

HowTo – Create an Encryption-Plugin for Cryptool 2.0 using Visual Studio 2008

This document is dedicated for the developer of a plugin for the e-learning program Cryptool 2.0. It describes step-by-step how you can create a new plugin in VS2008.

How this can be done is described in this document, building a sample plugin which delivers the functionality for a “new” **encryption algorithm**.

The new plugin contains the according encryption code and offers its functionality to the Cryptool application using the interface “IEncryptionAlgorithm” from the CrypPluginBase.

Additionally it delivers the according controls (like buttons, text boxes) and information to the TaskPane (icon, caption, descriptions).

Please be aware that the Cryptool 2.0 project is under development. Therefore, the screenshots in this document might represent the latest state of the project. However, we aim to provide a documentation and screenshots that can always guide you through the first steps on the development of a Plugin. If you anyhow detect a screenshot or description that does not lead you to success, we appreciate your feedback. Please visit the Cryptool project website (www.cryptool.org) to get in touch with us.

Author:	Sebastian Przybylski
Date:	2008-04-21
Version:	0.9
Reviewed by:	Philipp Südmeyer

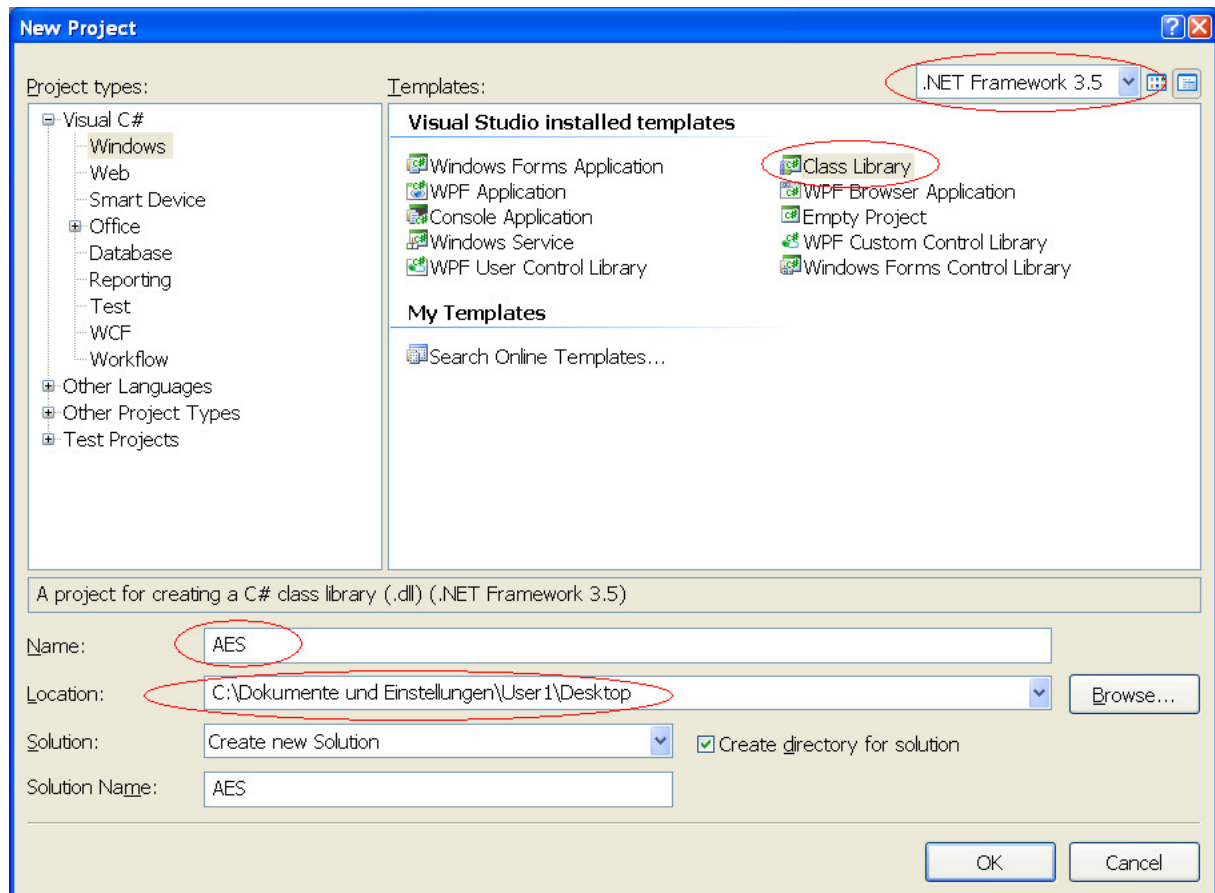
Content

HowTo – Create an Encryption-Plugin for Cryptool 2.0 using Visual Studio 2008	1
1. Create a new project in VS2008 for your plugin	3
2. Select the interface, your plugin wants to serve.....	4
3. Create the classes for the algorithm and for its settings	6
3.1 Create the class for the algorithm (AES)	6
3.2 Create the class for the settings (AESSettings).....	8
3.3 Add namespace for the class AES and the place from where to inherit.....	9
3.4 Add the interface functions for the class AES	9
3.5 Add namespace and interfaces for the class AESSettings.....	11
3.6 Add controls for the class AESSettings (if needed)	11

4.	Select and add an image as icon for the class AES	17
5.	Set the attributes for the class AES	20
6.	Set the private variables for the settings in the class AES.....	24
7.	Define the code of the class AES to fit the interface.....	25
8.	Complete the actual code for the class AES.....	28
9.	Import the plugin to Cryptool and test it	30
10.	Source code and source template.....	32

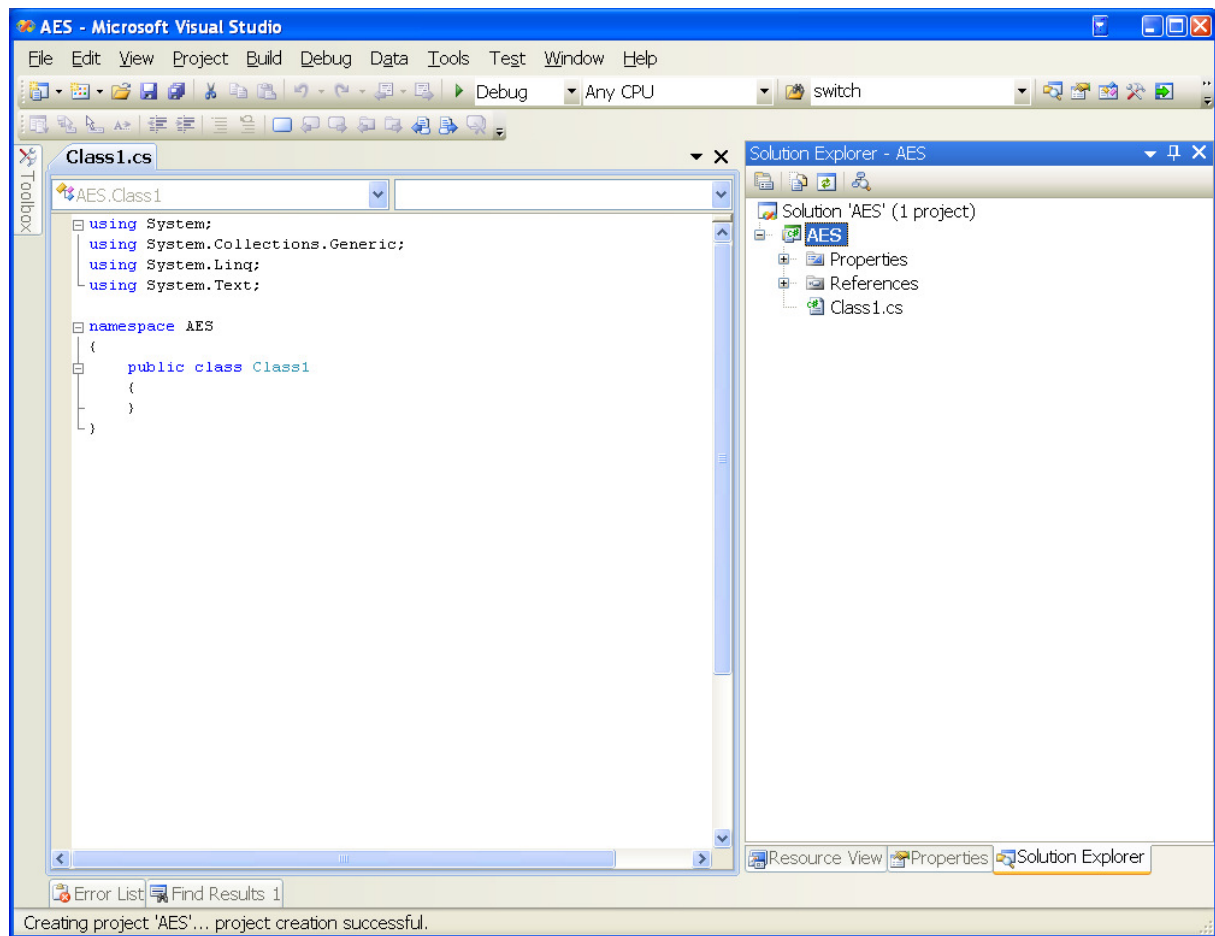
1. Create a new project in VS2008 for your plugin

Open Visual Studio 2008 and create a new project:



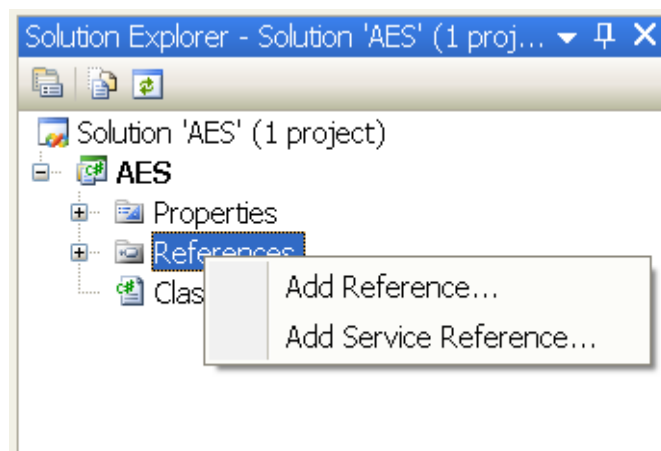
Select “.NET-Framework 3.5” as the target framework (the Visual Studio Express edition does not provide this selection because it automatically chooses the actual target framework), and “Class Library” as default template to create a DLL file. Give the project a unique and significant name (here: “AES”), and choose a location where to save (the Express edition will ask later for a save location when you close your project or your environment). Finally confirm by pressing the “OK” button.

Now your Visual Studio solution should look like this:



2. Select the interface, your plugin wants to serve

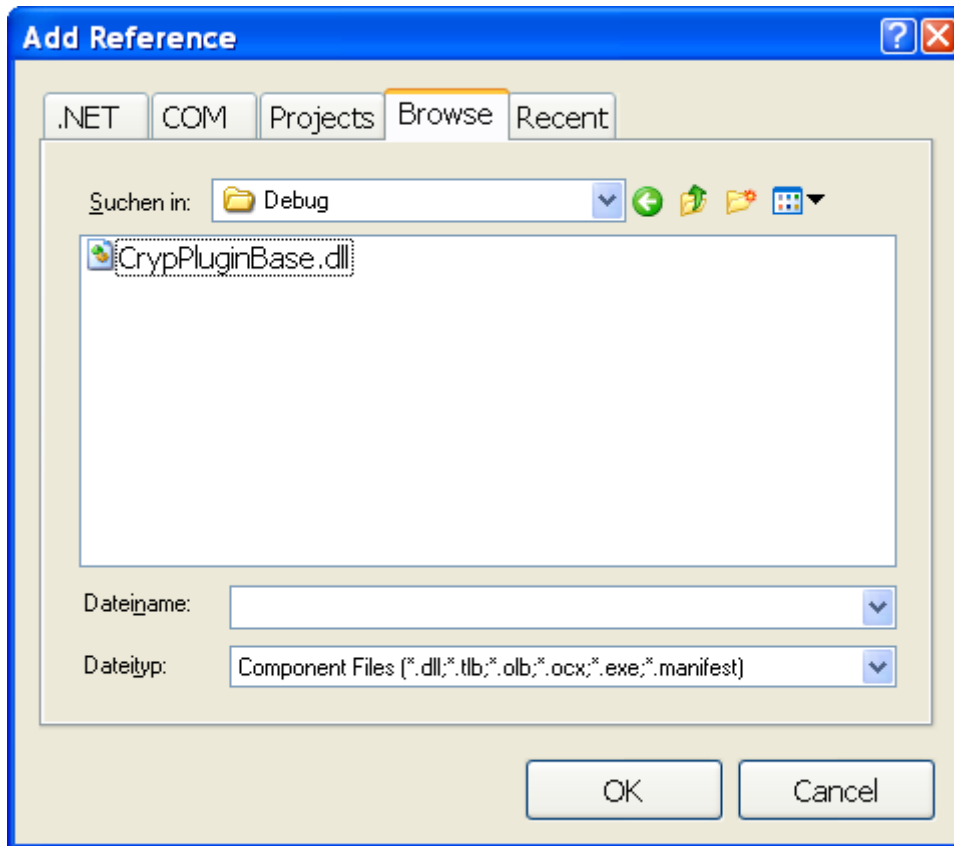
First we have to add a reference to the Cryptool library called “CrypPluginBase.dll” where all necessary Cryptool plugin interfaces are declared.



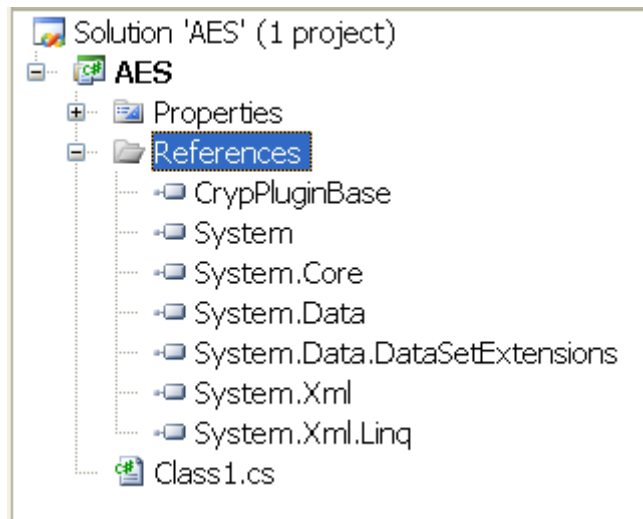
Right click on the “Reference” item in the Solution Explorer and choose “Add Reference...”. Now browse to the path where the library file is located (e.g. “c:\Documents and Settings\<Username>\My Documents\Visual Studio 2008\Projects\CrypPluginBase\bin\Debug”) and select the library by double clicking the file or pressing the “OK” button. You can also find the CrypPluginBase.dll on our

Subversion repository.

The current version is always located in the following path: “<https://www.vs.uni-due.de/svn/CrypTool2/trunk/CrypPluginBase/bin/Debug/>”



Afterwards your reference tree view should look like this:



If your plugin will be based on further libraries, you have to add them in the same way.

3. Create the classes for the algorithm and for its settings

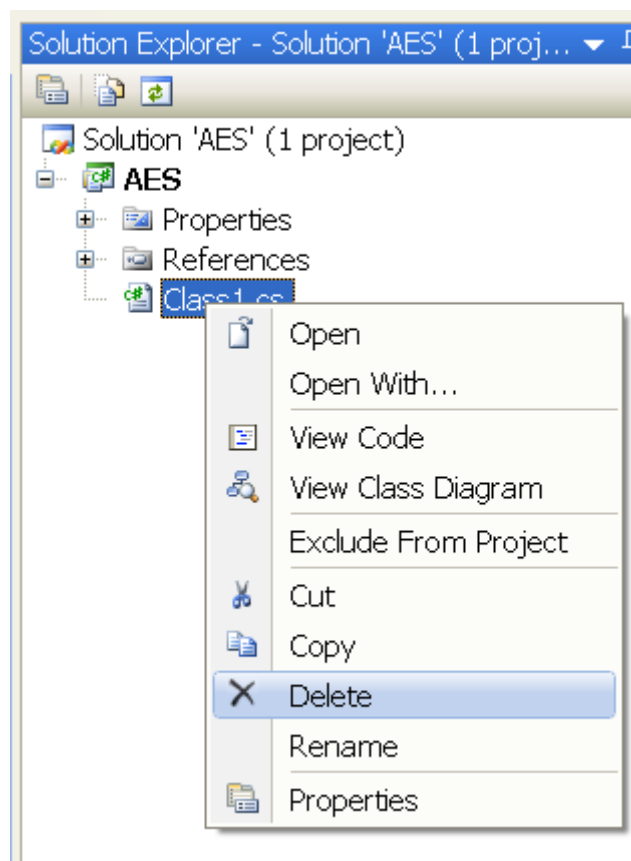
In the next step we have to create two classes. The first class named “AES” has to inherit from `IEncryptionAlgorithm`, and the second class named “AESSettings” from `IEncryptionAlgorithmSettings`.

3.1 Create the class for the algorithm (AES)

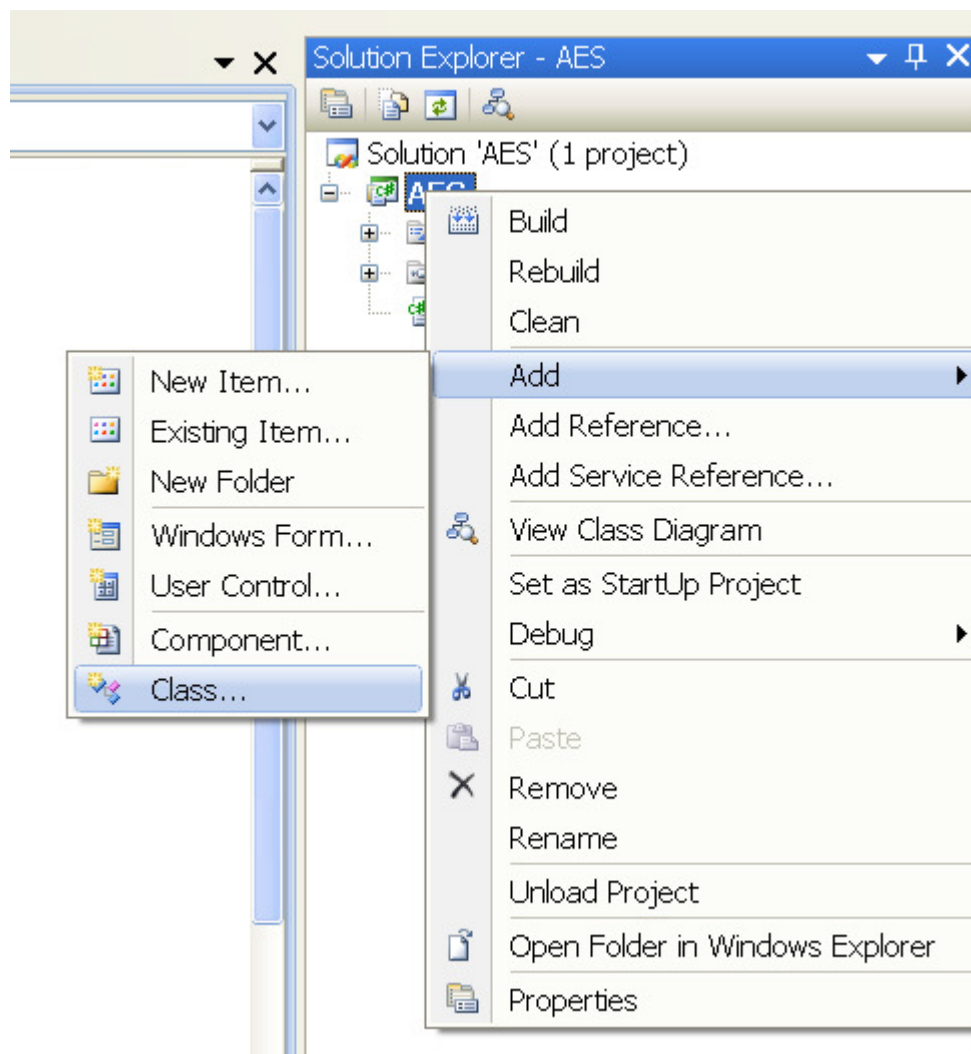
Visual Studio automatically creates a class which has the name “Class1.cs”. There are two ways to change the name to “AES.cs”:

- Rename the existent class
- Delete the existent class and create a new one.

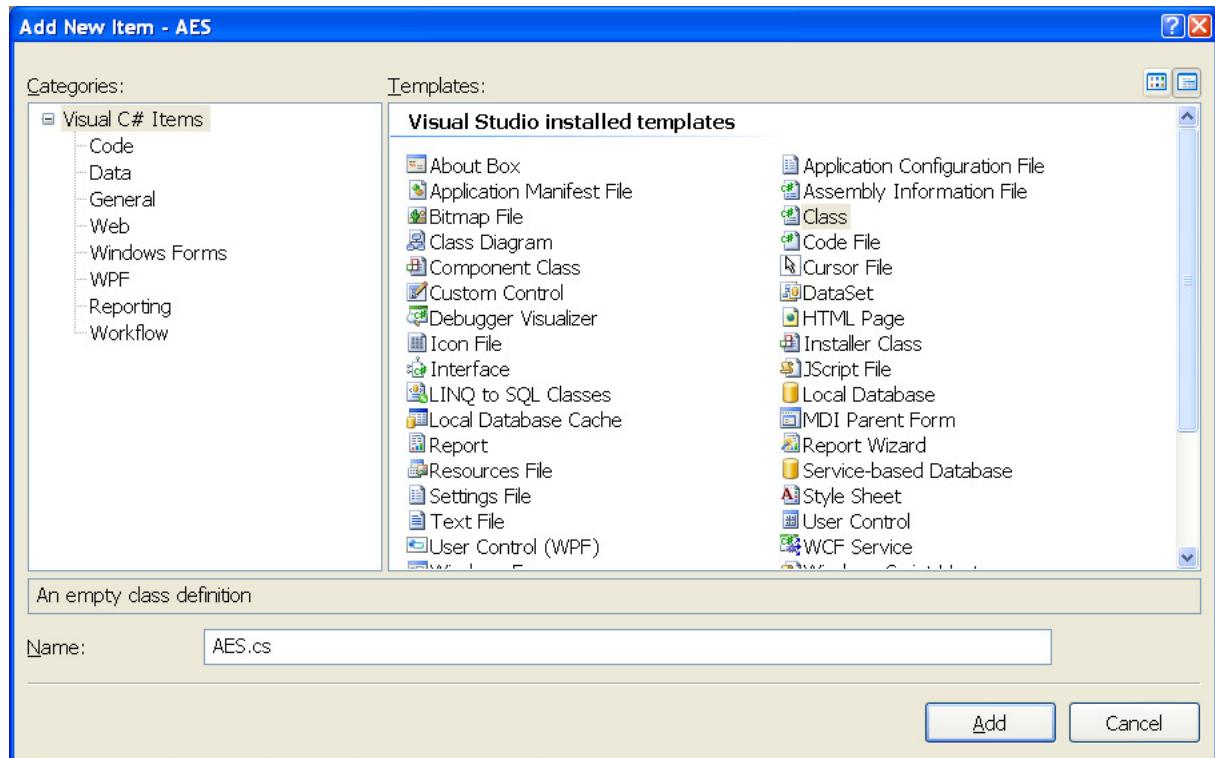
We choose the second way as you can see in the next screenshot:



Now right click on the project item “AES” and select “Add->Class...”.

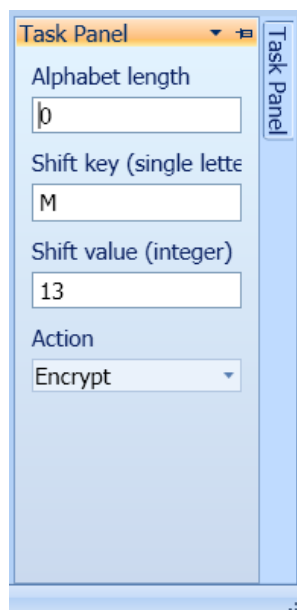


Give your class a unique name. We call the class as mentioned above “AES.cs” and make it public to be available to other classes.



3.2 Create the class for the settings (AESSettings)

Act in the same way to add a second class for `IEncryptionAlgorithmSettings`. We call the class “AESSettings”. The settings class provides the necessary information about controls, captions and descriptions and default parameters e.g. key settings, alphabets, key length and action to build the **TaskPane** in Cryptool. How a **TaskPane** could look like you can see below for the example of a Caesar encryption.



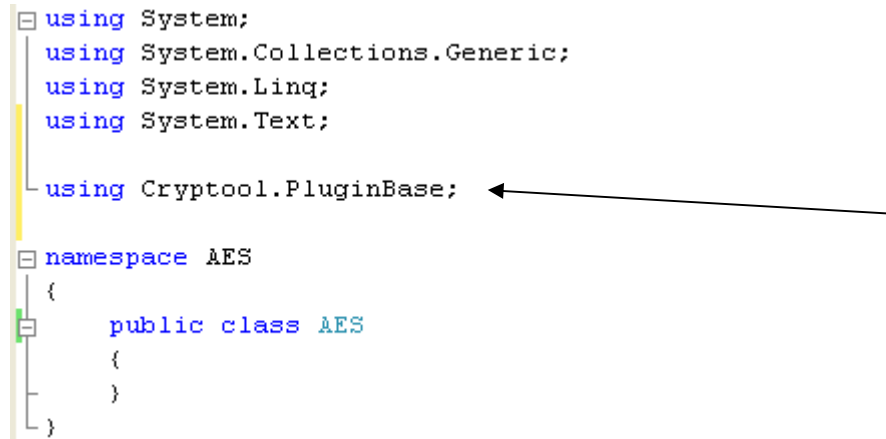
3.3 Add namespace for the class AES and the place from where to inherit

Now open the “AES.cs” file by double clicking on it at the Solution Explorer and include the CryptPluginBase namespace to the class header by typing in the according “using” statement:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

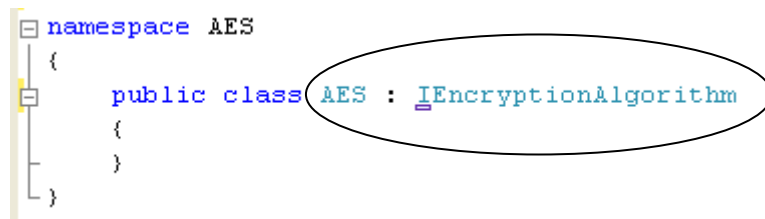
using Cryptool.PluginBase;

namespace AES
{
    public class AES
    {
    }
}
```



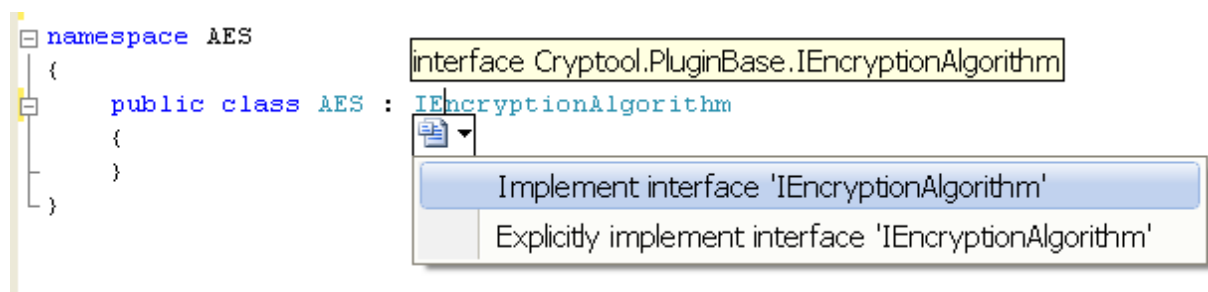
Next let your class “AES” inherit from IEncryptionAlgorithm by inserting of the following statement:

```
namespace AES
{
    public class AES : IEncryptionAlgorithm
    {
    }
}
```



3.4 Add the interface functions for the class AES

There is an underscore at the IEncryptionAlgorithm statement. Move your mouse over it or place the cursor at on and press “Shift+Alt+F10” and you will see the following submenu:



Choose the item “Implement interface ‘IEncryptionAlgorithm’”. Visual Studio will now place all available and needed interface members to interact with the Cryptool core (this saves you also a lot of code typing).

Your code will now look like this:

```

namespace AES
{
    public class AES : IEncryptionAlgorithm
    {
        #region IEncryptionAlgorithm Members

        public void Decrypt()
        {
            throw new NotImplementedException();
        }

        public void Encrypt()
        {
            throw new NotImplementedException();
        }

        public IEncryptionAlgorithmSettings Settings
        {
            get { throw new NotImplementedException(); }
            set { throw new NotImplementedException(); }
        }

        #endregion

        #region IPlugin Members

        public void Dispose()
        {
            throw new NotImplementedException();
        }

        public void Initialize()
        {
            throw new NotImplementedException();
        }

        public event ExecutionErrorHandler OnExecutionError;

        public event StatusBarProgressbarValueChangedHandler OnStatusBarProgressbarValueChanged;

        public event StatusBarTextChangedHandler OnStatusBarTextChanged;

        public System.Windows.Controls.UserControl PresentationControl
        {
            get { throw new NotImplementedException(); }
        }

        #endregion
    }
}

```

3.5 Add namespace and interfaces for the class AESSettings

Let's now take a look at the second class "AESSettings" by double clicking on the "AESSettings.cs" file in the Solution Explorer. First we also have to include the namespace of "Cryptool.PluginBase" to the class header and let the settings class inherit from "IEncryptionAlgorithmSettings" analogous as seen before at the AES class. Visual Studio will here also automatically place code from the Cryptool interface if available.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

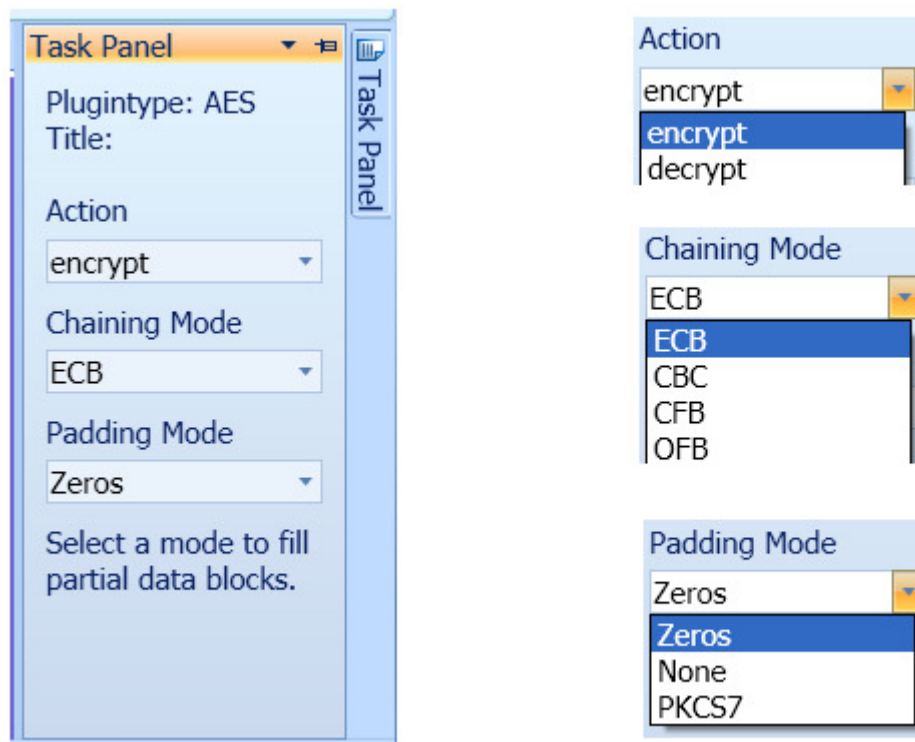
using Cryptool.PluginBase;

namespace AES
{
    public class AESSettings : IEncryptionAlgorithmSettings
    {
    }
}
```

3.6 Add controls for the class AESSettings (if needed)

Now we have to implement some kind of controls (like button, text box) if we need them in the Cryptool **TaskPane** to modify settings of the algorithm. We have cases where an algorithm doesn't have any kind of settings, so we would let this class empty. For this case you can see an example at the "HowTo-HashPlugin". Our AES algorithm needs some setting in the **TaskPane**. Encryption algorithms inheriting from IEncryptionAlgorithmSetting have one setting by default, given from the Cryptool interface. The property is called "Action" which is needed to detect if the user chose the encrypt function or the decrypt function.

Our goal is to get the **TaskPane** look like this:



First we have to define the member variables for the settings we want to provide. Our AES needs the following tree member variables:

- action = needed to decide between encrypt and decrypt
- mode = block cipher modes like ECB, CBC,...
- padding = padding mode to fill partial data blocks like zeros, PKCS7

Our member variables definition looks like this:

```
public class AESSettings : IEncryptionAlgorithmSettings
{
    private int action = 0; //0=encrypt, 1=decrypt
    private int mode = 0; //0="ECB", 1="CBC", 2="CFB", 3="OFB"
    private int padding = 0; //0="None", 1="Zeros", 2="PKCS7"
}
```

What we need next is the accordingly property function for this member variables.

First we declare the property for “action” with appropriate getter and setter functions:

```
public class AESSettings : IEncryptionAlgorithmSettings
{
    private int action = 0; //0=encrypt, 1=decrypt
    private int mode = 0; //0="ECB", 1="CBC", 2="CFB", 3="OFB"
    private int padding = 0; //0="None", 1="Zeros", 2="PKCS7"

    public int Action
    {
        get { return this.action; }
        set { this.action = (int)value; }
    }
}
```

To make this property identifiable for the TaskPane we need to attach an attribute to this property.

Attributes are meta data and a new kind of **declarative** information to define for example classes and class fields. You can also define custom components using attributes. At runtime you are able to get necessary information from your plugins with the aid of attributes.

In this case we have to use the “TaskPaneSettings” attribute as you can see below:

```
[TaskPaneSettings(
public int Action
{
    get { return this.action; }
    set { this.action = (int)value; }
}
```

The “TaskPaneSettings” attributes takes eight parameters:

```
[TaskPaneSettings(
1 of 2 TaskPaneSettingsAttribute.TaskPaneSettingsAttribute (string caption, string description, int order, bool enabled, DisplayLevel displayLevel, ControlType controlType,
string regularExpression, params string[] controlValues)
get { return this.action; }
set { this.action = (int)value; }
}
```

The first two parameters take a caption and a description of type string. Caption e.g. is needed to display the label of a control and the description provides the tool tip of the control by moving the mouse over it. All this labels are shown in the task pane. In this case we call our caption “Action” and make an according description as you can see below:

```
[TaskPaneSettings("Action", "Do you want the input data to be encrypted or decrypted?",
public int Action
{
```

The third parameter defines the position order of the control. This value from type integer is necessary for the TaskPane to collect the controls in the right order. Controls with lower value take the highest position at the TaskPane. Controls with the same value will be positioned with the same order as defined in the code. We want our action property to be placed at the top in our TaskPane, so we assign the lowest possible value to it:

```
[TaskPaneSettings("Action", "Do you want the input data to be encrypted or decrypted?", 1,
public int Action
{
```

The fourth parameter from type Boolean is needed to enable or disable the control in the TaskPane.

```
[TaskPaneSettings("Action", "Do you want the input data to be encrypted or decrypted?", 1, true,
public int Action
{
```

The fifth parameter chooses the display level. We have four display levels from type DisplayLevel:

- Beginner
- Experienced
- Expert
- Professional

The plugin developer can choose here, at which display level he wants to show his control in the TaskPane. The display level is selected in Cryptool at the home tab in the ribbon bar. If a control's display level is set to Experienced or higher, the control wouldn't be shown in the TaskPane when the user set the display level to "Beginner" in Cryptool.

```
[TaskPaneSettings("Action", "Do you want the input data to be encrypted or decrypted?", 1, true, DisplayLevel.Beginner,
public int Action
{
```

The sixth parameter from type ControlType provides the control type. At the moment you can choose between:

- ComboBox
- RadioButton
- TextBox

This control in the TaskPane is needed to display the input or output data (like the initialization vector) for the algorithms.

```
[TaskPaneSettings("Action", "Do you want the input data to be encrypted or decrypted?", 1, true, DisplayLevel.Beginner,
ControlType.ComboBox,
public int Action
{
```

With the seventh parameter one can describe regular expressions verify the user input. For example, if you want to allow the user to use only latin letters for the key, this would be the right place to safeguard this. In our case we don't want an expression for the AES and leave this parameter empty.

```
[TaskPaneSettings("Action", "Do you want the input data to be encrypted or decrypted?", 1, true, DisplayLevel.Beginner,
ControlType.ComboBox, "",
public int Action
{
```

The last parameter contains all values from type string array shown in the selected control which have to be supported by the given property. In this case our ComboBox control is needed for the "Action" property and provides accordingly the values "encrypt" and "decrypt".

```
[TaskPaneSettings("Action", "Do you want the input data to be encrypted or decrypted?", 1, true, DisplayLevel.Beginner,
ControlType.ComboBox, "", new string[] { "encrypt", "decrypt" })
public int Action
{
```

In a uniform manner, we can add more controls to be displayed in the TaskPane. Our AES algorithm will also provide a control for the chaining mode and for the padding mode. The whole code could look like this:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Cryptool.PluginBase;

namespace AES
{
    public class AESSettings : IEncryptionAlgorithmSettings
    {
        private int action = 0; //0=encrypt, 1=decrypt
        private int mode = 0; //0="ECB", 1="CBC", 2="CFB", 3="OFB"
        private int padding = 0; //0="None", 1="Zeros", 2="PKCS7"

        [TaskPaneSettings("Action", "Do you want the input data to be encrypted or decrypted?", 1, true, DisplayLevel.Beginner,
            ControlType.ComboBox, "", new String[] { "encrypt", "decrypt" })]
        public int Action
        {
            get { return this.action; }
            set { this.action = (int)value; }
        }

        [TaskPaneSettings("Chaining Mode", "Select the block cipher mode of operation.", 2, true, DisplayLevel.Experienced,
            ControlType.ComboBox, "", new String[] { "ECB", "CBC", "CFB", "OFB" })]
        public int Mode
        {
            get { return this.mode; }
            set { this.mode = (int)value; }
        }

        [TaskPaneSettings("Padding Mode", "Select a mode to fill partial data blocks.", 2, true, DisplayLevel.Experienced,
            ControlType.ComboBox, "", new String[] { "Zeros", "None", "PKCS7" })]
        public int Padding
        {
            get { return this.padding; }
            set { this.padding = (int)value; }
        }
    }
}
```

To ensure a right data binding between the TaskPane and you plugin, we need to add an event handler to our class as following:

```
public class AESSettings : IEncryptionAlgorithmSettings
{
    private members

    [TaskPaneSettings("Action", "Do you want the input data to be encrypted or decrypted?", 1, true, DisplayLevel.Beginner,
        ControlType.ComboBox, "", new String[] { ... })]

    [TaskPaneSettings("Chaining Mode", "Select the block cipher mode of operation.", 2, true, DisplayLevel.Experienced,
        ControlType.ComboBox, "", new String[] { ... })]

    [TaskPaneSettings("Padding Mode", "Select a mode to fill partial data blocks.", 2, true, DisplayLevel.Experienced,
        ControlType.ComboBox, "", new String[] { ... })]

    public event System.ComponentModel.PropertyChangedEventHandler PropertyChanged;
}
```

Next we need a function to refresh the TaskPane with our plugin like this:

```
protected void OnPropertyChanged(string name)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(name));
    }
}
```

All property functions have to call this Event.

The whole settings class should look like this:

```
namespace AES
{
    public class AESSettings : IEncryptionAlgorithmSettings
    {
        private int action = 0; //0=encrypt, 1=decrypt
        private int mode = 0; //0="ECB", 1="CBC", 2="CFB", 3="OFB"
        private int padding = 0; //0="None", 1="Zeros", 2="PKCS7"

        [TaskPaneSettings("Action", "Do you want the input data to be encrypted or decrypted?", 1, true, DisplayLevel.Beginner,
            ControlType.ComboBox, "", new String[] { "encrypt", "decrypt" })]
        public int Action
        {
            get { return this.action; }
            set
            {
                this.action = (int)value;
                OnPropertyChanged("Action");
            }
        }

        [TaskPaneSettings("Chaining Mode", "Select the block cipher mode of operation.", 2, true, DisplayLevel.Experienced,
            ControlType.ComboBox, "", new String[] { "ECB", "CBC", "CFB", "OFB" })]
        public int Mode
        {
            get { return this.mode; }
            set
            {
                this.mode = (int)value;
                OnPropertyChanged("Mode");
            }
        }

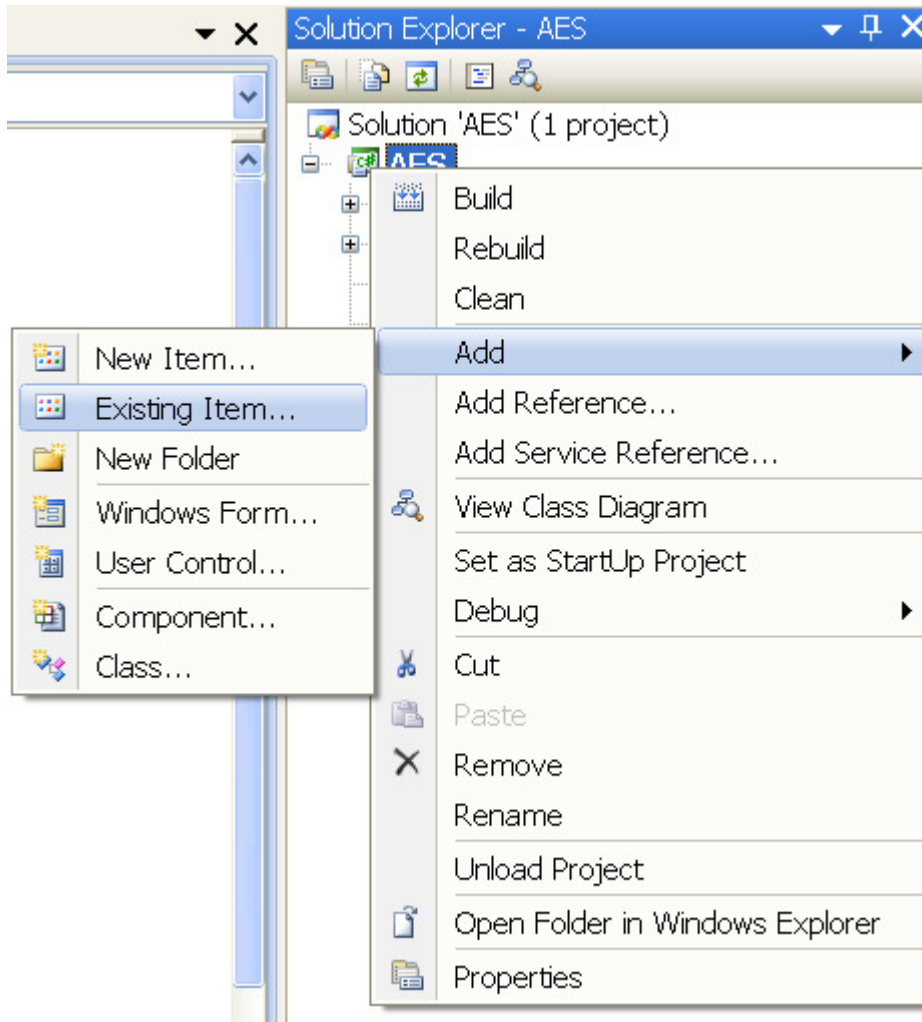
        [TaskPaneSettings("Padding Mode", "Select a mode to fill partial data blocks.", 2, true, DisplayLevel.Experienced,
            ControlType.ComboBox, "", new String[] { "Zeros", "None", "PKCS7" })]
        public int Padding
        {
            get { return this.padding; }
            set
            {
                this.padding = (int)value;
                OnPropertyChanged("Padding");
            }
        }

        public event System.ComponentModel.PropertyChangedEventHandler PropertyChanged;

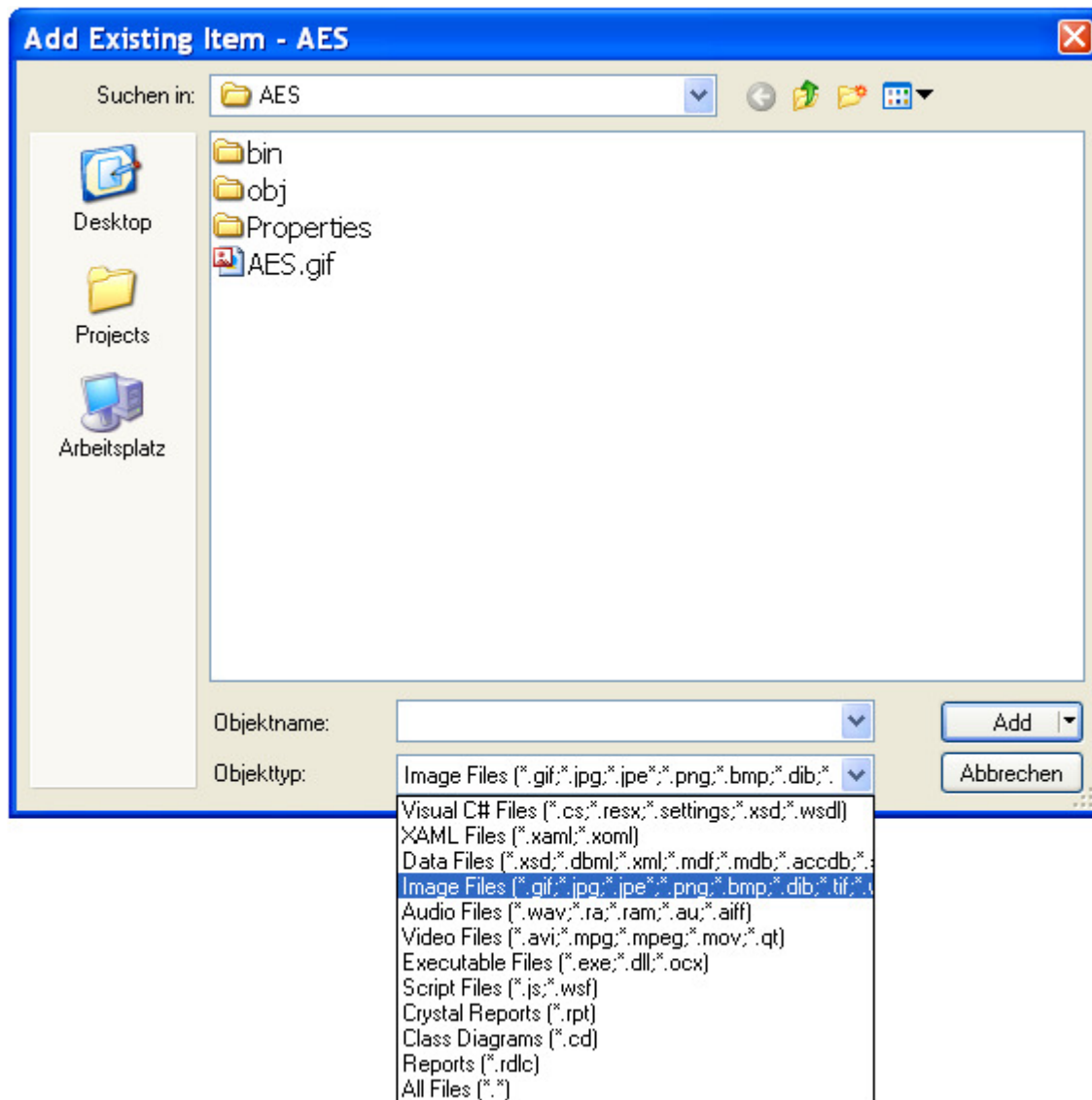
        protected void OnPropertyChanged(string name)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(name));
            }
        }
    }
}
```


4. Select and add an image as icon for the class AES

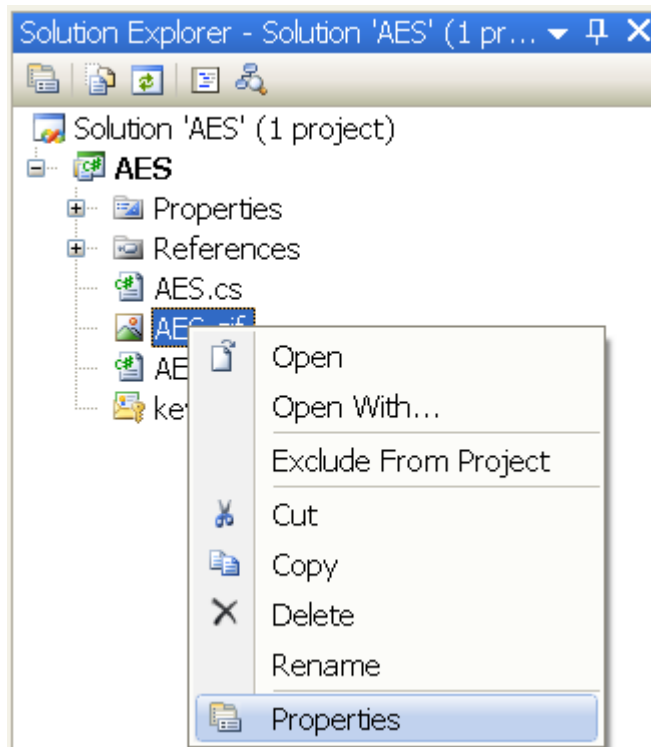
Before we go back to the code of the AES class, we have to add an icon image to our project, which will be shown in the Cryptool **ribbon bar**. Note that an icon must have a size of at least 32 x 32 pixels. Smaller icons will be accepted too, but looks unsightly in the ribbon bar. Therefore, right click on the project item “AES” within the Solution Explorer, and select “Add->Existing Item...”:



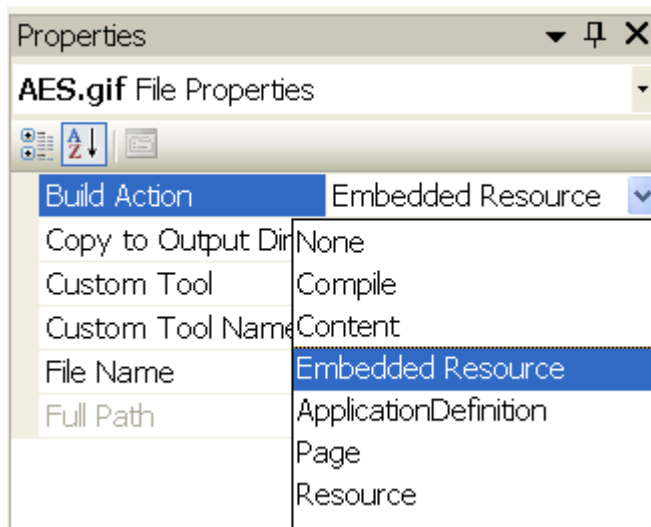
Then select “Image Files” as file type, and choose the icon for your plugin:



Finally we have to set the icon as an “Embedded Resource” to avoid providing the icon as a separate file. Make a right click on the icon and select the item “Properties”:



In the “Properties” panel you have to set the “Build Action” to “Embedded Resource”:



5. Set the attributes for the class AES

Now let's go back to the code of the AES class ("AES.cs" file). First we have to set the five elements of the attribute's list named "PluginInfo" of our class. These attributes are used by the Cryptool PluginBase and CrypCore to provide information about this plugin:

- its unique Guid (to identify the plugin) [Guid = Globally Unique Identifier]
- its name (e.g. to provide the button content)
- its short description (e.g. to provide the button tool tip)
- its detailed description (e.g. to provide a long describing text area) and
- its icon path (needed to provide e.g. the button image).

```
namespace AES
{
    [PluginInfo(
        public class AES : IEncryptionAlgorithm
        {
```

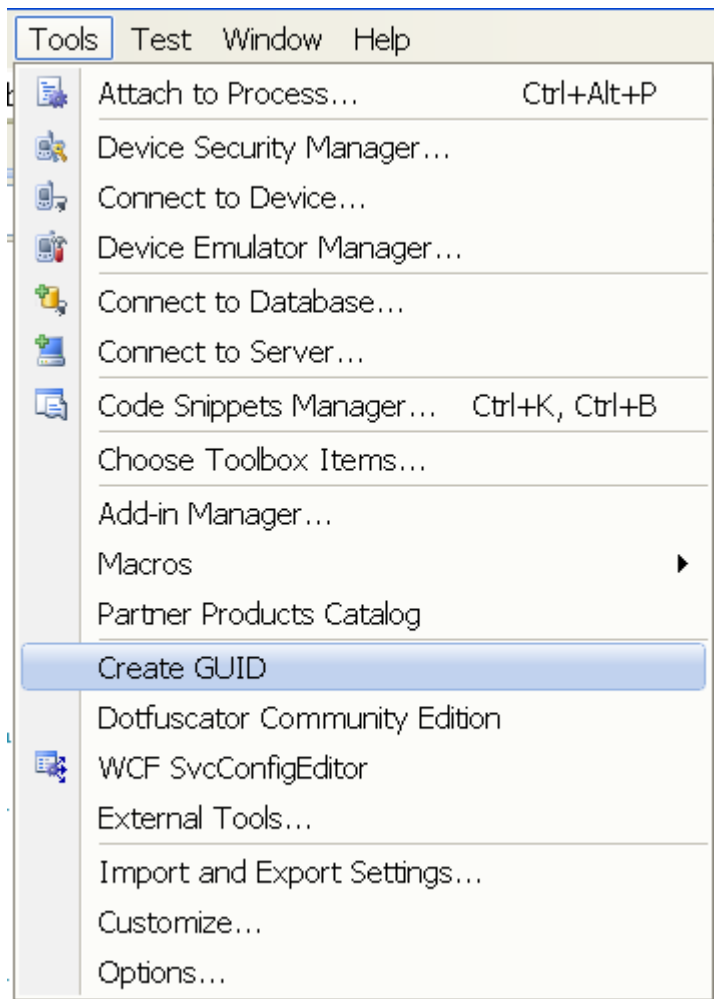
The „PluginInfo“ attribute takes five parameters (all four are of type string):

```
[PluginInfo(
1 of 2 PluginInfoAttribute.PluginInfoAttribute (string id, string name, string shortDescription, string detailedDescription, string iconName)
```

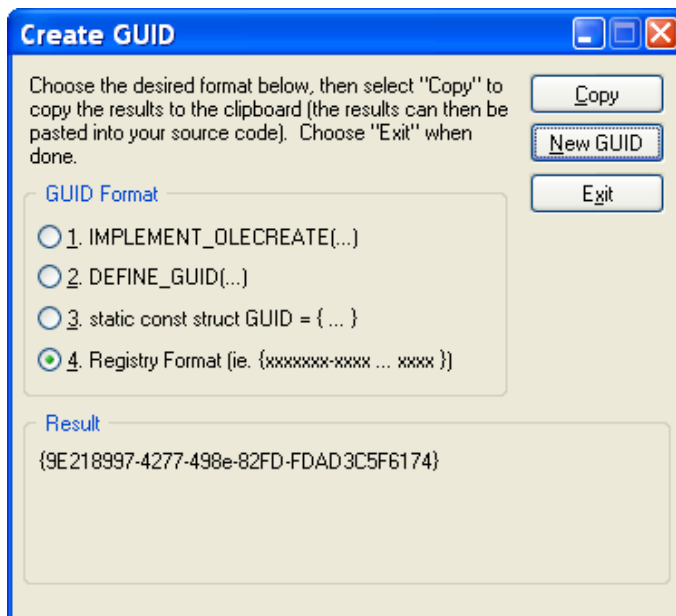
This first parameter takes a Guid to make a unique id of this plugin. Visual Studio gives us a tool to generate new Guids. The Visual C#-Express Edition doesn't provide this tool by default. But you can download and execute it apart, at:

<http://www.microsoft.com/downloads/details.aspx?familyid=94551f58-484f-4a8c-bb39-adb270833afc&displaylang=en>

In Visual Studio 2008 you will find this Guid generator at the menu: “Tools->Create GUID”.



Now you have to click on the “New GUID” button in the GUID creator dialog to generate a new id which is displayed within the “Result” group box. Now copy the generated Guid to your clipboard by pressing the “Copy” button and leave this window by pressing the “Exit” button.



Set your cursor again at the first parameter of your class attribute “[PluginInfo...” and paste your generated Guid from the clipboard. Then delete the curly brackets surrounding the pasted Guid and instead, surround the Guid with quotation marks (to make a type string) as you can see at the following screenshot:

```
[PluginInfo(
    "419AB9DE-2FAD-48d7-9098-A2DADD465EB6",
    public class AES : IEncryptionAlgorithm
    {
```

The next three parameters are needed to define the plugin’s name, short description and its detailed description:

```
[PluginInfo(
    "419AB9DE-2FAD-48d7-9098-A2DADD465EB6",
    "AES",
    "Advanced Encryption Standard (Rijandel)",
    "detailed description",
    public class AES : IEncryptionAlgorithm
    {
```

The last parameter tells Cryptool the name of the plugin’s icon. This parameter is made up by <namespace name>.<file name>, where <file name> is the file you chose in chapter 4.

```
[PluginInfo(
    "419AB9DE-2FAD-48d7-9098-A2DADD465EB6",
    "AES",
    "Advanced Encryption Standard (Rijandel)",
    "detailed description",
    "Cryptool.AES.AES.gif")]
    public class AES : IEncryptionAlgorithm
    {
```

Other than hash algorithms, encryption algorithms need to be defined to which encryption type they belong. This property is also done by an attribute as you can see below:

```
[PluginInfo(
    "419AB9DE-2FAD-48d7-9098-A2DADD465EB6",
    "AES",
    "Advanced Encryption Standard (Rijandel)",
    "detailed description",
    "Cryptool.AES.AES.gif")]
[EncryptionAlgorithmType(EncryptionAlgorithmType.SymmetricBlock)]

    public class AES : IEncryptionAlgorithm
    {
```

We have five encryption algorithm types to choose:

- Asymmetric
- Classic
- Hybrid
- Symmetric Block and
- Symmetric Stream

This property is needed e.g. to place the plugin in the right ribbon tab. We are also happy to add new classifications, if reasonable. Just get in touch with us.

6. Set the private variables for the settings in the class AES

The next step is to define some private variables needed e.g. for the settings, input data, output data, input key and initialization vector which could look like this:

```
public class AES : IEncryptionAlgorithm
{
    private AESSettings settings;
    private Stream inputStream;
    private Stream outputStream;
    private byte[] inputKey;
    private byte[] inputIV;
    private CryptoStream p_crypto_stream_dec;
    private CryptoStream p_crypto_stream_enc;
    private bool stop = false;
}
```

Please notice the sinuous line at the type “Stream” of the variable inputStream. This means you have to include the appropriate namespace. In this case it means you have to include the namespace “System.IO”. How to implement the automatically the necessary namespace you can see at chapter 3.4.

Finally we will need to include the following namespaces:

System.IO – to handle e.g. with streams

System.Security.Cryptography – if we want to use .net algorithms

Cryptool.PluginBase – the master namespace provided by Cryptool

System.ComponentModel – needed for the [PropertyChangedEventHandler](#)

Also notice, that we use streams for input and output to perform a file handling in Cryptool with huge files.

7. Define the code of the class AES to fit the interface

Next we have to complete our code to correctly serve the interface.

First we add a constructor to our class where we can create an instance of our settings class:

```
public class AES : IEncryptionAlgorithm
{
    private AESSettings settings;
    private Stream inputStream;
    private Stream outputStream;
    private byte[] inputKey;
    private byte[] inputIV;
    private CryptoStream p_crypto_stream_dec;
    private CryptoStream p_crypto_stream_enc;
    private bool stop = false;

    public AES()
    {
        this.settings = new AESSettings();
    }
}
```

Secondly, we have to complete the property function “Settings” provided by the interface:

```
public IEncryptionAlgorithmSettings Settings
{
    get { return this.settings; }
    set { this.settings = (AESSettings)value; }
}
```

Thirdly we have to define input and output properties with their according attributes. This step is necessary to bind our input and output data with the Cryptool editor plugins input and output controls.

In this case the AES algorithm needs three input properties (input stream, key, initialization vector) and one output property:

```
[Input("Input", "Data to be encrypted or decrypted")]
public Stream InputStream
{
    get { return this.inputStream; }
    set { this.inputStream = value; }
}

//[OptionalAttribute]
[Input("Key", "Must be 16, 24 or 32 bytes.")]
public byte[] InputKey
{
    get { return this.inputKey; }
    set { this.inputKey = value; }
}

//[OptionalAttribute]
[Input("IV", "IV to be used in chaining modes, must be the same as the Blocksize in bytes.")]
public byte[] InputIV
{
    get { return this.inputIV; }
    set { this.inputIV = value; }
}

[Output("Output stream", "Encrypted or decrypted output data")]
public Stream OutputStream
{
    get { return this.outputStream; }
    set { this.outputStream = value; }
}
```

The input and output attributes provide two parameters for the caption and the description of the accordingly input and output value. This properties will be shown in the editor workspace by choosing this plugin.

The last step to make our code compileable is to add three assembly references to provide the necessary “Windows” namespace for our **user control** function called “PresentationControl”. As explained above how to add new references to our project, you have to add the following .NET components:

- PresentationCore
- PresentationFramework
- WindowsBase

The function “PresentationControl” is served for the PluginBase if a plugin developer wants to provide his own graphic user interface for Cryptool.

8. Complete the actual code for the class AES

Up to now, the plugin is ready for the Cryptool base application to be accepted and been shown correctly in the Cryptool menu. What we need now, is the implementation of the actual algorithm functions “Encrypt()” and “Decrypt()” which are up to you as the plugin developer.

Let us demonstrate the Encrypt() and Decrypt() function, too. Our algorithm is based on the .NET framework:

```
public void Encrypt ()
{
    try
    {
        checkForInputStream();
        if (this.inputStream.CanSeek) this.inputStream.Position = 0;
        SymmetricAlgorithm p_alg = new AesCryptoServiceProvider();
        ConfigureAlg(p_alg);
        ICryptoTransform p_encryptor = p_alg.CreateEncryptor();
        outputStream = new CryptoolStream(this);
        p_crypto_stream_enc = new CryptoStream(this.inputStream, p_encryptor, CryptoStreamMode.Read);
        byte[] buffer = new byte[p_alg.BlockSize / 8];
        int bytesRead;
        int position = 0;
        DateTime startTime = DateTime.Now;
        while ((bytesRead = p_crypto_stream_enc.Read(buffer, 0, buffer.Length)) > 0 && !stop)
        {
            outputStream.Write(buffer, 0, bytesRead);

            if ((int)(inputStream.Position * 100 / inputStream.Length) > position)
            {
                position = (int)(inputStream.Position * 100 / inputStream.Length);
                if (OnStatusBarProgressbarValueChanged != null)
                    OnStatusBarProgressbarValueChanged(this, new StatusBarProgressbarValueChangedEventArgs(inputStream.Position, inputStream.Length));
            }
        }
        p_crypto_stream_enc.Flush();
        DateTime stopTime = DateTime.Now;
        TimeSpan duration = stopTime - startTime;
        (outputStream as CryptoolStream).FinishWrite();
        if (OnStatusBarTextChanged != null && !stop)
        {
            OnStatusBarTextChanged(this, new StatusBarTextChangedEventArgs("Encryption complete! (in: " + inputStream.Length.ToString() +
                " bytes, out: " + outputStream.Length.ToString() + " bytes)", this, LogLevel.Info));
            OnStatusBarTextChanged(this, new StatusBarTextChangedEventArgs("Time used: " + duration.ToString(), this, LogLevel.Debug));
        }
        if (stop)
        {
            OnStatusBarTextChanged(this, new StatusBarTextChangedEventArgs("Aborted!", this, LogLevel.Info));
        }
    }
    catch (Exception exception)
    {
        if (OnStatusBarTextChanged != null)
        {
            OnStatusBarTextChanged(this, new StatusBarTextChangedEventArgs(exception.Message, this, LogLevel.Info));
        }
    }
    finally
    {
    }
}
```

```

public void Decrypt()
{
    try
    {
        checkForInputStream();

        if (this.inputStream.CanSeek) this.inputStream.Position = 0;

        SymmetricAlgorithm p_alg = new AesCryptoServiceProvider();
        ConfigureAlg(p_alg);
        ICryptoTransform p_encryptor = p_alg.CreateDecryptor();
        outputStream = new CryptoolStream(this);
        p_crypto_stream_dec = new CryptoStream(this.inputStream, p_encryptor, CryptoStreamMode.Read);
        byte[] buffer = new byte[p_alg.BlockSize / 8];
        int bytesRead;
        int position = 0;
        DateTime startTime = DateTime.Now;
        while ((bytesRead = p_crypto_stream_dec.Read(buffer, 0, buffer.Length)) > 0 && !stop)
        {
            outputStream.Write(buffer, 0, bytesRead);
            if ((int)(inputStream.Position * 100 / inputStream.Length) > position)
            {
                position = (int)(inputStream.Position * 100 / inputStream.Length);
                if (OnStatusBarProgressbarValueChanged != null)
                    OnStatusBarProgressbarValueChanged(this, new StatusBarProgressbarValueChangedEventArgs(inputStream.Position, inputStream.Length));
            }
        }
        p_crypto_stream_dec.Flush();
        DateTime stopTime = DateTime.Now;
        TimeSpan duration = stopTime - startTime;
        (outputStream as CryptoolStream).FinishWrite();
        if (OnStatusBarTextChanged != null && !stop)
        {
            OnStatusBarTextChanged(this, new StatusBarTextChangedEventArgs("Decryption complete! (in: " + inputStream.Length.ToString() +
                " bytes, out: " + outputStream.Length.ToString() + " bytes)", this, LogLevel.Info));
            OnStatusBarTextChanged(this, new StatusBarTextChangedEventArgs("Time used: " + duration.ToString(), this, LogLevel.Debug));
        }
        if (stop)
        {
            OnStatusBarTextChanged(this, new StatusBarTextChangedEventArgs("Aborted!", this, LogLevel.Info));
        }
    }
    catch (Exception exception)
    {
        if (OnStatusBarTextChanged != null)
        {
            OnStatusBarTextChanged(this, new StatusBarTextChangedEventArgs(exception.Message, this, LogLevel.Error));
        }
    }
    finally
    {
    }
}

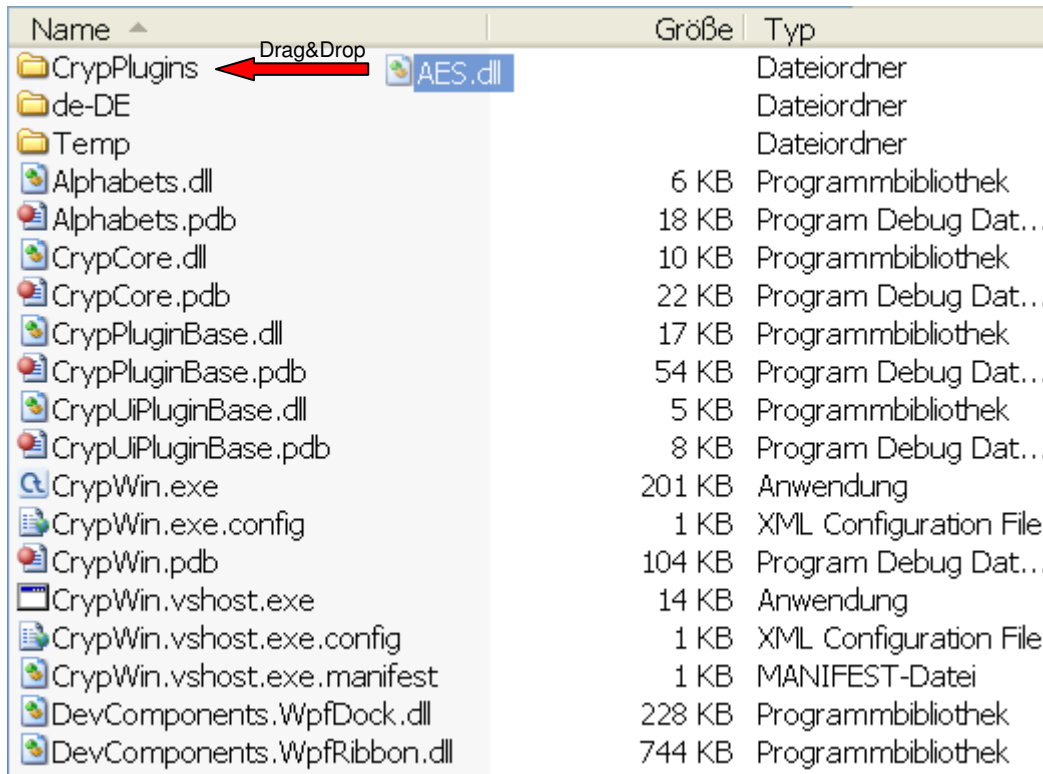
```

As mentioned above, encrypt and decrypt functions are based on the .NET framework cryptography. The functions also measure the start and stop time and send it to the Cryptool status bar. Exceptions are also send to Cryptool.

9. Import the plugin to Cryptool and test it

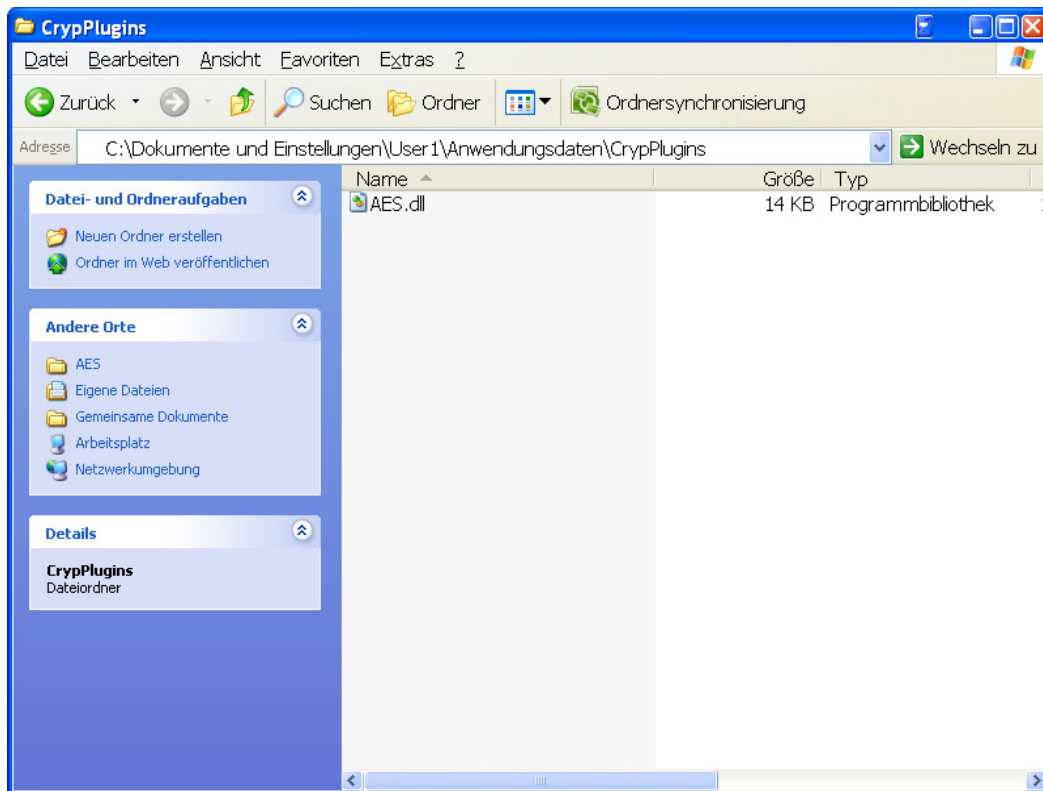
To test our new plugin in Cryptool, there are now three ways to do this:

1. Copy your plugin DLL file in the folder “CrypPlugins” which has to be in the same folder as the Cryptool executable, called “CrypWin.exe”. If necessary, create the folder “CrypPlugins”. This folder is called “Global storage” in the Cryptool architecture. Changes in this folder will take effect for all users on a multi user Windows. Finally restart Cryptool.



Name	Größe	Typ
CrypPlugins		Dateiordner
de-DE		Dateiordner
Temp		Dateiordner
Alphabets.dll	6 KB	Programmbibliothek
Alphabets.pdb	18 KB	Program Debug Dat...
CrypCore.dll	10 KB	Programmbibliothek
CrypCore.pdb	22 KB	Program Debug Dat...
CrypPluginBase.dll	17 KB	Programmbibliothek
CrypPluginBase.pdb	54 KB	Program Debug Dat...
CrypUIPluginBase.dll	5 KB	Programmbibliothek
CrypUIPluginBase.pdb	8 KB	Program Debug Dat...
CrypWin.exe	201 KB	Anwendung
CrypWin.exe.config	1 KB	XML Configuration File
CrypWin.pdb	104 KB	Program Debug Dat...
CrypWin.vshost.exe	14 KB	Anwendung
CrypWin.vshost.exe.config	1 KB	XML Configuration File
CrypWin.vshost.exe.manifest	1 KB	MANIFEST-Datei
DevComponents.WpfDock.dll	228 KB	Programmbibliothek
DevComponents.WpfRibbon.dll	744 KB	Programmbibliothek

2. Copy your plugin DLL file in the folder “CrypPlugins” which is located in your home path in the folder “ApplicationData” and restart Cryptool. This home folder path is called “Custom storage” in the Cryptool architecture. Changes in this folder will only take effect for current user. On a German Windows XP the home folder path could look like:
 „C:\Dokumente und Einstellungen\<User>\Anwendungsdaten\CrypPlugins“



3. You can also import new plugins directly from the Cryptool interface. Just execute CrypWin.exe and select the "Download Plugins" button. An "Open File Dialog" will open and ask where the new plugin is located. After selecting the new plugin, Cryptool will automatically import the new plugin in the custom storage folder. With this option you will not have to restart Cryptool. All according menu entries will be updated automatically. Notice, that this plugin importing function only accepts **signed** plugins.

This third option is a temporary solution for importing new plugins. In the future this will be done online by a web service.

10. Source code and source template

Here you can download the whole source code which was presented in this “Howto” as a Visual Studio **solution**:

<http://cryptool2.vs.uni-due.de/index.php/download>

Here you can download the Visual Studio plugin **template** to begin with the development of a new Cryptool plugin:

<http://cryptool2.vs.uni-due.de/index.php/development>