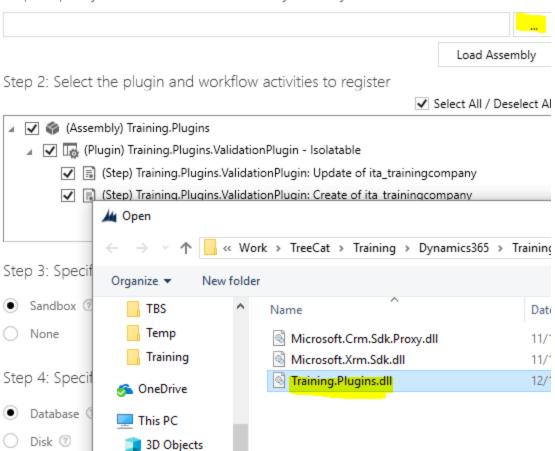
And re-compile the plugin.

Once you've got the updated plugin compiled, you need to update the dll in Dynamics. Go back to the Plugin Registration Tool, just keep in mind that the plugin and steps are already there, so this time we just need to update the dll(assembly):



Update Assembly: Training.Plugins

Step 1: Specify the location of the assembly to analyze



Make sure to enable the checkboxes on the next screen:

Update Assembly: Training.Plugins

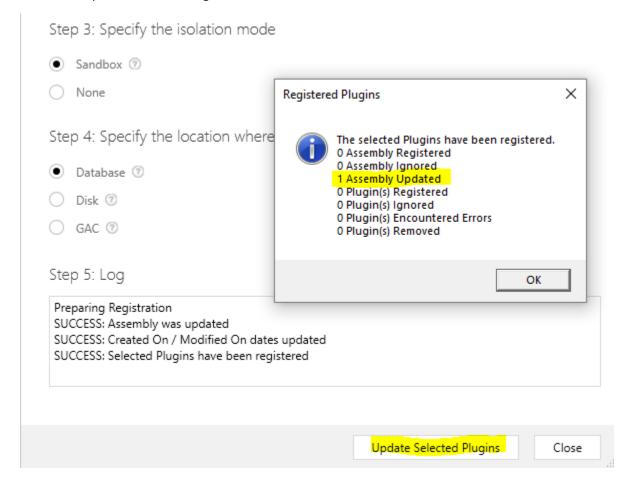
Step 1: Specify the location of the assembly to analyze

C:\Work\TreeCat\Training\Dynamics365\Training.Plugins\bin\Debug\Traini	ng.Plugins.dll	
✓ Update Assembly Properties and Content	Load Asse	mbly
Step 2: Select the plugin and workflow activities to register		
	Select All / De	select All
🗸 📝 🏟 (Assembly) Training.Plugins		
(Plugin) Training.Plugins.ValidationPlugin - Isolatable		

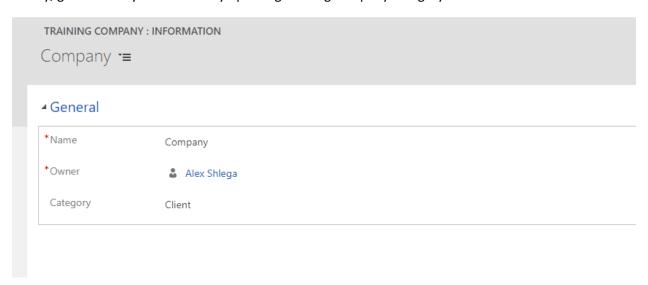
Step 3: Specify the isolation mode

Sandbox ?

And click "Update Selected Plugins"



Finally, go back to Dynamics and try updating training company category:



Everything works just as we wanted this time – there are no errors anymore.

13. OrganizationService

It helps when we have access to the Target, and it's certainly useful when we can do the validations, but that's not what plugins are all about. What really makes them work for lots of different scenarios is that we get access to the so-called Organization Service.

Organization Service represents a connection to Dynamics. Historically, Microsoft has been maintaining more or less the same Organization Service interface for a long time now (I am not sure about pre CRM4 versions, but, at least starting with CRM4, there has been almost the same organization service interface, at least as far as plugins development goes). The underlying implementation might have been updated a few times, some features might have been added and/or removed. But, in general, it's been the same idea and the same familiar "interface".

With the Organization Service, you can, basically, execute requests. You can query data, you can delete data, you can create, assign, update, associate data. You can query the metadata, you can publish metadata changes, you can even create new entities, fields, option set values, etc.

In general, Organization Service uses Execute method to execute such requests, and that method returns a response object as a result.

Here is a typical scenario:

Notice that call to service. Execute. That's where the CreateRequest is sent to the Dynamics server.

There are shortcuts for such methods as "Create", "Update", "Delete", etc. For example, you might rewrite the same code above in the following way:

```
Entity entity = new Entity("account");
entity["name"] = "Test";
entity.Id = service.Create(entity);
```

That would be a shorter form. Not every request would have this sort of a shortcut, though, so just keep that in mind.

We don't always need an Organization Service in the plugins - we can get "Target" from the context, we can get the attributes, we can raise exceptions. We can write the whole validation plugin without ever having to create an Organization Service, but that just emphasizes the simplicity of that first plugin.

What if we wanted to do something more complicated? For example, how about creating a plugin that, instead of doing that kind of validation, would have to query some data from the configuration entity in

order to put it to the description field? Or what if we wanted to read the error message from the configuration entity so it would not be hard-coded?

We will re-write our first plugin for this scenario shortly, but, for now, let's discuss a few other things related to the Organization Service.

First of all, how do you create an instance of that service? Remember we are talking about the plugins, and we have a serviceProvide parameter there:

That's the parameter we are using to get access to the context, but we can also use it to get an instance of IOrganizationService:

```
IOrganizationServiceFactory serviceFactory =
(IOrganizationServiceFactory)serviceProvider.GetService(typeof(IOrganizationServiceFactory));
OrganizationService service = serviceFactory.CreateOrganizationService(context.UserId);
```

Once we have an instance of IOrganizationService, there is a bunch of useful methods available to us:

https://msdn.microsoft.com/en-us/library/microsoft.xrm.sdk.iorganizationservice.aspx

Methods

	Name	Description
∉	Associate(String, Guid, Relationship, EntityReferenceCollection)	Creates a link between records.
≡	Create(Entity)	Creates a record.
≡	Delete(String, Guid)	Deletes a record.
≡	Disassociate(String, Guid, Relationship, EntityReferenceCollection)	Deletes a link between records.

≡	Execute(OrganizationRequest)	Executes a message in the form of a request, and returns a response.
≡	Retrieve(String, Guid, ColumnSet)	Retrieves a record.
⊕	RetrieveMultiple(QueryBase)	Retrieves a collection of records.
≡	<u>Update(Entity)</u>	Updates an existing record.

This is all for your reference so far – we will look at some of those methods later.

14. Check Point 3

- We just discussed plugin execution context
- Organization Service (even if we did not do much with it yet)
- We have updated our first plugin to fix an error



What's Next?

Basically, we are still laying out the foundation for what we'll be doing later, and there are still a few more concepts to discuss before we can really start writing more advanced plugins. At the same time, we are at the point where you should be able to write a simple plugin of your own, and that's what you are going to do right now.

15. Quiz

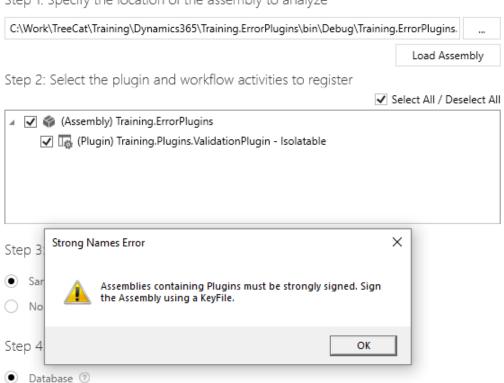
a) Have a look at this plugin code - do you see any problems there?

```
using System;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;
namespace Training.Plugins
{
    public class ValidationPlugin : IPlugin
        public void Execute(IServiceProvider serviceProvider)
            IPluginExecutionContext context = (IPluginExecutionContext)
                 serviceProvider.GetService(typeof(IPluginExecutionContext));
            Entity entity = (Entity)context.InputParameters["Target"];
             if (entity.Contains("ita_name"))
                 string name = (string)entity["ita_firstname"];
                 if ("Test".Equals(name))
                 {
                   throw new InvalidPluginExecutionException("Cannot use this name");
                 }
             }
        }
    }
}
```

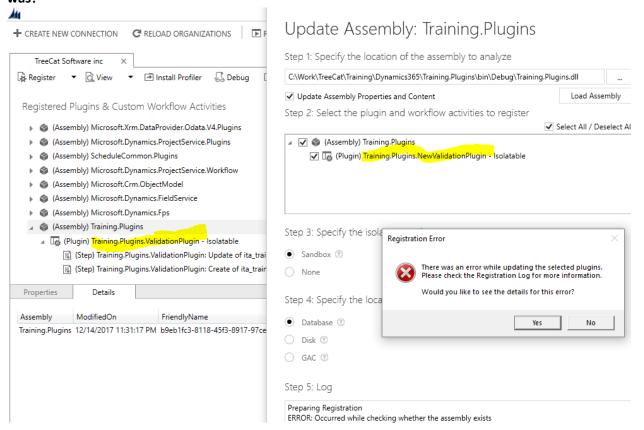
b) You have compiled a plugin assembly, and you are trying to register it in Dynamics. You are getting the error below – what do you still need to do?

Register New Assembly

Step 1: Specify the location of the assembly to analyze



c) You made a change in the original validation plugin code – it still compiles, but you are getting an error when trying to update the assembly in Dynamics. Can you figure out what the change was?



And here are the detais:

Unhandled Exception:

System. Service Model. Fault Exception `1 [[Microsoft. Xrm. Sdk. Organization Service Fault, All Control of the Control of Control

Microsoft. Xrm. Sdk, Version = 9.0.0.0, Culture = neutral, Public Key Token = 31bf3856 ad364 e35]]:

Plug-in assembly does not contain the required types or assembly content cannot be updated.

Detail: <OrganizationServiceFault xmlns="http://schemas.microsoft.com/xrm/2011/Contracts" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">

- <ActivityId>8902d82b-ecd0-424d-9c33-84f383836f56</ActivityId>
- <ErrorCode>-2147204725</ErrorCode>
- <ErrorDetails xmlns:a="http://schemas.datacontract.org/2004/07/System.Collections.Generic"
 />
- <Message>Plug-in assembly does not contain the required types or assembly content cannot be updated.
- <Timestamp>2017-12-15T02:26:50.6430335Z</Timestamp>
- <ExceptionRetriable>false</ExceptionRetriable>
- <ExceptionSource i:nil="true" />
- <InnerFault>

```
<ActivityId>8902d82b-ecd0-424d-9c33-84f383836f56</ActivityId>
  <ErrorCode>-2147204725</ErrorCode>
  <ErrorDetails
xmlns:a="http://schemas.datacontract.org/2004/07/System.Collections.Generic" />
  <Message>Plug-in assembly does not contain the required types or assembly content cannot
be updated.</Message>
  <Timestamp>2017-12-15T02:26:50.6430335Z</Timestamp>
  <ExceptionRetriable>false</ExceptionRetriable>
  <ExceptionSource i:nil="true" />
  <InnerFault i:nil="true" />
  <OriginalException i:nil="true" />
  <TraceText i:nil="true" />
 <OriginalException i:nil="true" />
 <TraceText i:nil="true" />
</OrganizationServiceFault>
```

Server stack trace:

at System.ServiceModel.Channels.ServiceChannel.HandleReply(ProxyOperationRuntime operation, ProxyRpc& rpc)

at System.ServiceModel.Channels.ServiceChannel.Call(String action, Boolean oneway, ProxyOperationRuntime operation, Object[] ins, Object[] outs, TimeSpan timeout) at System.ServiceModel.Channels.ServiceChannelProxy.InvokeService(IMethodCallMessage methodCall, ProxyOperationRuntime operation)

at System.ServiceModel.Channels.ServiceChannelProxy.Invoke(IMessage message)

Exception rethrown at [0]:

at Microsoft.Crm.Tools.Libraries.RegistrationHelper.UpdateAssembly(CrmOrganization org, String pathToAssembly, CrmPluginAssembly assembly, PluginType[] type) at Microsoft.Crm.Tools.AssemblyRegistration.PluginRegistrationViewModel.btnregisterClick()

16. Exercise

Scenario:

The training solution you imported to Dynamics earlier contains an entity called

ita_trainingconfiguration

You need to make sure that, for any create or update of that entity, ita_notestallowedmessage attribute has some data (other than null).

Can you add a new plugin to our current solution, and, in that plugin, implement the validations? You can use the plugin we just wrote as a sample

Hints:

- Add a new class
- Copy-paste code from the original plugin
- Modify validation code
- Change the error message
- Use Plugin Registration tool to update the assembly
- Use Plugin Registration tool to register Update/Create steps on the ita_trainingconfiguration entity
- Finally, test your work

17. Check Point 4

Q&A

What's Next?

There are a few more concepts we need to discuss now. Entities, entity references, attributes, various attribute types. We did not talk about all that yet, but this is something you need to become familiar with. Let's fill those gaps asap.

18. Entity, EntityReference, Attribute

Entities and fields are at the core of Dynamics. Account entity, contact entity, case entity; name field, description field, primary contact field - all that should sound familiar.

Not surprisingly, when developing for Dynamics 365, we are still working with entities and fields (although, we tend to call them attributes).

An entity is represented by the Microsoft.Xrm.Sdk.Entity SDK class in the plugins, so, to create an Entity object, you can do this:

var entity = new Entity("account");

Now here is a question. In the Entity class, there is an Id property:



What if, in Dynamics, you had an account with the following Id:

11f23b4b-64b1-489c-a2ed-f96f2f7cd14f

And what if, in the plugin code, you wrote this:

```
var entity = new Entity("account");
entity.ld = Guid.Parse("11f23b4b-64b1-489c-a2ed-f96f2f7cd14f");
var name = (string)entity["name"];
```

Assuming everything is correct with the syntax, can you guess why this code would still produce a runtime error?

The answer is that, when creating an entity in-memory using "new Entity(...)", you are not, really, loading any data from the back-end. You are creating an in-memory object that does not know anything about how an entity with that Id looks like in the database.

If you wanted to load that account from the database, you'd have to use the Organization Service:

```
IOrganizationServiceFactory serviceFactory =
  (IOrganizationServiceFactory)serviceProvider.GetService(typeof(IOrganizationServiceFactory));
IOrganizationService service = serviceFactory.CreateOrganizationService(context.UserId);
Var entity = service.Retrieve("account", Guid.Parse("11f23b4b-64b1-489c-a2ed-f96f2f7cd14f");"), new ColumnSet(true));
```

We will discuss some of those Organization Service methods later, but what you need to realize now is that the entity object you may have in your code is not necessarily synchronized with the database. Actually, there might be no corresponding record in the database at all.

You can easily do this:

```
var entity = new Entity("account");
entity["name"] = "test";
```

Without setting any Id at all.

With that out of the way, how can you access the attributes/fields in the plugin code?

I think you should already know the answer – but, if you don't yet, have a look at these examples:

- a) entity["<attribute name>"] = value;
- b) value = (attribute_type)entity["<attribute_name>"];
- c) value = entity.GetAttributeValue<attribute_type>("<attribute_name>");

Finally, what is an EntityReference? You can think of it as of an object that holds as little information as possible to still be able to identify an entity in the system:

Properties

Name	Description
ExtensionData	Gets or sets the structure that contains extra data.
Id	Gets or sets the ID of the record.
KeyAttributes	Gets or sets the key attributes.
LogicalName	Gets or sets the logical name of the entity.
Name	Gets or sets the value of the primary attribute of the entity.
RowVersion	Gets or sets the row version.

You will not find attribute values there, but you will find LogicalName and Id. For example, it can be an account with some Id. Or a contact with another Id.

As you keep working with the plugins, you will find that a number of Organization Service requests are utilizing EntityReferences. For example, you can create an AssignRequest like this:

There is no reason you would really need to pass any attributes to this kind of request since all it needs is a couple of Entity References to identify two entities: the target of the assignment, and the user/team to which that entity will be assigned.

19. Plugin step attribute - first plugin fine tuning

Remember how we had to add some conditions to our validation plugin so it stops failing when "ita_name" attribute is null or missing?

Here is that code, again:

There is another way to ensure that the attribute is there. Although, I should ask you a question first.. In the example above, ita_name may be modified once for every 100 updates, so 99% of the time the plugin minght really be doing nothing. It would only make sense to have some mechanism that would allow us to tell Dynamics that we only want the plugin to run when ita_name attribute is updated.

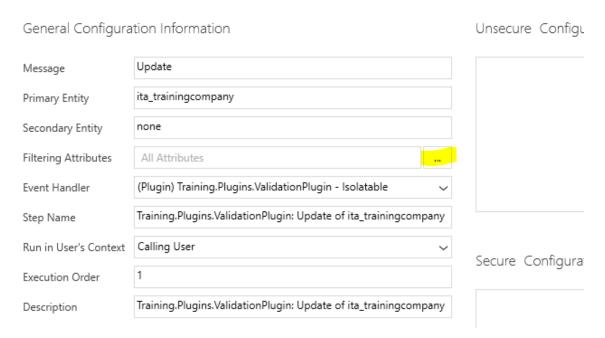
Let's return to the Plugin Registration Tool and select the "Update" step:

Registered Plugins & Custom Workflow Activities

- (Assembly) Microsoft.Xrm.DataProvider.Odata.V4.Plugins
- (Assembly) Microsoft.Dynamics.ProjectService.Plugins
- (Assembly) ScheduleCommon.Plugins
- (Assembly) Microsoft.Dynamics.ProjectService.Workflow
- (Assembly) Microsoft.Crm.ObjectModel
- (Assembly) Microsoft.Dynamics.FieldService
- (Assembly) Microsoft.Dynamics.Fps
- (Assembly) Training.Plugins
 - (Plugin) Training.Plugins.ValidationPlugin Isolatable
 - (Step) Training.Plugins.ValidationPlugin: Update of ita_trainingcompany
 - (Step) Training.Plugins.ValidationPlugin: Create of ita_trainingcompany

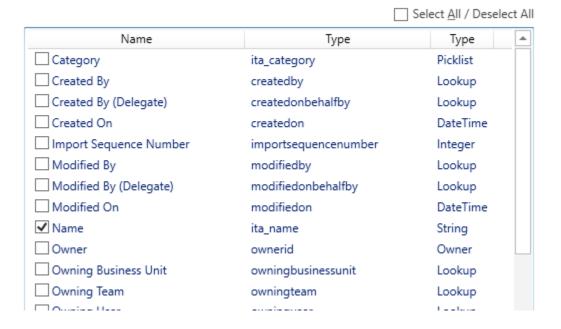
Double click on that step top open step properties window:

Update Existing Step



And, then, click on that highlighted button to open Filtering Attributes window:

Select Attributes



Make sure only Name attribute is selected, click "OK":

Update Existing Step

General Configuration Information

Message	Update	
Primary Entity	ita_trainingcompany	
Secondary Entity	none	
Filtering Attributes	ita_name	
Event Handler	(Plugin) Training.Plugins.ValidationPlugin - Isolatable	~
Step Name	Training.Plugins.ValidationPlugin: Update of ita_trainingcompany	
Run in User's Context	Calling User	~

Click "Update the step".

And that's it - the plugin will not run anymore if any other attribute but the "Name" is updated.

You won't always be using this functionality, but it's a good practice to use it when you can. On the one hand, this will improve performance. On the other hand, this will also allow you to make some assumptions in the code about which attributes are being updated.

Now let's try doing the same for the "Create" step – here is what you'll see there:

Update Existing Step

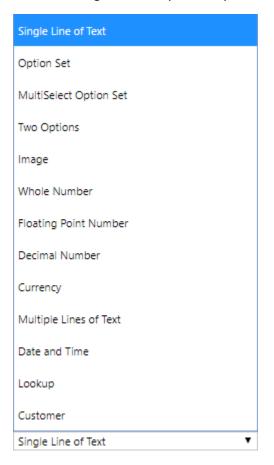
General Configuration Information

Message	Create
Primary Entity	ita_trainingcompany
Secondary Entity	none
Filtering Attributes	Message does not support Filtered Attributes
Event Handler	(Plugin) Training.Plugins.ValidationPlugin - Isolatable
Step Name	Training.Plugins.ValidationPlugin: Create of ita_trainingcompany

Just keep this in mind. When you register your plugin on the Create message, it will run for every Create – you cannot use filtering attributes there.

20. Working with different attribute types

When creating a field in Dynamics, you can choose from many different field types:



For each of those, there is a corresponding type in the SDK assembles which you can use to work with such fields in the plugin code. In all of the examples before, I used a text field:

```
string name = (string)entity["ita_name"];
```

For a text field, no special treatment is needed. It's a "string".

Let's have a look at the other field types and their corresponding .NET classes, though.

Single Line of Text

```
//GET
string name = (string)entity["ita_name"];
//SET
entity["ita_name"] = "test";
```

Multiple Lines of Text

```
//GET
string name = (string)entity["ita_name"];
//SET
entity["ita_name"] = "test";
```

Option Set

```
//GET
var selection = (OptionSetValue)entity["ita_selection"]
int selectedValue = selection.Value;

//SET
var selection = new OptionSetValue("10000000");
entity["ita_selection"] = selection;
```

Two Options

```
//GET
var selected = (bool)entity["ita_twooptions"]
//SET
entity["ita_selection"] = true;
```

Whole Number

```
//GET
int value = (int)entity["ita_wholenumber"]
//SET
entity["ita_wholenumber"] = value;
```

Floating Vlaue

```
//GET
float value = (float)entity["ita_floatingnumber "]
//SET
entity["ita_floatingnumber"] = value;
```

Decimal

```
//GET
decimal value = (decimal)entity["ita_decimalnumber "]
//SET
entity["ita_decimalnumber"] = value;
```

Date and Time

```
//GET
DateTime value = (DateTime)entity["ita_datetime"]
//SET
entity["ita_datetime"] = value;
```

Currency

```
//GET
Money value = (Money)entity["ita_money"]

//SET
Money value = new Money(1000);
entity["ita_money"] = value;
```

21. Check Point 5

- We have discussed 3 concepts/classes you need to be familiar with: Entity, EntityReference, Attribute
- We have also discussed how to fine tune your plugins by specifying which attribute the plugins should be triggered by
- We have also looked at how you can work with different attribute types in the plugins to get and/or set attribute values



What's Next?

In the next part, I will describe sample customization scenario which will require a few different plugins. While implementing those plugins, we will discsuss some of the other topics we have not touched so far.

22. Sample Customization Requirements

- a) When a priority field is updated on the training company, that field value should be pushed to the priority field on all related company contacts. We will need to develop a plugin and register it on update of the training company
 - I will walk you through the implementaiton of this plugin
- b) On the configuration entity, we have credit limits per rating (A/B/C) We need a plugin to validate that we don't exceed the limit when updating credig limit field on the training company records
 - I will walk you through the implementaiton of this plugin as well
- c) We also need a plugin to ensure that we can only link 1 credit limit per credit rating to each configuration entity
 - You will have 30 minutes to work on this plugin yourself. We will have Q & A after that. If the plugin is not ready, you will have 30 more minutes, and, then, we will review everyone's implementation.

TRAINING CONFIGURATION: INFORMATION First = ■ General *Name First No Test Allowed test message Message **Credit Limits** Credit Limit Credit Rating ↑ Α \$300,000.00 В \$150,000.00 C \$100,000.00



23. Plugin #1: Pushing priority field updates to the related company contacts

There are different ways to organize individual plugins in your project – we are going to have a plugin per entity for the sample customization scenario, and we will use "if" conditions in the plugin to determine what exactly the plugin needs to do.

Basically, it's the brute force approach. You will pick up other styles or find your own as you keep dveloping the plugins.

So, let's do it step by step:

- Create a new C# class in the original Training. Plugins project, and call it TrainingCompanyPlugin.cs
- Copy the content of the ValidationPlugin.cs to your new class, make sure you change the class name to TrainingCompanyPlugin:

```
□using System;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;

□namespace Training.Plugins
{
□ public class TrainingCompanyPlugin : IPlugin
□ public void Execute(IServiceProvider serviceProvider)
{
```

- At the beginning of the plugin, we will need to make sure ita_priority field is being updated. Here is how we can do that:

- If it is being updated, we can simply copy new value to all the contacts. Although, question is, how do we get those contacts?

We will look into it shortly, but, for now, let's compile the project and register the plugin in Dynamics.. Although, there is one other thing which I usually do when I want to make sure the plugin is working – let's raise an error inside that "if":

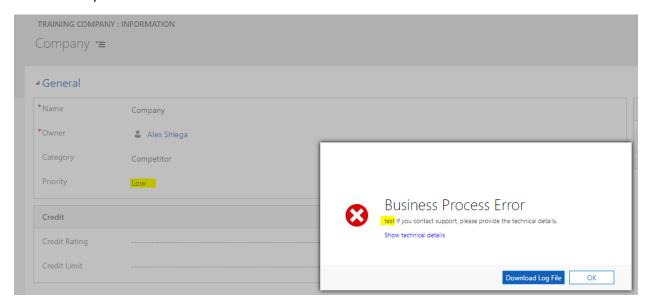
```
Entity entity = (Entity)context.InputParameters["Target"];
if (entity.Contains("ita_priority"))
{
    throw new InvalidPluginExecutionException("test");
}
```

Exercise: With that in place, let's do an exercise.

Let's compile the project and register this plugin on Update of the ita_trainingcompany entity. Don't forget to specify the triggering attributes. If you do it correctly, you should see an error message next time you try updating "Priority" field on the training company entity.

It should not take longer than 15 minutes to get this done, so try to figure it out. After that, I'll walk you through the process on my screen.

Here is what you should see as a result:



Now that we have registered the plugin, let's write some useful code to replace that exception.

First, you will need to know how to load all the associated contacts, and this is where we need to talk about the OrganizationService in more details.

24. Plugin #1: Using Queries with OrganizationService

We did discuss, earlier, how to get an instance of IOrganizationService in the plugin code, so let's replace the "if" block we have there now with this:

```
if (entity.Contains("ita_priority"))
{
    IOrganizationServiceFactory serviceFactory =
    (IOrganizationServiceFactory)serviceProvider.GetService(typeof(IOrganizationServiceFactory));
    IOrganizationService service = serviceFactory.CreateOrganizationService(context.UserId);
}
```

There are two methods in the IOrganizationService interface that you can use to retrieve entities from Dynamics:

≡	Retrieve(String, Guid, ColumnSet)	Retrieves a record.
≡♦	RetrieveMultiple(QueryBase)	Retrieves a collection of records.

The first one (Retrieve) will retrieve a single entity from Dynamics – you'll need to pass entity logical name, entity Id, and the list of columns. For example, it might be something like this:

```
service.Retrieve("account", accountId, new ColumnSet(true));
or
service.Retrieve("account", accountId, new ColumnSet("name"));
```

The second one (RetrieveMultiple) will retrieve a collection of entities according to the Query you will pass to this method.

So, in a nutshell, here is how the code might look like for RetrieveMultiple:

```
QueryExpression qe;
//Create the query expression
var result = service.RetrieveMultiple(qe);
foreach(var e in result.Entities)
{
    //Do what you need to do with each entity in the result set
}
```

That's rather straightforward, but how do we create a query?

There are two different types of queries in the SDK – you can use a QueryExpression, or you can use a FetchExpression.

The difference between them is that, when using a FetchExpression, you are working with FetchXml. For example:

The easiest way to build FetchXml is by using the Advanced Find – there is Download Fetch Xml option there:



Once you have downloaded the xml, you may need to adjust it to your needs, but, usually, it's just minor tweaks. For example, in the xml above, I had to replace the ID of that "Test" company with the ID of the Company entity being updated in the pugin, and, also, I removed all the attributes I did not need.. as well as sorting.

Once you have that, you still need to update every contact in the result set:

```
foreach(var e in result.Entities)
{
    Entity updatedContact = new Entity(e.LogicalName);
    updatedContact.Id = e.Id;
    updatedContact["ita_priority"] = entity["ita_priority"];
    service.Update(updatedContact);
}
```

Although, you might probably do it like this instead, right?

```
e["ita_priority"] = entity["ita_priority"];
service.Update(updatedContact);
```

So why creating a new in-memory etity object?

The reason is that, if you call update on the entity retrieved from the query, there might be some other attributes, and, even though you won't be changing them, they will still be sent back to the server

through that *service.Update* call. Remember how we can trigger a plugin on some attributes? If those attributes are included in the call, all such plugins and/or workflows will, actually, trigger. Even though the attributes will not, really, be updated.

That's why, even though in this example we might probably simplify the code by using the entities retrieved from the query, it's a good practice to create a new entity object specifically for the update so you can have better control over which attributes will be updated.

Here is the complete code for this plugin (using FetchXml):

```
using System;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;
namespace Training.Plugins
    public class TrainingCompanyPlugin : IPlugin
        public void Execute(IServiceProvider serviceProvider)
            IPluginExecutionContext context = (IPluginExecutionContext)
                serviceProvider.GetService(typeof(IPluginExecutionContext));
            Entity entity = (Entity)context.InputParameters["Target"];
            if (entity.Contains("ita_priority"))
                IOrganizationServiceFactory serviceFactory =
(IOrganizationServiceFactory)serviceProvider.GetService(typeof(IOrganizationServiceFactory));
                IOrganizationService service = serviceFactory.CreateOrganizationService(context.UserId);
                string fetchXml =
                    @"<fetch version=""1.0"" output-format=""xml-platform"" mapping=""logical""
distinct=""false"">
                        <entity name=""ita_trainingcontact"">
                          <attribute name=""ita_trainingcontactid"" />
                          <filter type=""and"">
                            <condition attribute=""ita_company"" operator=""eq"" value=""{0}"" />
                          </filter>
                        </entity>
                    </fetch>";
                fetchXml = string.Format(fetchXml, entity.Id);
                var qe = new FetchExpression(fetchXml);
                var result = service.RetrieveMultiple(qe);
                foreach(var e in result.Entities)
                    Entity updatedContact = new Entity(e.LogicalName);
                    updatedContact.Id = e.Id;
                    updatedContact["ita_priority"] = entity["ita_priority"];
                    service.Update(updatedContact);
                }
           }
       }
   }
```

25. Record Count, Paging, and Total Record Count

Since we are talking about FetchXml queries, let's discuss a couple of features you may need to know about.

Just for the sake of learning, let's imagine that we wanted to query 10 contacts only. Can we do that? And, if yes, can we also get the total number of contacts? And, finally, once we have those first 10, how do we get the following 10, and, then, to the 10 after that, and so on?

There are a few attributes you can add to your fetchxml:

```
<fetch version="1.0" count="..." page="..." paging-cookie="..." returntotalrecordcount="true" output-
format="xml-platform" mapping="logical" distinct="false">
```

If you ever need to find out what attributes you can use with Fetch, you can find complete fetchXml schema definition here:

https://msdn.microsoft.com/en-us/library/gg309405.aspx

But, right now, we just need to discuss those 3 attributes highlighted above.

"Count" and "Page" attributes are working together (although, you can use "count" independently). If you use 10 for "count" and 5 for "page", you will get 10 records from the 5th page of the resultset.

"Paging-cookie" is a performance optimization attribute that can, according to MSDN, imrove performance for very large datasets:

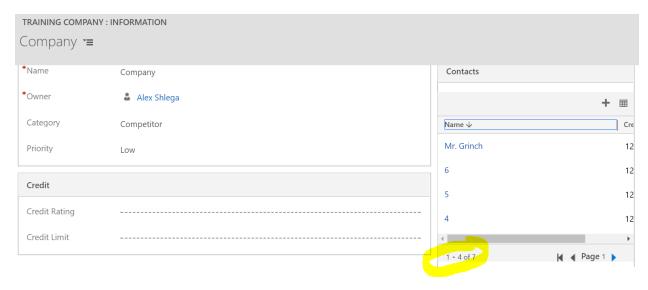
The paging cookie is a performance feature that makes paging in the application faster for very large datasets.

https://msdn.microsoft.com/en-us/library/gg309717.aspx

Finally, returntotal record count is an optional attribute that instructs the Organization Service that you want to know how many records are available in total. Keep in mind that if the number is greater than 5000, you will keep seeing 5000 there. Even if there are hundreds of thousands of records.

Let's try all of that quickly.

First, let's set up a company with 7 contacts:



In the plugin we just created, let's replace FetchXml expression with this:

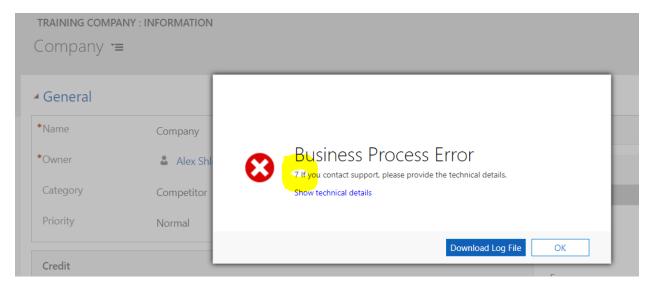
```
string fetchXml =
                       @"<fetch version=""1.0""
                             count=""5""
                             page=""{1}""
                             paging-cookie=""{2}""
                             returntotalrecordcount=""true""
                             output-format=""xml-platform""
                             mapping=""logical""
                             distinct=""false"">
                           <entity name=""ita_trainingcontact"">
  <attribute name=""ita_trainingcontactid"" />
                              <filter type=""and"">
                                <condition attribute=""ita_company"" operator=""eq""</pre>
value=""{0}"" />
                             </filter>
                           </entity>
                       </fetch>";
```

Now let's also refactor the plugin so it starts using page numbers, count, etc.

```
using System;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;
namespace Training.Plugins
    public class TrainingCompanyPlugin : IPlugin
        public EntityCollection RetrieveContacts(IOrganizationService service, Guid companyId, int page,
string pagingCookie)
       {
            if (pagingCookie != null && pagingCookie != "")
                pagingCookie = pagingCookie.Replace("\"", "'").Replace(">", ">").Replace("<", "&lt;");</pre>
            string fetchXml =
                    @"<fetch version=""1.0""
                          count=""5""
                          page=""{1}""
                          paging-cookie=""{2}""
                          returntotalrecordcount=""true""
                          output-format=""xml-platform""
                          mapping=""logical""
                          distinct=""false"">
                        <entity name=""ita_trainingcontact"">
                          <attribute name=""ita_trainingcontactid"" />
                          <filter type=""and"">
                            <condition attribute=""ita_company"" operator=""eq"" value=""{0}"" />
                          </filter>
                        </entity>
                    </fetch>";
            fetchXml = string.Format(fetchXml, companyId, page, pagingCookie);
            var ge = new FetchExpression(fetchXml);
            var result = service.RetrieveMultiple(qe);
            return result;
       }
        public void Execute(IServiceProvider serviceProvider)
            IPluginExecutionContext context = (IPluginExecutionContext)
                serviceProvider.GetService(typeof(IPluginExecutionContext));
            Entity entity = (Entity)context.InputParameters["Target"];
            if (entity.Contains("ita_priority"))
            {
                IOrganizationServiceFactory serviceFactory =
(IOrganizationServiceFactory)serviceProvider.GetService(typeof(IOrganizationServiceFactory));
                IOrganizationService service = serviceFactory.CreateOrganizationService(context.UserId);
                int pageNumber = 1;
                string pagingCookie = "";
                EntityCollection result = null;
                do
                {
                    result = RetrieveContacts(service, entity.Id, pageNumber, pagingCookie);
                    throw new InvalidPluginExecutionException(result.TotalRecordCount.ToString());
                    foreach (var e in result.Entities)
                    {
                        Entity updatedContact = new Entity(e.LogicalName);
                        updatedContact.Id = e.Id;
                        updatedContact["ita_priority"] = entity["ita_priority"];
                        service.Update(updatedContact);
                    pagingCookie = result.PagingCookie;
                    pageNumber++;
                } while (result.MoreRecords);
           }
       }
   }
}
```

In that code above, notice the line in red. The one where an InvalidPluginExecutionException is raised. This is one of the techniques I use all the time when I want to see what's happening in the plugin without having to load a plugin profiler etc.

Re-build the pugin and register it in Dynamics, then try updating priority field on the company that has those 7 associated contacts, and you should see this error message:



So yes, there are 7 contacts, as expected.

You should keep something in mind, though. In order to find out total records count, Dynamics has to run a more complex query. As the number of records in your system starts growing, you may have to weigh in the pros and const of knowing that number – sometimes, you just need to page through the results, so don't ask Dynamics for the total record count when you don't really need to know it.

Also, notice that call to result. MoreRecords. When you are using paging, that's the easiest way to determine if the page you were looking at is not the last page and there are still records left.

Exercise: Now remove the exception line, rebuild the plugin, register it in Dynamics, and do the same test again

You should now see how priority field on the contacts becomes synchronized with the priority field on the company record whenever you update the latter.

26. Exercise

What we have missed to cover so far:

Imagine there is a company record already, and we are adding a new contact to that company.

Can you come up with a plugin solution to ensure that our new contact will be assigned the same priority as its parent company?

27. Check Point 6

- We have just covered our first customization requirement
- Along the way, we have discussed FetchXml and how we can use it with RetrieveMultiple to run queries against Dynamics
- We have also discussed how paging works and what we can do with record counts



What's Next?

There is an alternative to FetchXml, which is called QueryExpression. One is not necessarily better than the other, but you should be able to work with both. We will re-write the same plugin using Query Expression next. But, first, let's do an exercise.

28. Plugin #1: Using a QueryExpression

When using Query Expressions in Dynamics plugins, we basically have two options:

- We can use Fetch Expressions
- And we can use Query Expressions

Both are inherited from the QueryBase, though:

And both are offering the same functionality – you can even use special SDK requests to convert one to another:

___public sealed class FetchExpression : QueryBase

https://msdn.microsoft.com/en-us/library/hh547457.aspx

We can easily rewrite our RetrieveContacts method using a QueryExpression:

And this example shows exactly what the difference is. When using a QueryExpression, there is no XML anymore. The whole query is created using standard programming interfaces.. basically, it's an object on which you can access certain methods and properties to configure that query.

The main advantage of this approach is that, instead of working with XML and/or strings, you can work with an actual object. That's, also, a disadvantage, though, since FetchXml is natural for Dynamics in the sense that you can go to Advanced Find and download it.

So, depending on your development needs, you may prefer one of these two techniques. Personally, I usually find FetchXml a little more straightforward since I don't have to keep in mind how QueryExpression will be converted to Fetch (and it will be, since, in the end, Dynamics needs FetchXml to run a query). On the other hand, when there is a lot of preparation involved, QueryExpression may be easier to work with.

BTW, from the code above, you can see how we have access to the Count, TotalRecordCount, Page, and PagingCookie.

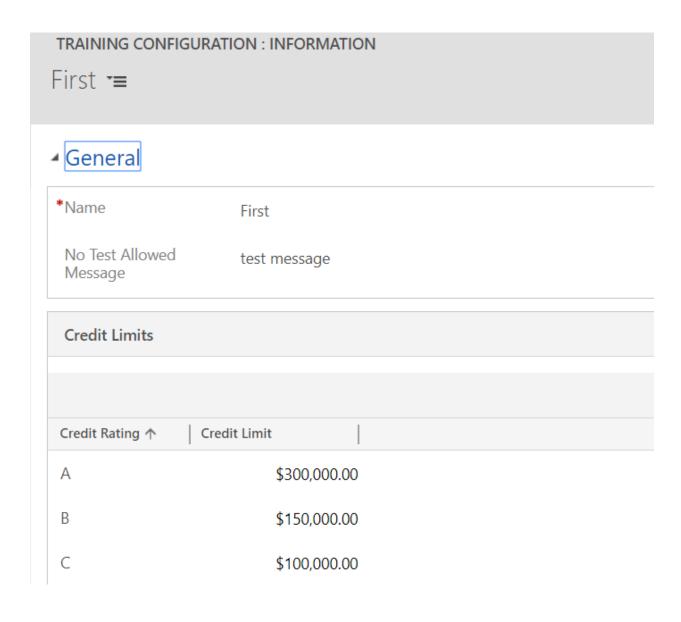
Either way, let's have another exercise

- Rename the FetchXml version of this plugin to TrainingCompanyPluginFetchVersion.cs
- Create a Copy of that plugin class
- Make sure the copy is called TrainingCompanyPlugin.cs (and the class is called TrainingCompanyPlugin)
- Use QueryExpression version of the RetrieveContacts function (See above)
- Build the assembly and use plugin registration tool to update Dynamics
- Run the same test as before confirm that priority field value from the company is propagated to the associated contact records

29. Plugin #2: Validating the credit limit

Before we continue, let me remind you of the requirement:

On the configuration entity, we have credit limits per rating (A/B/C) We need a plugin to validate that we don't exceed the limit when updating credig limit field on the training company records



TRAINING COMPANY: INFORMATION Company Name Company Owner Alex Shlega Category Competitor Priority Low Credit Credit Credit Limit

29.1 Setting it up

Let's think about what the plugin is going to do first:

- It will need to be registered on both Update and Create of the Training Company entity
- If will only need to do the validations if both Credit Rating and Credit Limit fields on the company record are populated (but keep in mind they don't have to be updated at the same time)
- Based on the Credit Rating of the company, the plugin will need to retrieve maximum credit limit for that rating from the configuration entity
- Then, if the credit limit on the company record is greater than the maximum retrieve above, the plugin will need to throw an error

We might create a new class for this plugin, but, on the other hand, we might reuse the same plugin class we created before (TrainingCompanyPlugin). That's what I will do this time.

First, let's re-factor the code. In the code below, I have created two methods:

- ExecutePriorityUpdate
- ExecuteCreditLimitValidation

The first method is doing exactly what the plugin used to do before

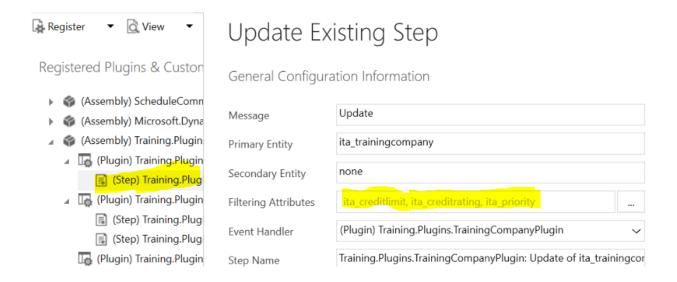
The second method will be doing the validations

Then I have updated the Execute method to call one/both of those other ones, depending on which attributes are being updated in the entity.

```
using System;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;
namespace Training.Plugins
    public class TrainingCompanyPlugin : IPlugin
       public EntityCollection RetrieveContacts(IOrganizationService service, Guid companyId, int page,
string pagingCookie)
            QueryExpression qe = new QueryExpression("ita_trainingcontact");
            qe.Criteria.AddCondition(new ConditionExpression("ita_company", ConditionOperator.Equal,
companyId));
            qe.PageInfo.PageNumber = page;
            qe.PageInfo.Count = 5;
            qe.PageInfo.PagingCookie = pagingCookie;
            qe.PageInfo.ReturnTotalRecordCount = true;
            var result = service.RetrieveMultiple(qe);
            return result;
        }
        public void ExecuteCreditLimitValidation(IPluginExecutionContext context, IOrganizationService
service, Entity entity)
```

```
}
       public void ExecutePriorityUpdate(IPluginExecutionContext context, IOrganizationService service,
Entity entity)
        {
            int pageNumber = 1;
            string pagingCookie = "";
            EntityCollection result = null;
            do
            {
                result = RetrieveContacts(service, entity.Id, pageNumber, pagingCookie);
                foreach (var e in result.Entities)
                    Entity updatedContact = new Entity(e.LogicalName);
                    updatedContact.Id = e.Id;
                    updatedContact["ita_priority"] = entity["ita_priority"];
                    service.Update(updatedContact);
                pagingCookie = result.PagingCookie;
                //throw new InvalidPluginExecutionException(pagingCookie);
                pageNumber++;
            } while (result.MoreRecords);
       }
        public void Execute(IServiceProvider serviceProvider)
            IPluginExecutionContext context = (IPluginExecutionContext)
                serviceProvider.GetService(typeof(IPluginExecutionContext));
            IOrganizationService service = null;
            IOrganizationServiceFactory serviceFactory = null;
            Entity entity = (Entity)context.InputParameters["Target"];
            bool creditRatingCheck = entity.Contains("ita_creditrating") && entity["ita_creditrating"] !=
null
                || entity.Contains("ita_creditlimit") && entity["ita_creditlimit"] != null;
            bool priorityUpdate = entity.Contains("ita_priority");
            if(creditRatingCheck || priorityUpdate)
            {
                serviceFactory =
(IOrganizationServiceFactory)serviceProvider.GetService(typeof(IOrganizationServiceFactory));
                service = serviceFactory.CreateOrganizationService(context.UserId);
            }
            if(creditRatingCheck)ExecuteCreditLimitValidation(context, service, entity);
            if (priorityUpdate) ExecutePriorityUpdate(context, service, entity);
       }
   }
}
```

Once again, we can now build the dll and register it in Dynamics using Plugin Registration tool. We will need to add Credit Limit and Credit Rating attributes to the Update step:

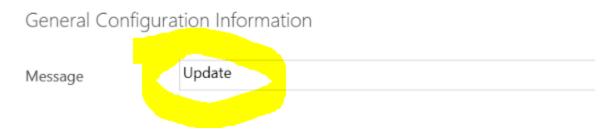


And, then, we will have a bit of a problem. There is no "Create" step yet, so we should definitely create a step. However, do we want the plugin to do anything with the priority in that step? We don't, so we need to add a condition to the plugin code:

```
bool priorityUpdate = context.MessageName == "Update" && entity.Contains("ita_priority");
```

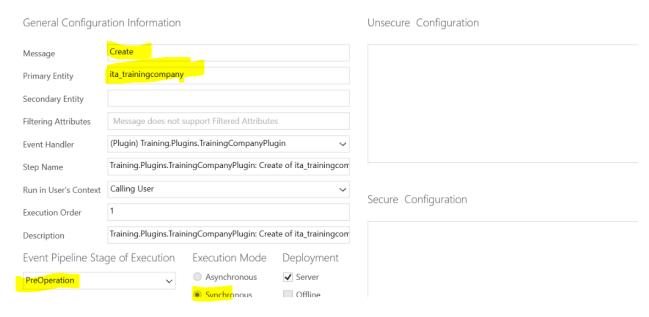
The message name you see there is the same message name you see in the Plugin Registration tools when registering a step:

Update Existing Step



Now we need to rebuild the dll and to update it in Dynamics again. Then we can add a new step:

Register New Step



We have everything set up and configured, but the plugin is not going to do anything yet – there is still no validation code there. We are going to add it shortly, but we need to talk about the concept of Prelmages and PostImages first.

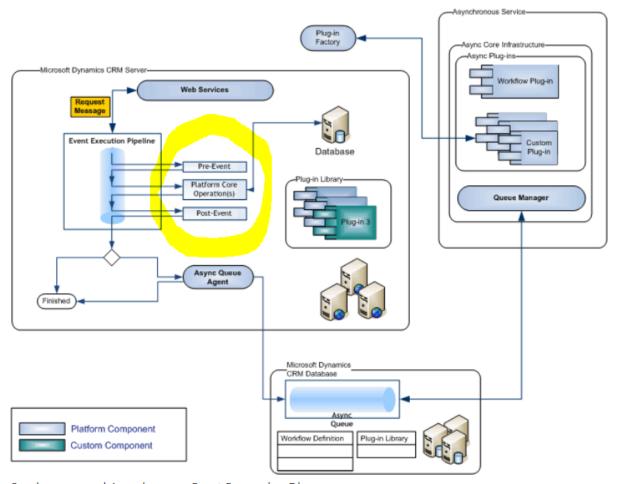
29.2 Pre-Operation, Post-Operation, Pre-Image and Post-Image

And, in order to talk about the Pre/Post Images, we'll need to discuss the "Event execution pipeline":

Architecture and related components

Ĵ.

The following figure illustrates the overall architecture of the Microsoft Dynamics 365 platform with respect to both synchronous and asynchronous event processing.



Synchronous and Asynchronous Event Processing Diagram

https://msdn.microsoft.com/en-us/library/gg327941.aspx

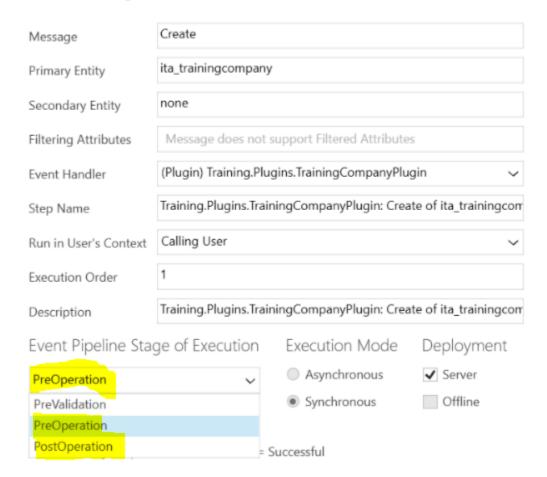
It it looks somewhat confusing, focus on the highlighted area for now. In short, here is the idea:

- A Request Message is the data package you send to Dynamics through an SDK/WebAPI call
- An Event is when the data has been modified in Dynamics or when it's been queried from Dynamics
- You can register your plugin steps in Pre-Event (which is before the data has been updated/queried) or in Post-Event (after it has been updated/queried)

In the plugin registration tool, this is called "Pre-Operation" and "Post-Operation":

Update Existing Step

General Configuration Information



Below are some examples of the plugins you may want to register in Pre-Operation:

- You may want to assign default credit limit to every new training company. If you do it in the
 pre-event, you can simply populate credit limit attribute in the plugin "Target" without
 having to call an extra "Update"
- You may want to do custom duplicate validations before a new training company is created

And below are some examples of the plugins you may want to register in Post-Operation:

- Once a new training company record has been created/updated, you may want to update aggregated "total credit limit" field on the configuration entity

Once a new training company record has been created/updated, you may want to add
default training contact records to that company. You cannot do it in the Pre-Operation
since the training company record does not exist in the database at that time, so there is
nothing to link those default contacts to. In the Post-Operation, that's not a problem

Why are we talking about all this? Remember how we are using the following code to see which attributes are being modified?

```
Entity entity = (Entity)context.InputParameters["Target"];
if (entity.Contains("ita_priority"))
{
}
```

This is because "Target" entity will only have the attributes being modified (for update) or populated (for create) on the client side.

Come to think of it, that creates a bit of a problem for the credit limit validation plugin we wanted to implement since how are we supposed to deal with the following sequence of events:

- A new training company will be created, and "Credit Rating" attribute will be populated as part of the "Create" request
- And, then, "Credit Limit" attribute will be updated in a subsequent update request

In the plugin code, we won't be able to get both attributes from the "Target" entity for either of those requests – only one attribute will be available every time.

This is where we need to talk about Pre-Images and Post-Images, and they should probably make more sense in the context of "Pre-Operation" and "Post-Operation":

Transaction

Pre-Operation

(Data updates are not in the database yet)

- Target
- Pre Image(for Update)

Post-Operation

(Now data updates are in the database)

- Target
- Pre Image(for Update)
- Post Image

Target Entity: only contains the attributes being updated

A "Pre-Image" is what our Target entity looked like before the update.

A "Post-Image" is what it looks like after the update. Normally, a Post-Image should give us exactly the same set of attributes/values that we would get by running a "Retrieve" request on the Target entity Id in the Post-Operation.

By utilizing pre images and post images, we can do some interesting comparisons, but what's important for our validation plugin is that we need to start using pre image in combination with Target to access both "Credit Limit" and "Credit Rating" fields at the same time.

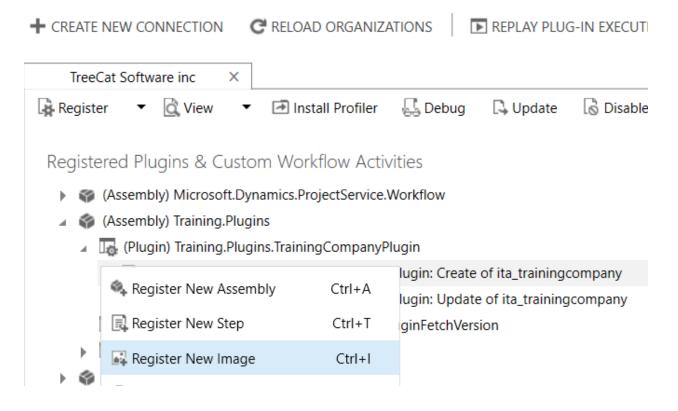
So, basically, the idea is:

- We will still be using "Target" to verify if either of those attributes was submitted as part of the request
- If an attribute we are interested in is missing from the "Target", we will need to get that attribute from PreImage

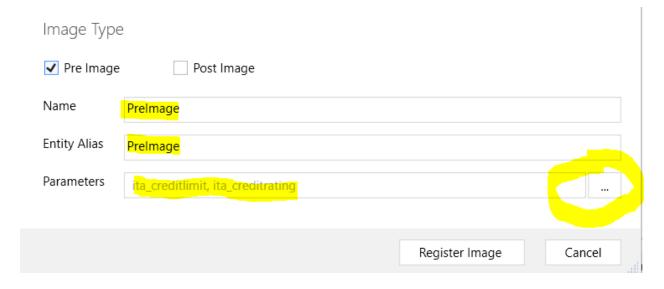
Sure, another option might be to simply register our validation plugin in PostOperation, in which case we might use PostImage to get access to the attributes all the times. But it's better to do validations as early as possible anyway, so we will stick to the PreOperation.

29.3 Adding a Pre Image

As you have probably guessed, images are added to the SDK Message Processing Steps using the Plugin Registration Tool. Open the tool, select the step, right click on that step, and choose "Register New Image" command:



We have two steps (Create and Update), so let's try it with the "Create" step first.



If you try it like that, you will see an error message:





Unable to register the Image due to an error.

Would you like to see the details for this error?



The problem is that there can be no "Pre Image" for the "Create" message since the entity does not exist, yet, when "Create" happens. If you choose to see the error details above, you will see exactly that:

Error Message - Detail



In that sense, Plugin Registration is relying on the SDK for error handling, so it will let you go through the steps of setting up an image, and, then, it will try to create that image in Dynamics. That's when Dynamics will throw an error, and that's when you'll finally realize there is a problem with what you were trying to do. Still, most of that is common sense – as in this case, for example, you just need to keep in mind that "Create" message does not support PreImages.

So, let's do exactly the same, but for the Update step:

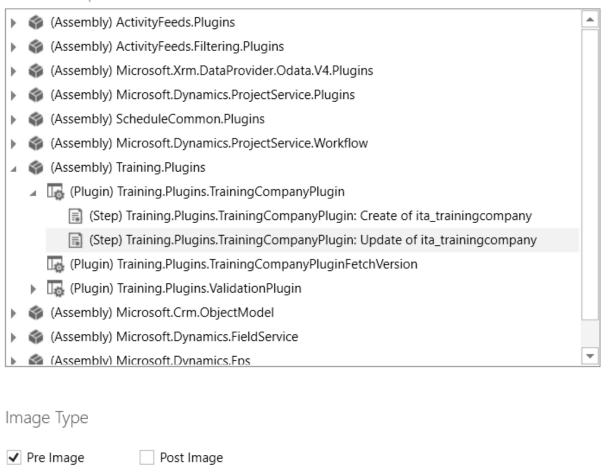
Register New Image

Select a Step

Name

Entity Alias

Parameters



Click "Register Image", and you'll see the image there:

ita_creditlimit, ita_creditrating

Prelmage

Prelmage

Registered Plugins & Custom Workflow Activities

- (Assembly) Microsoft.Dynamics.ProjectService.Workflow
- (Assembly) Training.Plugins
 - III. (Plugin) Training.Plugins.TrainingCompanyPlugin
 - [(Step) Training.Plugins.TrainingCompanyPlugin: Create of ita_trainingcompany



- (Step) Training.Plugins.TrainingCompanyPlugin: Update of ita_trainingcompany
 - (Image) PreImage (ita_creditlimit,ita_creditrating) Updated Entity

(DL. sia) Tarinia - DL. sia - Tarinia - Carra - DL. sia Fatabi/assia

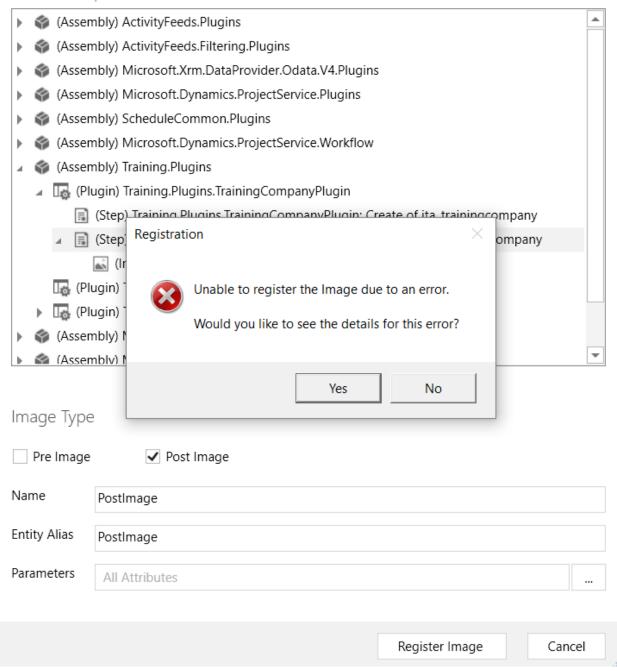
Note: if you wanted to register a Post Image, the process would be pretty much the same. You would have to choose a different Image Type, but that would be the only difference.

Although, here is a quick question.

Can you guess why I am not allowed to register a Post Image on the same "Update" step on which we just registered a Pre Image (hint: it does not have anything to do with multiple images being registered there)

Register New Image

Select a Step



If you don't know the answer, try registering a Post Image on that step in your environment, and, once you get an error message, click "Yes" to see the error details. You will find the explanations there.

29.4 Implementing the validations

Now that we have a PreImage, how do we use it in the code?

From the developer standpoint, images are entities. Literally:

Entity preImage = context.PreEntityImages["PreImage"];

In the plugin context, there are two collections:

PreEntityImages

And

PostEntityImages

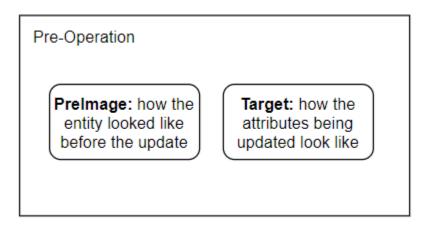
Depending on the type of the image, you will find it in one of those two collections. For the image name, you will be using "Entity Alias" name from the image registration:

Image Type		
✓ Pre Image Post Image		
Name	Prelmage	
Entity Alias	Preimage	
Parameters	ita_creditlimit, ita_creditrating	

Once you have an image, you can access all the attributes of that image in the same way you would do it with the "Target", for example:

preImage["ita_creditrating"]
preImage["ita_creditlimit"]
etc.

However, here is what we have to work with now:



There is a PreImage that has all the attributes as they looked like before the update started, and there is a plugin "Target" that has all the attributes being updated.

We are interested in two attributes: ita_creditlimit and ita_creditrating, but our Target entity won't always have both of those attributes – it may have 1 (any ono) or two. If there is only one, the other one might be available form the PreImage (if that attribute had ever been populated before).

To handle this scenario, I would usually add this kind of function to the plugin:

```
public T GetAttribute<T>(Entity entity, Entity preImage, string attributeName)
{
   if (entity != null && entity.Contains(attributeName)) return (T)entity[attributeName];
   else if (preImage != null && preImage.Contains(attributeName)) return (T)preImage[attributeName];
   else return default(T);
}
```

Not if I have a pre image and a target, I can pass them to that function together with the attribute name, and, as a result, I will either get null (more eactly, default(T)), or the value of that attribute.. this code will check the Target first, and, if the attribute is not there, it will try the pre image.

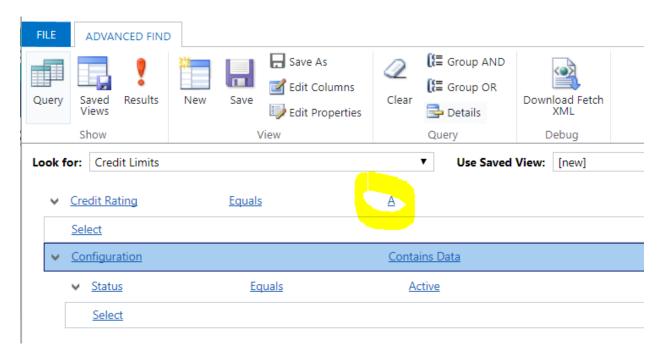
Now I can re-write ExecuteCreditLimitValidation like this:

```
public void ExecuteCreditLimitValidation(IPluginExecutionContext context, IOrganizationService service,
Entity entity)
{
    Entity preImage = context.PreEntityImages.Contains("PreImage") ? context.PreEntityImages["PreImage"] :
null;
    var creditLimit = GetAttribute<Money>(entity, preImage, "ita_creditlimit");
    var creditRating = GetAttribute<OptionSetValue>(entity, preImage, "ita_creditrating");
    if(creditLimit != null && creditRating != null)
    {
        //DO THE VALIDATIONS
    }
}
```

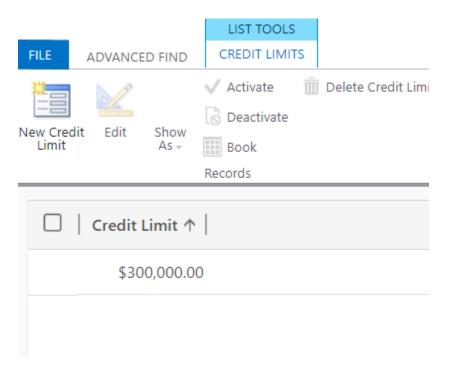
This code also takes care of a few other things:

- "Create" step won't have a pre-image
- There is no need to validate anything if the Credit Limit or/and the Credit Rating are not provided

To do the validations, we need to query maximum credit limit for the selected credit rating from the configuration. Which means we need to run this kind of query:



Instead of the "A" rating, we will need to use company's current rating. As a result, we will get a "Credit Limit" record which will give us the credit limit for that rating.



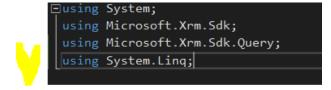
Here is the FetchXml behind that query:

```
<fetch version="1.0" output-format="xml-platform" mapping="logical" distinct="false">
 <entity name="ita_creditlimit">
  <attribute name="ita_creditlimitid" />
  <attribute name="ita_creditlimit" />
  <order attribute="ita creditlimit" descending="false" />
  <filter type="and">
   <condition attribute="ital creditrating" operator="eq" value="455000000" />
  </filter>
  link-entity name="ita_trainingconfiguration" from="ita_trainingconfigurationid"
to="ita configuration" link-type="inner" alias="ac">
   <filter type="and">
    <condition attribute="statecode" operator="eg" value="0" />
   </filter>
  </link-entity>
 </entity>
</fetch>
```

We will need to run that FetchXml from the plugin, and we will need to replace 455000000 with the proper value. Should be easy to do (but you may need to review what we did with the QueryExpressions while creating Plugin #1).

To do this, let's do a couple of things:

1. Add System.Ling to the *using* part of the plugin class



2. Add the following function to the plugin

```
public Money GetCreditLimit(IOrganizationService service, OptionSetValue creditRating)
    string fetchXml = @"
       <fetch version=""1.0"" output-format=""xml-platform"" mapping=""logical"" distinct=""false"">
          <entity name=""ita_creditlimit"">
             <attribute name=""ita_creditlimitid"" />
             <attribute name=""ita_creditlimit"" />
             <order attribute=""ita_creditlimit"" descending=""false"" />
             <filter type=""and"">
                 <condition attribute=""ita_creditrating"" operator=""eq"" value=""{0}"" />
             </filter>
             <link-entity name=""ita_trainingconfiguration"" from=""ita_trainingconfigurationid""</pre>
to=""ita_configuration"" link-type=""inner"" alias=""ac"">
                <filter type=""and"">
                  <condition attribute=""statecode"" operator=""eq"" value=""0"" />
                </filter>
             </link-entity>
           </entity>
        </fetch>
     fetchXml = string.Format(fetchXml, creditRating.Value);
     var qe = new FetchExpression(fetchXml);
```

```
var result = service.RetrieveMultiple(qe).Entities.FirstOrDefault();
if(result != null)
{
    return (Money)result["ita_creditlimit"];
}
else
{
    return null;
}
```

That function is using eactly the same FetchXml we created in the Advanced Find above. It takes a few parameters:

- OrganizationService we need that to run the query
- creditRating of the company being validated that's an OptionSetValue we need there to adjust the conditions

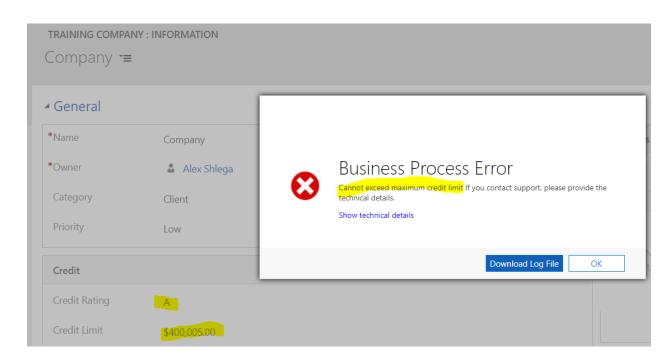
There is one new method we did not discuss so far which often makes it easier to work with the queries when you need to choose only the first record from the list:

```
var result = service.RetrieveMultiple(qe).Entities.FirstOrDefault();
```

If there are no records, this method will return **null**. Otherwise, it will return the first record in the result set. Which means we can use this method in the ExecuteCreditLimitValidation now:

```
public void ExecuteCreditLimitValidation(IPluginExecutionContext context, IOrganizationService service,
Entity entity)
{
    Entity preImage = context.PreEntityImages.Contains("PreImage") ? context.PreEntityImages["PreImage"] :
null;
    var creditLimit = GetAttribute<Money>(entity, preImage, "ita_creditlimit");
    var creditRating = GetAttribute<OptionSetValue>(entity, preImage, "ita_creditrating");
    if (creditLimit != null && creditRating != null)
    {
        var maxLimit = GetCreditLimit(service, creditRating);
        if (maxLimit != null && maxLimit.Value < creditLimit.Value)
        {
            throw new InvalidPluginExecutionException("Cannot exceed maximum credit limit");
        }
    }
}</pre>
```

Update the code, register the plugin, and do a test. Can you see an error message?



Try using various combinations of Credit Rating / Credit Limit fields to see how the plugin is intercepting those calls now. Try creating a new company, try updating an existing company - make sure it's all working as expected.

- We have discussed QueryExpressions
- We have discussed Pre-Operation, Post-Operation, Pre-Images, and Post-Images
- We have also developed a plugin to validate company credit limit



What's Next?

There is still requirement #3. That'll be the exercise.

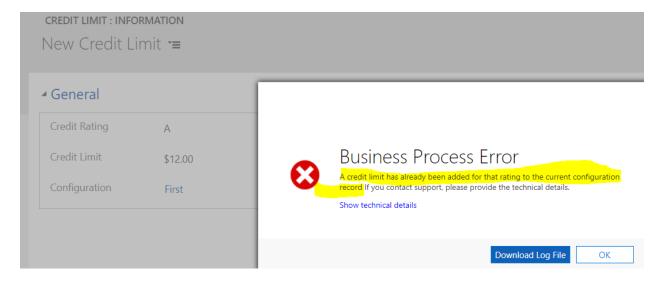
31 Exercise: Plugin #3

We need a plugin to ensure that we can only link 1 credit limit per credit rating to each configuration entity.

Here is what you need to do:

- Add a new plugin class to the project we've been working with. Call the plugin CreditLimitPlugin
- Using code samples from the plugins we have developed so far, create this plugin. Don't
 worry too much about structuring your code in the best possible way at this point. It's not
 the purpose of this exercise
- Register the plugin in Dynamics and test it

As a result, you should be able to see an error message like this one whenever you try adding a duplicate credit limit:



In fact, by copy-pasting code from two other plugins this one can be up and running in less than 30 minutes. But it's not a speed test – take those 30 minutes and see how far you can get it and what questions show up.

Just wanted to remind you of a few things:

- 1. In the plugin code, you can use EntityReference attribute type just the same way you do it with OptionSetValues etc
- 2. If you were using a QueryExpression, you might create a condition this way:

```
qe.Criteria.AddCondition(new ConditionExpression("ita_configuration",
ConditionOperator.Equal, configuration.Id));
```

Notice how ita_configuration is a lookup, and, in the code above, a condition is added to the query expression to filter the query by the value of that lookup attribute

3. The plugin should work for "Create" and "Update" messages

Below is the code I used for this plugin, but you may have come up with something else. It should, likely, be the same idea:

```
using System;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;
using System.Ling;
namespace Training.Plugins
  public class CreditLimitPlugin: IPlugin
    public T GetAttribute<T>(Entity entity, Entity preImage, string attributeName)
      if (entity != null && entity.Contains(attributeName)) return (T)entity[attributeName];
      else if (preImage != null && preImage.Contains(attributeName)) return (T)preImage[attributeName];
      else return default(T);
    public void ExecuteCreditLimitDUplicateValidation(IPluginExecutionContext context, IOrganizationService service, Entity entity)
      Entity preImage = context.PreEntityImages.Contains("PreImage") ? context.PreEntityImages["PreImage"] : null;
      var configuration = GetAttribute<EntityReference>(entity, preImage, "ita_configuration");
      var creditRating = GetAttribute<OptionSetValue>(entity, preImage, "ita_creditrating");
      if (configuration != null && creditRating != null)
        QueryExpression qe = new QueryExpression("ita_creditlimit");
        qe.Criteria.AddCondition(new ConditionExpression("ita creditlimitid", ConditionOperator.NotEqual, entity.Id));
        qe.Criteria.AddCondition(new ConditionExpression("ita_configuration", ConditionOperator.Equal, configuration.Id));
        qe. Criteria. Add Condition (new Condition Expression ("ita_creditrating", Condition Operator. Equal, credit Rating. Value)); \\
        if(service.RetrieveMultiple(qe).Entities.FirstOrDefault() != null)
```

```
throw new InvalidPluginExecutionException("A credit limit has already been added for that rating to the current configuration
record");
    public void Execute(IServiceProvider serviceProvider)
    {
       IPluginExecutionContext context = (IPluginExecutionContext)
         serviceProvider.GetService(typeof(IPluginExecutionContext));
      IOrganizationService service = null;
      IOrganizationServiceFactory serviceFactory = null;
       Entity entity = (Entity)context.InputParameters["Target"];
       bool duplicateCheck = entity.Contains("ita_configuration") && entity["ita_configuration"] != null
         || entity.Contains("ita_creditrating") && entity["ita_creditrating"] != null;
       if(duplicateCheck)
         service Factory = (IOrganization Service Factory) service Provider. Get Service (type of (IOrganization Service Factory)); \\
         service = serviceFactory.CreateOrganizationService(context.UserId);
         \label{lem:exact context} Execute Credit Limit DUplicate Validation (context, service, entity);
    }
 }
```

33 Remaining Theory

There are a few more concepts you need to know about when developing plugins for Dynamics. We did not discuss them so far since they would not fit well into the examples I used above, and I did not want to overcomplicate those examples. But, with all the work we've done, it would be a shame to walk out of this course without even touching them.

Here is what we still need to discuss:

- Sandboxed plugins vs Non-Sandboxed plugins
- Synchronous pluvins vs Asynchronous plugins
- Pre-Validation stage
- Plugin execution order
- Plugins recursion
- Transactions (yes, there is still this "DB" thing hidden somewhere)
- Early Bound vs Late Bound development
- Debugging the plugins

34.1 Sandboxed plugins vs Non-Sandboxed plugins

While working with the plugin registration tool, you may have noticed the setting called "the isolation mode":

Update Assembly: Training.Plugins

Step 1: Specify the location of the assembly to analyze

Load Assembly Step 2: Select the plugin and workflow activities to register ✓ Select All / Deselect All (Assembly) Training.Plugins Plugin (Plugin) Training.Plugins.TrainingCompanyPlugin (Step) Training.Plugins.TrainingCompanyPlugin: Update of ita_trainingcompany (Plugin) Training.Plugins.ValidationPlugin (Step) Training.Plugins.ValidationPlugin: Update of ita_trainingcompany 🦪 🖹 (Stan) Training Pluging Validation Plugin: Croate of ita training company Step 3: Specify the isolation mode



Step 4: Specify the location where the assembly should be stored



That selection is a legacy of the on-premise version – you can only use "Sandbox" in the online environment. It's important to understand what it means, though:

Microsoft Dynamics 365 (online & on-premises) support the execution of pluq-ins and custom workflow activities in an isolated environment. In this isolated environment, also known as a sandbox, a plug-in or custom activity can make use of the full power of the Microsoft Dynamics 365 SDK to access the organization web service. Access to the file system, system event log, certain network protocols, registry, and more is prevented in the sandbox. However, sandbox plug-ins and custom activities do have access to external endpoints like Azure Cloud Services.

https://msdn.microsoft.com/en-us/library/gg334752.aspx

So, in short, when in the Sandbox mode, your code is not running in full-trust. Which means that sandbox-ed plugins have some limitations, and here are those that (I think) are the most important ones:

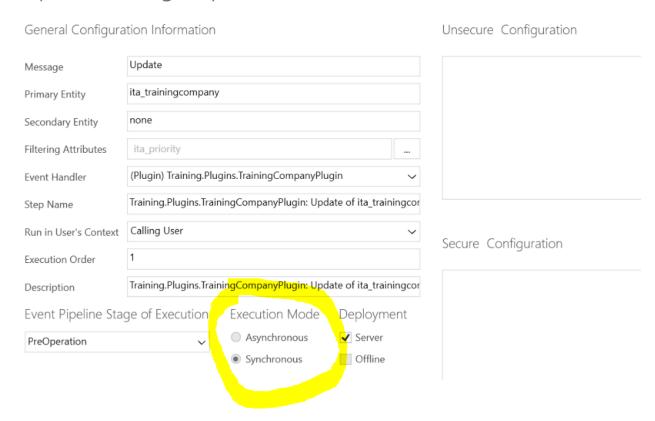
- Only the HTTP and HTTPS protocols are allowed.
- Access to localhost (loopback) is not permitted.
- IP addresses cannot be used. You must use a named web address that requires DNS name resolution.
- There is a 2 minutes execution time limit on the plugins

If you are using on-premise version, you may still have a choice. However, for better compatibility of your plugins it's probably better to keep them registered in the SandBox anyway. What you may need to keep in mind is that SandBox-ed plugins will be running in the dedicated SandBox service process, and it's possible that such a service will be running on a server that's different from the Dynamics web server.

34.1 Synchronous vs Asynchronous

Another configuration setting we have not discussed yet can be configured on each of the steps, and it applies to both of the isolation modes:

Update Existing Step



We used to have synchronous steps everywhere so far; however, sometimes it may be better to use asynchronous steps.

The difference between those two execution mode is exactly the same that you can experience when you are using background workflow vs real-time workfows. A synchronouse plugin will start right away, and the whole event won't finish until that synchronous plugin has finished. If that plugin is running in the pre/post operation, it will also be running in the same database transaction as the main event.. In other words, imagine you are updating training company name and you have a synchronous plugin there to validate training company name (that's what we did at the beginning of thise course, right?)

Since it's a synchronous step, the validation will kick in, it will throw an exception, and the whole operation will rollback. Since that step will be running in the main transaction.

Now if you make it asynchronous, the plugin will still run. However, it will run AFTER the main transaction has finished. Dynamics will schedule it, and, then, will run it.. And you will even see an error message in the System Jobs, but it won't be affecting the main transaction anymore.

So, when choosing the execution mode, think about the following:

- Do you want that step to run in the main transaction? (are there any date updates? Do you want them to happen before the main transaction has finished? Are there any errors? Are they supposed to interrupt the main transaction? Etc)
- How long will it normally take to run that step? Will the users notice it if you make it a synchronous step? Would it be better, then, to run it asynchronously unless (just keep in mind that, if that step will be making some data updates, those updates will happen asynchronously as well)

You will probably find that most of the times you will prefer to make all those steps synchronous. Error handling and data updates will be easier and more straightforward in that mode. However, when it comes to the client-side performance, you should think twice since asynchronous plugins will usually be a better choice.

34.2 Transactions

Below is a copy-paste from MSDN:

Plug-ins may or may not execute within the database transaction of the Microsoft Dynamics 365 platform. Whether a plug-in is part of the transaction is dependent on how the message request is processed by the pipeline. You can check if the plug-in is executing in-transaction by reading the IsInTransaction 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 may be part of the transaction.

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

https://msdn.microsoft.com/en-us/library/gg327941.aspx#bkmk DatabaseTransactions

What is stage 20? It's Pre-Operation.

What is stage 40? It's Post-Operation.

Stage 10 is Pre-Validation that we still need to talk about – will do it shortly.

However, just to give you some examples of how it works, let's go over a few scenarios:

There is a synchronous plugin registered on Update in Pre-Operation, and that plugin throws an error

All the data changes that may have happened in that and other plugins by that time will be rolled back.

There is a synchronous plugin registered on Update in Post-Operation, and that plugin throws an error

All the data changes that may have happened in the plugin by that time will be rolled back. Also, all the changes that may have happened in any Pre-Operation or Post-Operation plugins by that time will be rolled back.

There is an asynchronous plugin registered on Update in Pre-Operation, and that plugin throws an error

Well, actually, you can't register an asynchronous plugin in Pre-Operation. It has to be synchronous for that.

<u>There is an asynchronous plugin registered on Update in Post-Operation, and that plugin throws an error</u>

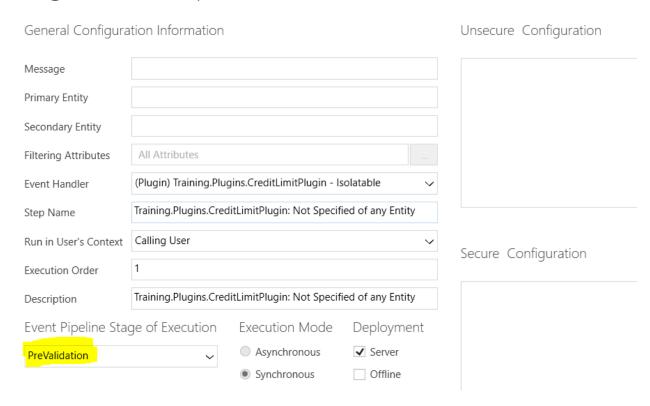
By the time that plugin runs, main transaction will have been committed already. So, any synchronouse data updates won't be affected. Any changes that the plugin itself has caused(and which are synchronous for that plugin) will be rolled back.

Just keep in mind that what's a-synchronous from one standpoint is still synchronous from another. For example, an asynchronous plugin might update a record. There might be another sycnrhonous plugin registered on that update. If an error happened in the asynchronous plugin after that update, it would have rolled back all the changes applied in that second synchronous plugin, too.

34.3 Pre-Validation

Quite frankly, Pre-Validation is a very unusual stage. What may be making it even more confusing in the newest versions of the Plugin Registration tool is that it's the default selection for the stage:

Register New Step



Never register a step in pre-validation. Unless you know why you are doing it.

Keep in mind that a plugin step registered in Pre-Validation will have a couple of unique characteristics:

- It may run outside of database transaction
- Pre-Validation occurs prior to security checks

You may end up with all sorts of incorrect scenarios there. For example, if an update happens in the prevalidation step, that update might be stored in the database even if the user did not have permissions to run the original operation.

On the other hand, that kind of update/create/delete/any other data operation that should be rollback is exactly why you may want to have the step registered in pre-validation.

In other words, there are legitimate scenarios where you may want to take advantage of those two unique characteristics of the Pre-Validation step, but always make sure you have a reason for that.

BTW, you can't register an a-synchronous Pre-Validation step, since a-synchronous steps are always "delayed" steps. And, since pre-validation has to happen before anything else, it can't be delayed:

Pre-Validation

DB Transaction, Synchronous

Pre-Operation

Platform Core (Main Operation, Internal)

Post-Operation

Post-Operation, Async

34.4 Plugin Execution Order

You can register many different plugins on the same message/stage, but what if it matters in which order those plugins should run?

For example, the first plugin may have to do some validations. The second plugin, once the validations have passed, is supposed to do some data updates. Etc.

If you were developing all of those plugins right now, you might probably control the order of those individual operations by organizing your plugin code properly. However, what if it's an existing system, there is a third-party plugin there, and you need to add your own validation plugin to run before that third-party plugin? You can't change the code, so you have to do something else.

This is where you can use execution order setting – it's available on every message processing step:

Register New Step

General Configuration Information

Message						
Primary Entity						
Secondary Entity						
Filtering Attributes	All Attributes					
Event Handler	(Plugin) Training.Plugins.CreditLimitPlugin - Isolatable					
Step Name	Training.Plugins.CreditLimitPlugin: Not Specified of any Entity					
Run in User's Context	Calling User					
Execution Order	1					
Description	Training.Plugins.CreditLimitPlugin: Not Specified of any Entity					
Event Pipeline Stage of Execution		Execution Mode	Deployment			
PreValidation ~		Asynchronous	✓ Server			
		Synchronous	Offline			

It may be a bit confusing, though. Execution Order helps Dynamics decide in which order the plugins should run, but it's the last setting Dynamics will consider. In other words, if the step is supposed to

start at all for that message/stage/attribute/etc, Dynamics will look at the execution order to determine the order in which the steps will start. Otherwise, execution order will make no difference.

For example, if one step is registered in Pre-Operation and another step is registered in Post-Operation, Pre-Operation step will always start first no matter how you assing the orders.

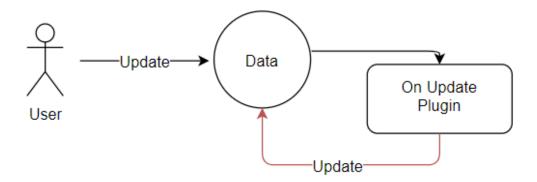
That said, it's easy to update execution order on any step – just open Plugin Registration Tool, open step properties, and change the number in the execution order field.

Two different steps can have the same order. By default, every step will have 1 in that field. If you have two steps with the same order, I'd make no assumptions about how the plugins will run, then.

Steps with the lower order number will run first, steps with the higher order number will run last.

34.5 Plugins Recursion

What if you register a plugin step on update of some attribute, and, then, update the same attribute in the plugin again?



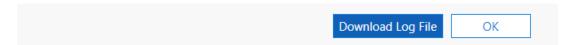
Dynamics can't decide, for you, when is the right time to stop this endless loop, but it will detect the recursion, and, after a few repetitions, will stop it:

Error



This workflow job was canceled because the workflow that started it included an infinite loop. Correct the workflow logic and try again. For information about workflow logic, see Help. If you contact support, please provide the technical details.

Show technical details



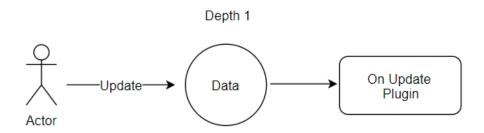
It will do the same in many different situations. It can be one plugin, or a chain of plugins, or a chain of workflows, or a mix of those. They can be synchronous or a-synchronous, they just have to be executing as a result of the same original operation. There are some finer settings (which you can't easily change, especially in the online environments) which control when Dynamics will interfere, but, in the end, as soon as the number of those sequential plugins/workflows reach a certain limit within a pre-defined timeframe, you will get a similar error message.

There are a few optimizations/workarounds available to you:

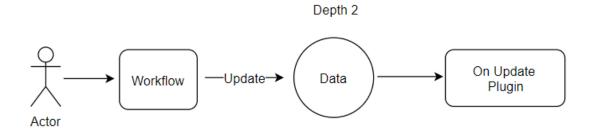
- Always define specific triggering attributes for your steps. Don't just leave "all attributes" there. That way, you will reduce the possibility of occasional recursion, but you won't be able to completely avoid it
- You can use **context.Depth** property which is available in the plugin execution context. It will keep increasing for every new workflow/plugin in the "chain". For every new "update", Depth will start with 1. Then, if, inside the first plugin step you do something that start another plugin, that second plugin will have Depth=2 in the context. And it will keep increasing till it reaches the point where Dynamics will interrupt the process. Although, keep in mind that those have to be sequential events where one event is causing the other. If, for instance, there are two pre-operation steps on Update of the same entity, both reponding to the update of the same attributes, "Depth" will be the same in those two steps(since, from the Depth perspective, they are "parallel".. not sequential). This is why you will often see this condition in the plugins:

if (context.Depth > 1) return;

That works, but that introduces a hidden problem. Imagine that you have a plugin that's doing something on update and that has that kind of condition. You have deployed the plugin, you have testing it by opening the entity in Dynamics, updating a field, and verifying that the plugin kicked in. It is depth 1, so it's all good:



But what if we change it a little?



Now it's, suddenly, Depth 2, and the plugin will not kick in when the update happens from a workflow/another plugin. Which you may or may not discover until such time when

somebody has created a workflow, started to use it, and, then, somebody else pointed out that the data is not consistent anymore

- Do not make the beginner's mistake

```
Never do this:
Entity entity = (Entity)context.InputParameters["Target"];
...
service.Update(entity);
Instead, always create a new in-memory entity, assign the id, and set only the attributes you want to update:
Entity entity = (Entity)context.InputParameters["Target"];
...
Entity updatedEntity = new Entity(entity.LogicalName);
updatedEntity.Id = entity.Id;
updatedEntity["attribute"] = value;
service.Update(updatedEntity);
```

- Try not to call service. Update / service. Create at all – in the pre-operation step, you can use Target to set the attributes. There is no need for a separate Update call, so there is no recursion at all:

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

34.6 Early Bound vs Late Bound

Everywhere above, we have been using late bound code. Basically, later-bound is when we are accessing an attribute like this:

```
string name = (string)entity["name"];
```

This is different from early-bound, when we are accessing an attribute like this:

```
string name = entity.Name;
```

You can use late bound any time, there is no need to any additional perparation steps. It's an extremely flexible way of coding, but, of course, when you are using late-bound, you cannot rely on the compiler to tell you if attribute names are correct.

In order to use early-bound, you have to do some prep work first. But, once you've done that, you can expect the compiler to tell you right away if you are using right attribute names, so you can avoid unnesessary run-time errors. And, besides, if you look at those two lines above, the second example looks cleaner because there are no type conversions there.

That said, personally I have the tendency of using late-bound either way.

Still, how do you get early-bound to work? Apparently, different Dynamics environments will have different entities, and, even the same entities, will have different attributes.

This is why Dynamics provides an SDK utility to generate proxy classes for the specific environment:

https://msdn.microsoft.com/en-us/library/gg327844.aspx

Here is a sample command line – you will need to replace organization url, user name, password, and, also, namespace name and output file name if you want them to be different:

C:\Dev\SDK\Tools\CoreTools\CrmSvcUtil.exe

/url:https://tcdec2017.api.crm3.dynamics.com/XRMServices/2011/Organization.svc /namespace:Training.Plugins /out:Xrm.cs /username:"USERNAME" /password:"PASSWORD"

This will generate early-bound classes for all the entities in your organization, and that's going to be a lot – for example, I just got almost 20 MB file generated for the online trial.

If you wanted to limit that to only a few entities, you migth use the approach suggested by Erik Pool here:

http://erikpool.blogspot.ca/2011/03/filtering-generated-entities-with.html

Once you have the file, add it to your project, and you can start writing the code this way:

```
Account account = new Account();
account.Name = "Test";
account.Id = service.Create(account);
```

This is still going to be translated into late-bound calls behind the scene:

```
public string Name
{
    get
    {
        return this.GetAttributeValue<string>("name");
    }
    set
    {
        this.OnPropertyChanging("Name");
        this.SetAttributeValue("name", value);
        this.OnPropertyChanged("Name");
    }
}
```

But, as far as custom code goes, you are still getting the benefits of early-bound code.

What you need to remember is that you do have to re-generate those early-bound classes every time you make a change in Dynamics. Or, at least, every time you want that change to become available to the early-bound code in your plugins.

Although, you don't have to, since, technically, you can do this:

```
Account account = new Account(); //early bound technique
account["name"] = "Test"; //late bound technique
account.Id = service.Create(account);
```

But that's getting it too far since you'll be mixing two paradigms – not the best practice at all.

35 Debugging the plugins

You have developed a plugin, you deployed it, it went through a few iterations, and, suddenly, your users are starting to complain about the errors coming from that plugin.

What do you do?

Actually, that's a hell of a question because it depends, depends, and depends.

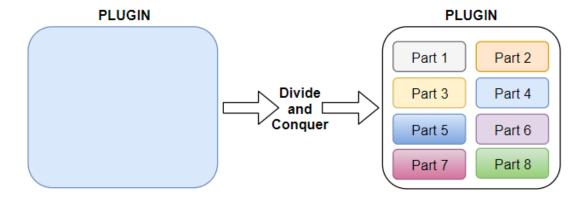
How do you normally do it in the Visual Studio? You start your application under debugger and, then, just keep stepping over the code. In case with the plugins, you are in a very peculiar situation, though, because, more likely than not, you cannot easily attach Visual Studio to the Dynamics process for debugging. If it's an online environment, you just cannot do it at all. If it's an on-premise environment, you may be able to do it, so this is your option 1. You will find step-by-step instructions here:

https://msdn.microsoft.com/en-us/library/gg328574.aspx

If that's not an option, you have 3 other alternatives – neither of those come close to debugging in the Visual Studio, but, normally, one of the/all of them together should be able to help you out:

- You can raise errors from the plugin, and, so, by passing some meaningful info in the error messages, you should be able to find out what's going on in the plugin
- You can use a Tracing Service
- You can use a Plugin Profiler

I'll describe each of those techniques below, but, before we go there, let me suggest a useful plugin development strategy: **Divide and Conquer**



If you think it's more suitable for politics and wars, maybe you just did not do a lot of plugin development yet.

See, your eventual success depends not only on how well you know the SDK, but, also(probably even more), on how quickly you can put your ideas to work. To put it simply, in the absense of standard debugging tools you would save yourself a lot of time if you were able to develop with as little need for debugging as possible. Therefore, develop in small increments, test your changes as soon as you can and as often as you can, make sure you maintain the test scenarios for your plugins, and, ideally, create some kind of automated test framework. When you are doing development in really small increments, it's much easier to find out what's causing a problem just by looking at the recent additions to the code – in many cases, you don't even need to debug anything. This is one of the reasons why .NET developers switching to Dynamics will often find it a bit complicated to debug the plugins – they'll create a big chunk of code hoping to debug it at some point, but it's not that simple in Dynamics.

It may not help that much if you are taking over existing plugins from somebody else. But you can still follow the same approach – assume what's in there works and take it from there in small increments, don't just rewrite everything.

Finally, there is one other option, which is not, really, about "plugin debugging" so much, but which can often yield the results even faster. It is to test parts of your plugin code in a console utility by reusing the plugin code there. If you figure out how to do that, you will have the full power of Visual Studio debugging tools at your disposal.

A lot of concepts we discussed so far are applicable to the Dynamics console applications as well, since both plugins and console applications will be using the same SDK assembly references. There are some differences, of course, since you will have to create an instance of OrganizationService manually, and you will not have plugin context in the console app. If you wanted to explore this option, have a look at the following example:

https://msdn.microsoft.com/ru-ru/library/gg695803(v=crm.7).aspx

35.1	(Coming Soon)	Raising errors:	InvalidPlugin	ExecutionExcep	otion

35.2 (Coming Soon) Tracing Service

35.3 (Coming Soon) Plugin Profiler

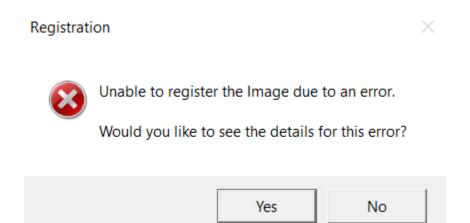
- 36 Quiz
- 36.1 You are registering a plugin step on Create message. Which "images" do you have access to?
 - a) Pre-Image
 - b) Post-Image
 - c) Both
- 36.2 In your update plugin, you just got Target entity from the plugin context, and you have this code there:

entity["new_name"] = "test";

The code runs on update, there are no other conditions in the code, but you don't see "test" in the new_name attribute of the entity even once you have reloaded the browser page.

Why would this be happening? Choose all that applies.

- d) You have registered the step in Pre-Operation, not in Post-Operation
- e) You have registered the step in Post-Operation, not in Pre-Operation
- f) You have configured the step to run asynchronously
- g) You have configured the step to run synchronously
- 36.3 You are adding a Pre Image to the step using the Plugin Registration Tool, and you are getting the error message below:



What is the likely reason for this error?

36.4 You have a QueryExpression, and you want to retrieve the first record from the result:

```
var result = service.RetrieveMultiple(qe);
Is there a simple way to do that?
```

36.5 You have a plugin running on Udate in the Post-Operation. What is the likely problem with this code:

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

36.6 Have a look at the plugin code below – do you see any problems there?

```
using System;
using Microsoft.Xrm.Sdk;
using Microsoft.Xrm.Sdk.Query;
namespace Training.Plugins
    public class ValidationPlugin : IPlugin
        public void Execute(IServiceProvider serviceProvider)
            IPluginExecutionContext context = (IPluginExecutionContext)
                 serviceProvider.GetService(typeof(IPluginExecutionContext));
            Entity entity = (Entity)context.InputParameters["Target"];
             if (entity.Contains("ita_name"))
             {
                 string name = (string)entity["ita_firstname"];
                 if ("Test".Equals(name))
                   throw new InvalidPluginExecutionException("Cannot use this name");
             }
        }
    }
}
```

You have compiled a plugin assembly, and you are trying to register it in Dynamics. You are getting the error below – what do you still need to do?

Register New Assembly

Step 1: Specify the location of the assembly to analyze

