



Universal Converter Plugin Creation Guide

Written by Anthony Daly

Table Of Contents

1	Plugin Introduction	Page 3
1.1	What do I need to create plugins?	
1.2	Plugins, plugins... what exactly are they?	
1.3	What type of plugins can I create?	
1.4	Let's get started!	
2	Conversion Plugin Tutorials	Page 5
2.1	LamaPig 2010	
3	Reference	Page 12
4	Final Word	Page 19

Plugin Introduction

Welcome to the Universal Converter Plugin Creation Guide. In this guide you will find out how to create fully-functional plugins for the award-winning unit conversion program – Universal Converter. However, before you dive into creating your first plugin, make sure you have the following things...

What do I need to create plugins?

To create Universal Converter plugins, and follow the examples in this guide, you should have the things in the following list:

- ➔ Experience with a .NET programming language such as VB.NET or C#
- ➔ Visual Studio or any other development environment (optional)
- ➔ A basic understanding of concepts such as object-orientation, interfaces etc.

Do not get discouraged if you have never written a plugin for any other program before; in this guide, you will find various tutorials which you can follow, as well as links to accompanying source code that you can download and modify to serve as a basis for your own plugins, or run directly¹.

Plugins, plugins... what exactly are they?

Wikipedia describes plugins as: "a set of software components that adds specific capabilities to a larger software application"².

This can mean many things, mostly which are specific to each application, but generally this concept remains the same regardless of the application. Universal Converter has a very specific plugin framework which allows you to do some things and not others. You will learn these things as you progress through this guide.

¹ Demo source code requires Visual Studio 2010 to open

² [Wikipedia](#) article on plugins

What type of plugins can I create?

The Universal Converter plugin system is still very new, and as of yet there is only one type of plugin that you can create: Conversion Plugins.

However, during the rest of 2010 and 2011, a lot of work will be put into expanding this plugin system, such as adding capabilities to add custom actions when converting and window plugins.

- ➡ **Conversion Plugins** – These are plugins which add custom conversions into the program.

Let's get started!

Well, after that introduction into the world of Universal Converter plugins, you're finally ready to try your hand at creating your first plugin! In the next chapter, you will find a tutorial on how to create a fictional conversion plugin:

- 'LamaPig 2010'

For beginners, it is best if you follow the tutorial as set out, but you can modify details if you want to, don't be afraid to let your imagination run wild!

If you would rather know about the methods and functions you can use directly, hop over to the [Reference](#) section.

Please Note: Tutorial code is given in the VB.NET programming language with Visual Studio 2010 as the IDE.

Please scroll to the next page.

Conversion Plugin Tutorials

LamaPig 2010

- For the purpose of this tutorial, in the code, llama will be spelt as lama, just to distinguish the 'unit' from the animal. However, I have decided to keep the 'pig' unit the same...

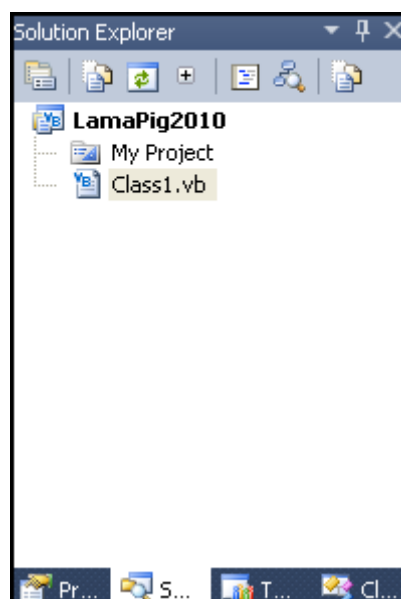
Welcome to the LamaPig 2010 tutorial. As with the other tutorials in this guide, just follow the steps as they are numbered (1,2,3...). Code snippets are marked by their colour and **special font**.

 = information


Tutorial

1. Open Visual Studio and select *'File → New Project'*
2. Select *'Class Library'* and type 'LamaPig2010' as the name of the project.
3. Click *'OK'*

 Visual Studio should create a new project as shown below, with an empty class:



4. Right click on the project node in the Solution Explorer and select *'Add Reference'*.
5. Click *'Browse'* and navigate to the installation directory of Universal Converter. This is normally 'C:\Program Files\Universal Converter\'. Select the 'UCPluginBase.dll' file here and click *'Add Reference'*.

 Now that we have added a reference to the plugin base library, we can start to actually make the plugin itself.


- Save the project -

6. Rename the empty class (Class1.vb) to 'LamaPigPlugin.vb' and double click on this file. This will be the main file of your conversion plugin.
7. Modify the code in this file so it looks the same as below:

```
Imports UCPluginBase
Imports UCPluginBase.Interfaces


<Serializable(> _
Public Class LamaPigPlugin

End Class
```

 The first two lines import the needed namespaces for the plugin. The `<Serializable(>` attribute is needed if the plugin supports saving (more information on this later).


8. Just underneath the line that reads "Public Class LamaPigPlugin", add the following line and hit the enter key:

```
Implements IConversion
```

 Your file should now be populated with empty methods and properties that we must implement. Do not get overwhelmed by the number of these; most of them are just true/false values.


9. We are now going to fill in the methods and properties with some code. First, scroll to the 'AllowsSave' property and modify it so that it looks like the code below:

```
Public ReadOnly Property AllowsSave As Boolean Implements IConversion.AllowsSave
    Get
        Return True
    End Get
End Property
```

 The AllowsSave property specifies whether the conversion allows the user to save their conversion. Normally this should be set to True.


10. Scroll down a bit more to the 'DefineOtherArgs()' method and add the following line in the method body:

```
Return Nothing
```

 For this basic tutorial we don't need any other arguments to be passed to our conversion, so we return nothing.

11. Find the 'DisableInvert' property and add the following line within the 'Get' and 'End Get' keywords.

```
Return False
```

 DisableInvert is normally used when the conversion doesn't work both ways, i.e. $A \rightarrow B$ is okay, but $B \rightarrow A$ is not.

12. Find the 'DisableZeroResultDialogue' property and add the following line within the 'Get' and 'End Get' keywords.

```
Return False
```


 DisableZeroResultDialogue is only used in conversions that use words as output.

13. Find the 'GetConversionInfo()' method. Modify it so it looks like the code below:

```
Public Function GetConversionInfo() As ConvInfo Implements
IConversion.GetConversionInfo

    Dim ci As New ConvInfo
    With ci
        .Common = False
        .ConversionName = Name
        .ConvItems = GetConvertFromList().ToList
        .DateAdded = New Date(2010, 7, 31)
        .Information = "This is a simple plugin example."
    End With

    Return ci
End Function
```


-
-  The code shown above tells Universal Converter what to show when the conversion is selected in the Conversion Information window inside the program. It contains things such as the date it was created (DateAdded) and the name of the conversion (ConversionName).

14. Modify the 'GetConvertFromList()' method to look as such:

```
Public Function GetConvertFromList() As String() Implements
IConversion.GetConvertFromList


    Dim s As New List(Of String)
    s.Add("Lama")
    s.Add("Pig")

    Return s.ToArray
End Function
```

-
-  The GetConvertFromList() function is an important one because it defines what items get put in the 'convert from' list inside the program. In this tutorial, we will add two items – 'Lama' and 'Pig'.


15. Add this line inside the 'GetConvertToList()' method:

```
Return GetConvertFromList()
```

-
-  We are using the same items in the 'convert to' list as in the 'convert from' list.


16. Scroll to the 'GetExtraInfo' property and add the following line within the 'Get' and 'End Get' keywords.

```
Return ""
```

 Only return a value here if you have extra information to show to the user when converting.


17. Scroll to the 'Name' property and add the following line within the 'Get' and 'End Get' keywords.

```
Return "LamaPig 2010"
```

 Pretty self-explanatory, this is the name of the conversion. It has to be less than 22 characters.

18. For the properties 'NeedsInternet', 'OverrideDefaultOpen', 'OverrideDefaultSave', 'OverrideDefaultValidation' and 'PerformValidation' add the following line within the 'Get' and 'End Get' keywords.

```
Return False
```

 These four properties will almost always be set to 'False', however, there are some occasions where you may want them to be set to 'True'. Check out the [Reference](#) section for more information.

Now, I'm guessing that your a bit tired of the words 'return' and 'false' by now. Don't worry there's only one more method that we must modify, and that's the biggy – the 'Convert()' method.

You've probably figured out that we will be converting lamas to pigs. The conversion ration for this (in case you didn't know!) is as such.

1 lama = 5 pigs
1 pig = 1/5 lama

So let's move on and write the code for the Convert method before we deploy our addin and actually use it inside Universal Converter.

19. Find the 'Convert()' method and copy and paste this code into it:

```

Dim convfrom As String = ""
Dim convto As String = ""
Dim amount As Double = 0

'Cast to the correct data types
Try
    convfrom = CStr(convertFrom)
    convto = CStr(convertTo)
    amount = Cdbl(data)
Catch icex As InvalidCastException
    host.ShowInvalidEntryError(Me)
    [error] = True
    Return host.InvalidEntryErrorMessage
Catch ex As Exception
    Return ""
End Try

If convertFrom = "Lama" AndAlso convertTo = "Lama" Then
    Return CStr(amount) + " " + "lamas"
Else If convertFrom = "Lama" AndAlso convertTo = "Pig" Then
    amount *= 5
    Return CStr(amount) + " " + "pigs"
Else If convertFrom = "Pig" AndAlso convertTo = "Pig" Then
    Return CStr(amount) + " " + "pigs"
Else If convertFrom = "Pig" AndAlso convertTo = "Lama" Then
    amount /= 5
    Return CStr(amount) + " " + "lamas"
End If

Return ""

```



What's happening here? Well, first of all, the parameters of type 'Object' are cast to their own assumed types (Strings and Doubles). Then in the 'If' statement, the conversion types are tested and the amount is adjusted accordingly. Finally, the amount is returned as a string (`Return CStr(amount)`) with the units added on.

That's it! You've made your first conversion plugin. There are a few other methods that we could fill in (that are currently blank) but I'll leave that for you to explore.

On the next page you can see the final steps of this tutorial which cover how to actually deploy and use the plugin inside Universal Converter.

Using the plugin

20. Click *'Build'* inside Visual Studio to build the plugin. Make sure the project is set to 'Release Build' and not 'Debug Build'.
21. Navigate to the *'bin/release'* folder of your project and copy the 'LamaPig2010.dll' file.
22. Go to *'My Documents → Universal Converter → Plugins'* and paste the dll here.
23. Finally, change the file extension of the dll to '.ucp'. This enables Universal Converter to recognise it as a plugin.
24. Start up Universal Converter.
25. You should now see 'LamaPig 2010' listed as a conversion! Click on it and test it out!

Thanks for following this tutorial. Hopefully you now feel a bit more confident about creating your own conversions.

You can download full source code for this tutorial from [here](#)









It is advised that you read the [Final Word](#) which gives a few links to some contacts and resources you may like to keep handy.

If you ever need to see what methods and properties are available to you as a Universal Converter Plugin Developer, as well as explanations of them, make sure to visit the [Reference](#) section.

Note: The reference section and this guide as a whole is likely to grow and change pretty quickly, so from time to time, re-download this PDF guide from [here](#).

Reference

File: UCPluginBase.dll

{}	UCPluginBase
	ConvInfo <code>Public Structure ConvInfo</code> Represents a conversion information object with info about the conversion.
	ClearAll <code>Public Sub ClearAll()</code> Clears all the fields.
	Common <code>Public Property Common As Boolean</code> Specifies whether the conversion is common or not.
	ConversionName <code>Public Property ConversionName As String</code> The conversion's name.
	ConvItems <code>Public Property ConvItems As System.Collections.Generic.List(Of String)</code> All available conversion items to display.
	DateAdded <code>Public Property DateAdded As Date</code> The date that the conversion was added.
	Information <code>Public Property Information As String</code> The main body of text for the conversion information.
	New <code>Public Sub New(ByVal ConversionName As String, ByVal DateAdded As Date, ByVal Common As Boolean, ByVal Information As String, ByVal ConvItems As System.Collections.Generic.List(Of String))</code> Creates a new instance of the structure.

Parameters:

ConversionName: The conversion name.

DateAdded: The date the conversion was added.

Common: Whether this conversion is 'common'.
Information: The main body of information.
ConvItems: A list of the items in this conversion.



State

Public NotInheritable Class State

A snapshot of the current application state.



Conversion

Public ReadOnly Property Conversion As UCPluginBase.Interfaces.IConversion
 Specifies the base conversion.



ConvertFrom

Public ReadOnly Property ConvertFrom As Object
 Specifies the base unit to convert from.



ConvertTo

Public ReadOnly Property ConvertTo As Object
 Specifies the base unit to convert to.



Data

Public ReadOnly Property Data As Object
 Specifies the base data.



New

Public Sub New()
 Initializes this class.



New

Public Sub New(ByVal data As Object, ByVal convertFrom As Object, ByVal convertTo As Object, ByVal conversion As UCPluginBase.Interfaces.IConversion, ByVal ParamArray otherParameters As Object())
 Initializes this class.

Parameters:

data: The data.
convertFrom: The object to convert from.
convertTo: The object to convert to.
conversion: The conversion.
otherParameters: Additional parameters to serialize.



Operator <>

Public Shared Operator <>(ByVal s1 As UCPluginBase.State, ByVal s2 As UCPluginBase.State) As Boolean



Operator =

Public Shared Operator =(ByVal s1 As UCPluginBase.State, ByVal s2 As UCPluginBase.State) As Boolean



OtherParameters

`Public ReadOnly Property OtherParameters As Object()`

Specifies the base parameters.

{} UCPluginBase.Interfaces



IConversion

`Public Interface IConversion`

The public interface for conversions.



AfterOpen

`Sub AfterOpen(ByVal state As UCPluginBase.State, ByVal host As UCPluginBase.Interfaces.IHost)`

Allows for overriding the opening state.

Parameters:

state: A State object.

host: The object representing the main application.



AllowsSave

`ReadOnly Property AllowsSave As Boolean`

Specifies whether the conversion allows saving.



BeforeSave

`Sub BeforeSave(ByVal state As UCPluginBase.State, ByVal host As UCPluginBase.Interfaces.IHost)`

Allows for overriding the saving state.

Parameters:

state: A State object.

host: The object representing the main application.



Convert

`Function Convert(ByVal host As UCPluginBase.Interfaces.IHost, ByVal data As Object, ByVal convertFrom As Object, ByVal convertTo As Object, ByVal otherArgs() As Object, ByVal error As Boolean) As String`

Converts the data specified to the desired data type.

Parameters:

host: The object representing the main application.

data: The data to convert.

convertFrom: What to convert from.

convertTo: What to convert to.

otherArgs: Other arguments that may be passed from the program.

error: Specifies whether an error took place.

Return Values:

The result of the conversion, in string form.

**DefineOtherArgs**

Function DefineOtherArgs() **As String**()

Defines any arguments the conversion wants from the program at run-time.

**DeInitialize**

Sub DeInitialize(**ByVal** host **As** UCPluginBase.Interfaces.IHost)

This method is called when the conversion is unloaded.

Parameters:

host: The object representing the main application.

**DisableInvert**

ReadOnly Property DisableInvert **As Boolean**

Specifies whether to disable the invert button.

**DisableZeroResultDialogue**

ReadOnly Property DisableInvert **As Boolean**

Specifies whether to disable the zero result dialogue.

**GetConversionInfo**

Function GetConversionInfo() **As** UCPluginBase.ConvInfo

Gets the conversion information.

Return Values:

A ConvInfo object.

**GetConvertFromList**

Function GetConvertFromList() **As String**()

Returns a list of strings to load in the convert from combobox.

**GetConvertToList**

Function GetConvertToList() **As String**()

Returns a list of strings to load in the convert to combobox.

**GetExtraInfo**

ReadOnly Property GetExtraInfo **As String**

Gets the extra information (if any). This is normally advice on how to use the conversion, if it only takes certain values.

**Initialize**

Sub Initialize(**ByVal** host **As** UCPluginBase.Interfaces.IHost)

Sets the initial details that this conversion requires, with full access to the Ihost object.

Parameters:

host: The object representing the main application.



Name

ReadOnly Property Name *As String*

The name of this conversion. This has to be less than 22 characters.



NeedsInternet

ReadOnly Property NeedsInternet *As Boolean*

Specifies whether the conversion needs an internet connection.



OverrideDefaultOpen

ReadOnly Property OverrideDefaultOpen *As Boolean*

Specifies whether to override the default opening state.



OverrideDefaultSave

ReadOnly Property OverrideDefaultSave *As Boolean*

Specifies whether to override the default saving state.



OverrideDefaultValidation

ReadOnly Property OverrideDefaultValidation *As Boolean*

Specifies whether to override the default validation.



PerformValidation

Function PerformValidation(*ByVal* host *As UCPluginBase.Interfaces.IHost*)

Allows custom validation to be performed against the data on the main screen.

Parameters:

host: The object representing the main application.

Return Values:

A value whether validation succeeded or not.



IHost

Public Interface IHost

Represents the main Universal Converter application.



AppState

Property AppState *As UCPluginBase.IHost*

Gets/Sets the current application state.



GetVersion

Function GetVersion() *As String*

Returns the application version in string format.



InvalidEntryErrorMessage

Property InvalidEntryErrorMessage *As String*

The data error message displayed as a result for conversions if the conversion is passed invalid data.



ShowFeedback

Sub ShowFeedback(*ByVal* message *As String*,
ByVal title *As String*, *ByVal* type *As*
UCPluginBase.Interfaces.IHost.FeedbackType)

Shows a message as feedback.

Parameters:

message: Specifies the message to show.

title: The title of the message box.

type: The type of feedback.



ShowInvalidEntryError

Sub ShowFeedback(*ByVal* conversion *As*
UCPluginBase.Interfaces.IConversion)

Shows a generic invalid entry error message.

Parameters:

conversion: The conversion to show the error message for.



TryAgainLaterMessage

Property TryAgainLaterMessage *As String*

The error message displayed as a result if something has gone wrong, suggesting to try again later.



UpdateProgress

Sub UpdateProgress(*ByVal* progressVal *As Integer*)

Updates the main application progress bar.

Parameters:

progressVal: Specifies the new value to set as the progress. Between 0 and 10.



IHost.FeedbackType

Public Enum FeedbackType *As Integer*

Specifies the type of feedback.



Error

Public Const Error *As*

UCPluginBase.Interfaces.IHost.FeedbackType = 2



Information

Public Const Information *As*

UCPluginBase.Interfaces.IHost.FeedbackType = 0



Warning

`Public Const` Warning `As`

`UCPluginBase.Interfaces.IHost.FeedbackType = 1`

This hierarchical structure is Copyright © 2010 Anthony R. Daly. All copying for commercial use is strictly prohibited.

Final Word

I hope you have found this guide useful and helpful to you in creating Universal Converter™ plugins. As already mentioned, the plugin system is very new to Universal Converter. However, keep an eye out, because the plugin system, and [UC Online™](#) as a whole is going to expand a lot in the very near future.

Below are some contact and information links which it would be useful to refer to at times:

- ➡ General contact email (for any questions or queries)

general@universalconverter.net

- ➡ Support and troubleshooting email (for any errors or troubles)

support@universalconverter.net

- ➡ Universal Converter website

<http://www.universalconverter.net/>

- ➡ UC Online website

<http://online.universalconverter.net/>

- ➡ Submitting files to UC Online (guide)

[UCOnlineSubmissionGuide.pdf](#)

Thank you for reading this guide, I hope I will see your plugin up on UC Online soon!

Anthony Daly
Creator of Universal Converter