HowTo – Create a Hash-Plugin for CrypTool 2.0  
using Visual Studio 2008

This document is dedicated for the source code developer of a plugin for the e-learning program CrypTool 2.0.  
It describes step-by-step how you can create a new plugin with Microsoft Visual Studio 2008.

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

The new plugin contains the according hash code and offers its functionality to the CrypTool application using the interface “IHash” from the “Cryptool.CrypPluginBase.Cryptography” class.  
Additionally it delivers the according controls (like buttons, text boxes) and information to the TaskPane and the ContextMenu (icon, caption, descriptions).

|  |  |
| --- | --- |
| Author: | Sebastian Przybylski |
| Date: | 2009-06-19 |
| Version: | 0.9.4 |
| Reviewed by: |  |

Content

[HowTo – Create a Hash-Plugin for CrypTool 2.0 using Visual Studio 2008 1](#_Toc233174603)

[1. Create a new project in VS2008 for your plugin 3](#_Toc233174604)

[2. Select the interface, your plugin wants to serve 4](#_Toc233174605)

[3. Create the classes for the algorithm and for its settings 6](#_Toc233174606)

[3.1 Create the class for the algorithm (MD5) 6](#_Toc233174607)

[3.2 Create the class for the settings (MD5Settings) 8](#_Toc233174608)

[3.3 Add namespace for the class MD5 and the place from where to inherit 9](#_Toc233174609)

[3.4 Add the interface functions for the class MD5 10](#_Toc233174610)

[3.5 Add namespace and interfaces for the class MD5Settings 12](#_Toc233174611)

[3.6 Add controls for the class MD5Settings (if needed) 12](#_Toc233174612)

[4. Select and add an image as icon for the class MD5 14](#_Toc233174613)

[5. Set the attributes for the class MD5 17](#_Toc233174614)

[6. Set the private variables for the settings in the class MD5 20](#_Toc233174615)

[7. Define the code of the class MD5 to fit the interface 21](#_Toc233174616)

[8. Complete the actual code for the class MD5 26](#_Toc233174617)

[9. Keep this in mind 27](#_Toc233174618)

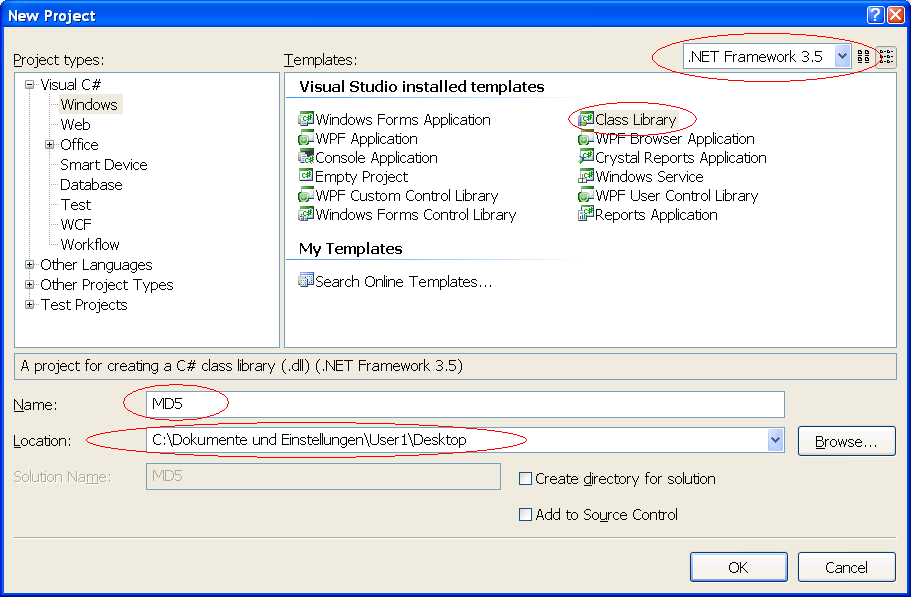
[10. Import the plugin to CrypTool and test it 28](#_Toc233174619)

[11. Source code and source template 29](#_Toc233174620)

[12. Provide a workflow file of your plugin 30](#_Toc233174621)

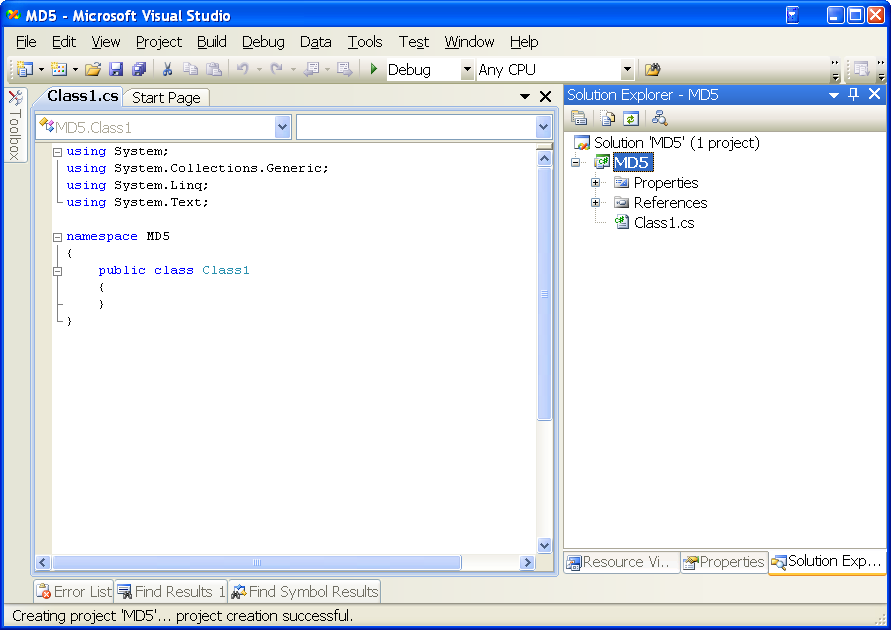
# Create a new project in VS2008 for your plugin

Open Visual Studio 2008 and create a new project:



Select “.NET-Framework 3.5” as the target framework (the Visual Studio Express edition don’t 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: “MD5”), 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.

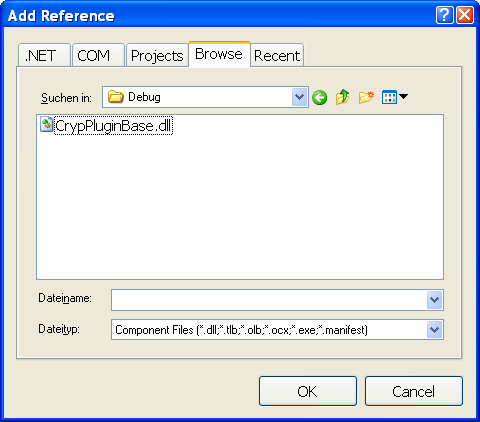
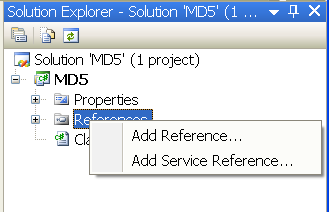
Now your Visual Studio solution could look like this:



# Select the interface, your plugin wants to serve

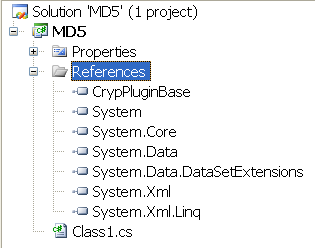
First we have to add a reference to the CrypTool library called “CrypPluginBase.dll” where all necessary CrypTool plugin interfaces are declared.

Make a right click in the Solution Explorer on the “Reference” item and choose “Add Reference”.



Now browse to the path where the library file “CrypPluginBase.dll” 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.

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.

# Create the classes for the algorithm and for its settings

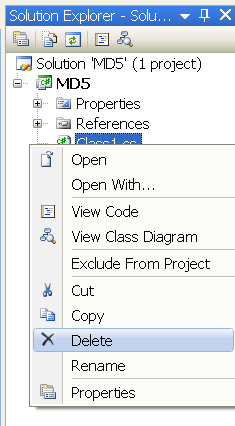
In the next step we have to create two classes. The first class named “MD5” has to inherit from IHash, and the second class named “MD5Settings”from ISettings.

* 1. Create the class for the algorithm (MD5)

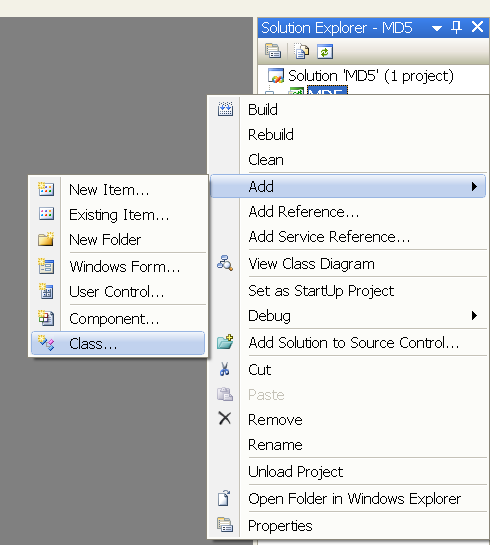
Visual Studio automatically creates a class which has the name “Class1.cs”. There are two ways to change the name to “MD5.cs”:

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

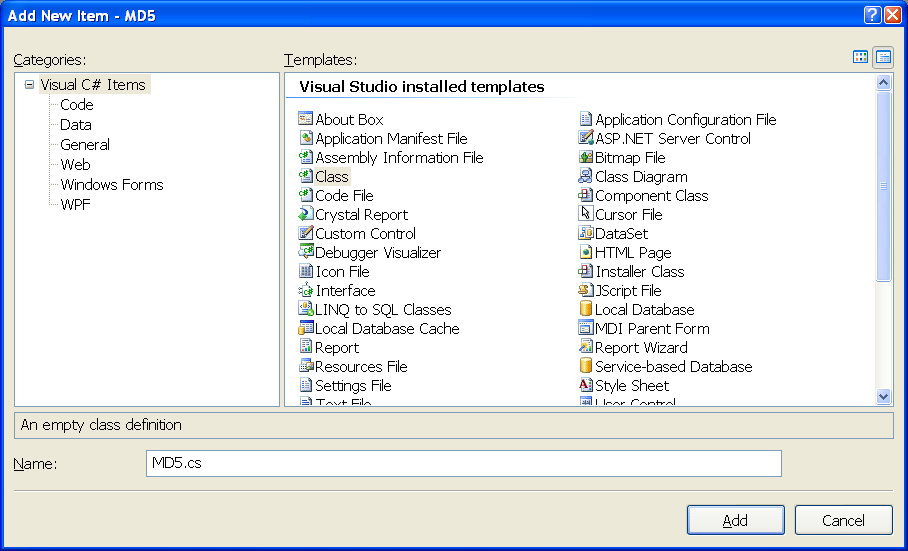
Which one you choose is up to you. We choose the second way as you can see in the next screenshot:



Now make a right click on the project item “MD5” and select “Add->Class…”:

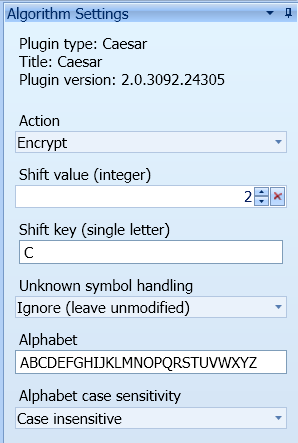


Now give your class a unique name. We call the class as mentioned above “MD5.cs” and make it public to be available to other classes.



* 1. Create the class for the settings (MD5Settings)

Add a second public class for ISettings in the same way. We call the class “MD5Settings”. The settings class provides the necessary information about controls, captions and descriptions and default parameters for 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.



* 1. Add namespace for the class MD5 and the place from where to inherit

Now open the “MD5.cs” file by double clicking on it at the Solution Explorer and include the necessary namespaces to the class header by typing in the according “using” statement. The CrypTool 2.0 API provides the following namespaces:

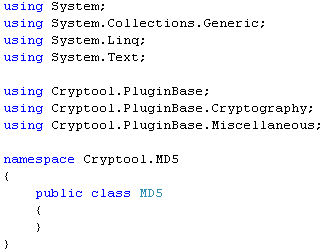
* Cryptool.PluginBase = interfaces like IPlugin, IHash, ISettings, attributes, enumerations, delegates and extensions.
* Cryptool.PluginBase.Analysis = interface for the crypto analysis plugins like “Stream Comparator”
* Cryptool.PluginBase.Cryptography = interface for all encryption and hash algorithms like AES, DES or MD5 hash
* Cryptool.PluginBase.Editor = interface for editors you want to implement for CrypTool 2.0 like the default editor
* Cryptool.PluginBase.Generator = interface for generators like the random input generator
* Cryptool.PluginBase.IO = interface for CryptoolStream, input and output plugins like text input, file input, text output and file output
* Cryptool.PluginBase.Miscellaneous = provides all event helper like GuiLogMessage or PropertyChanged
* Cryptool.PluginBase.Tool = interface for all foreign tools which CrypTool 2.0 has to provide and which does not exactly support the CrypTool 2.0 API
* Cryptool.PluginBase.Validation = interface which provides method for validation like regular expression

In this case we want to implement a MD5 hash which means we need to include the namespaces “Cryptool.PluginBase” to provide “ISettings” for the MD5Settings class, the “Cryptool.PluginBase.Cryptography” to provide “IHash” for the MD5 class and “Cryptool.PluginBase.Miscellaneous” to use the entire CrypTool event handler.

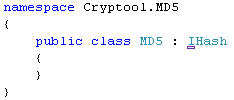
It is important to define a new default namespace of our public class (“MD5”). In CrypTool the default namespace is presented by “Cryptool.[name of class]”. Therefore our namespace has to be defined as follows: “Cryptool.MD5”.

Up to now the source code should look as you can see below:

needed CrypTool namespaces

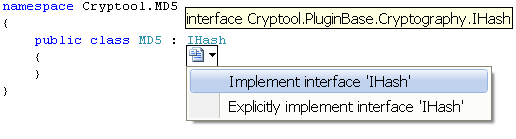


Next let your class “MD5” inherit from IHash by inserting of the following statement:



* 1. Add the interface functions for the class MD5

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



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

Your code will now look like this:



* 1. Add namespace and interfaces for the class MD5Settings

Let’s now take a look at the second class “MD5Settings” by double clicking at the “MD5Settings.cs” file at 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 “ISettings” analogous as seen before at the MD5 class. Visual Studio will here also automatically place code from the CrypTool interface if available.

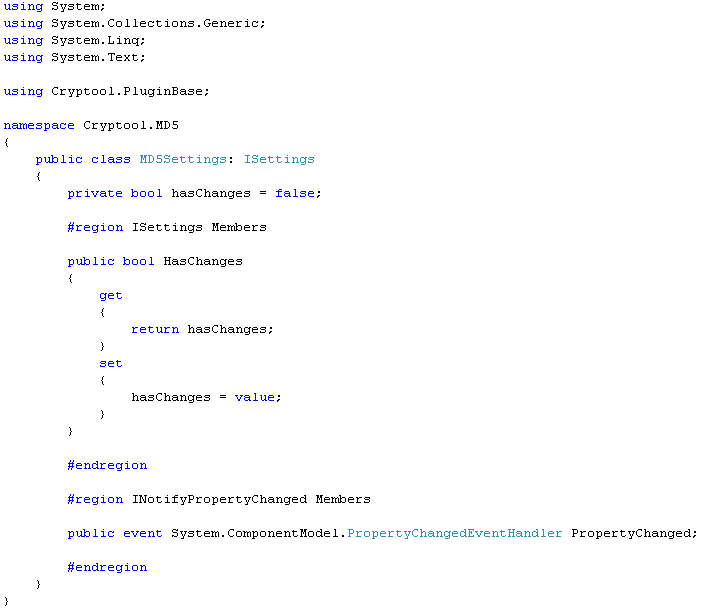


* 1. Add controls for the class MD5Settings (if needed)

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. Our MD5 hash algorithm doesn’t have any kind of settings, so we will let this class empty. In this case the **TaskPane** will be empty in CrypTool. The only modification we have to do is to finish the implementation of the “HasChanges” property to avoid any “NotImplementedException”. If you want to provide a TaskPane with some kind of controls, take a look at the encryption HowTo which you can download at:

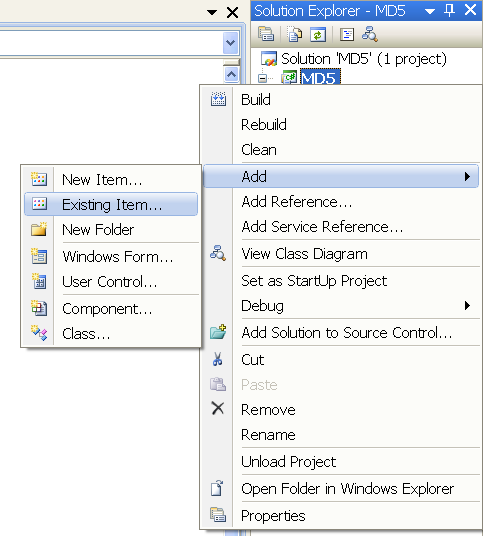
<http://cryptool2.vs.uni-due.de/downloads/howto/howto_encryptionplugin.pdf>

The code should look now like this:

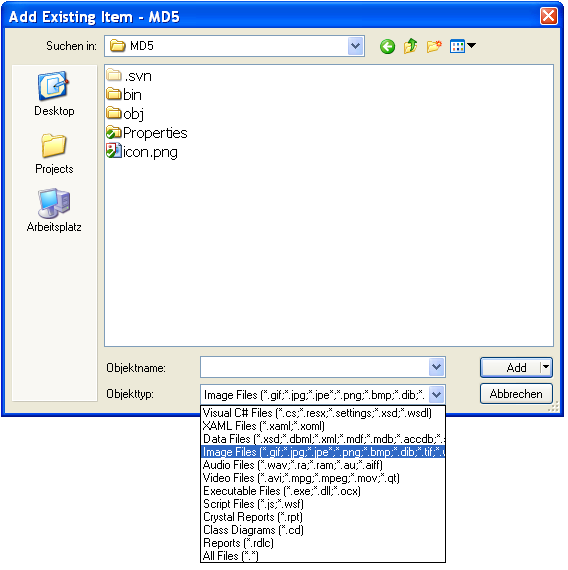


# Select and add an image as icon for the class MD5

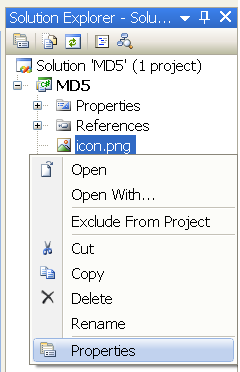
Before we go back to the code of the MD5 class, we have to add an icon image to our project, which will be shown in the CrypTool **ribbon bar** or/and **navigation pane**. Therefore, make a right click on the project item “MD5” within the Solution Explorer, and select “Add->Existing Item…”:



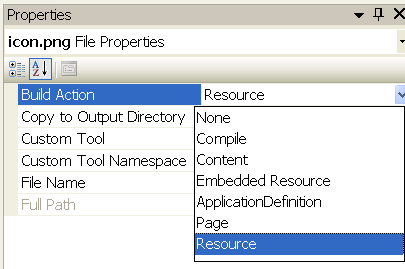
Then select “Image Files” as file type, and choose the icon for your plugin:



Finally we have to set the icon as a “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 “Resource” (not embedded resource):



# Set the attributes for the class MD5

Now let’s go back to the code of the MD5 class (“MD5.cs” file). First we have to set the necessary attributes for our class. This attributes are used to provide additional information for the Cryptool 2.0 environment. If not set, your plugin won’t show up in the GUI, even if everything else is implemented correctly.

Attributes are used for **declarative** programming and provide meta data, that can be attached to the existing .NET meta data , like classes and properties. Cryptool provides a set of custom attributes, that are used to mark the different parts of your plugin.

*[Author]*

The first attribute called “Author” is optionally, which means we are not forced to define this attribute. It provides the additional information about the plugin developer. We set this attribute to demonstrate how it has to look in case you want to provide this attribute.



As we can see above the author attribute takes four elements of type string. These elements are:

* Author = name of the plugin developer
* Email = email of the plugin developer if he wants to be contact
* Institute = current employment of the developer like University or Company
* Url = the website or homepage of the developer

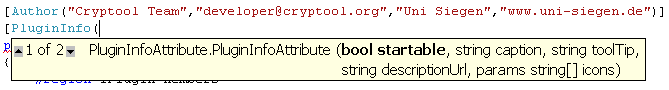
All this elements are also optionally. The developer decides what he wants to publish.

Our author attribute should look now as you can see below:



*[PluginInfo]*

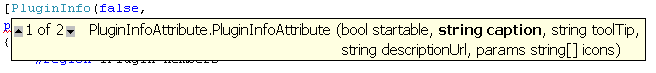
The second attribute called “PluginInfo” provides the necessary information about the plugin like caption and tool tip. This attribute is mandatory. The attribute has the definition as you can see below:



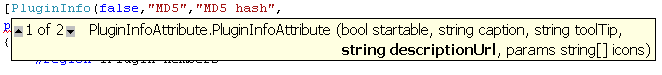
This attribute expects the following elements:

* startable =   
  Set this flag to true only if your plugin is some kind of input- or generator plugin (probably if your plugin just has outputs and no inputs). In all other cases use false here. This flag is important. Setting this flag to true for a non input/generator plugin will result in unpredictable chain runs.
* caption =   
  from type string, the name of the plugin (e.g. to provide the button content)
* toolTip = from type string, description of the plugin (e.g. to provide the button tool tip)
* descriptionUrl = from type string, define where to find the whole description files (e.g. XAML files)
* icons = from type string array, which provides all necessary icon paths you want to use in the plugin (e.g. the plugin icon as seen above)

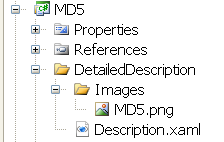
In this case the first parameter called “startable” has to be set to “false”, because our hash algorithm is not an input- or generator plugin.



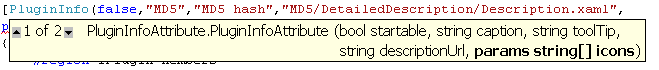
The next two parameters are needed to define the plugin’s name and its description:



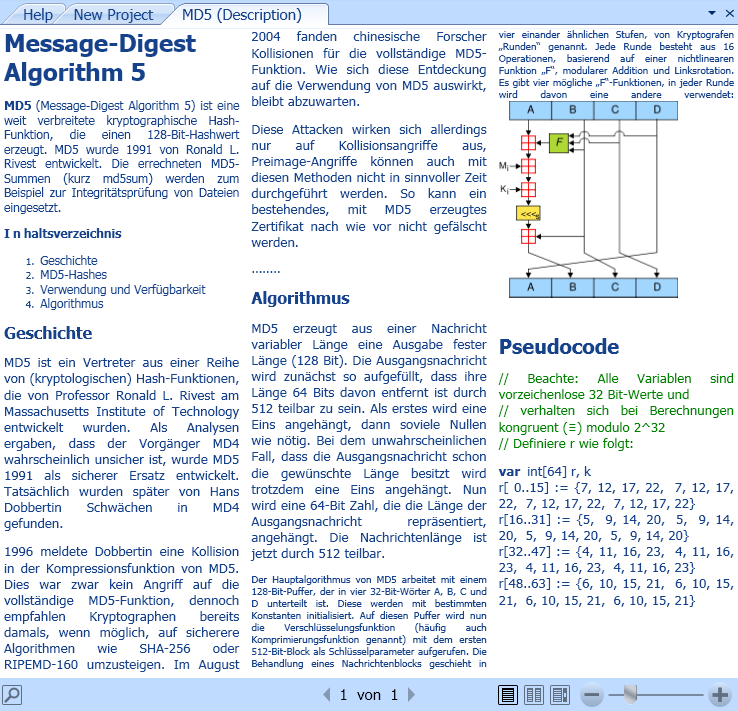
The fourth element defines the location path of the description file. The parameter is made up by <Assembly name>/<filename> or <Assembly name>/<Path>/<file name> if you want to store your description files in a separate folder. The description file has to be of type XAML. In our case we create a folder called “DetailedDescription” and store our XAML file there with the necessary images if needed. How you manage the files and folders is up to you. This folder could now look as you can see below:



Accordingly the attribute parameter has to be set to:

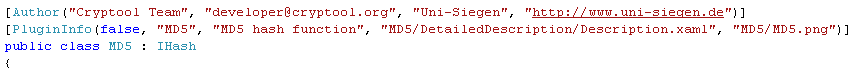


The detailed description could now look like this in CrypTool (right click plugin icon on workspace and select “Show description”):



The last parameter tells CrypTool the names of the provided icons. This parameter is made up by <Assembly name>/<file name> or <Assembly name>/<Path>/<file name>.

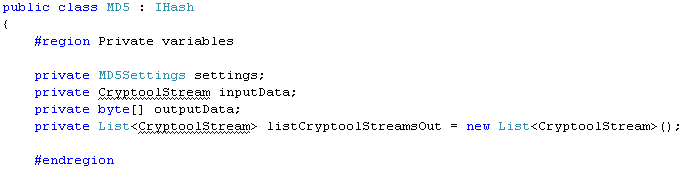
The most important icon is the plugin icon, which will be shown in CrypTool in the ribbon bar or navigation pane (This is the first icon in list, so you have to provide at least one icon for a plugin). As named above how to add an icon to the solution accordingly we have to tell CrypTool where to find the icon by setting this parameter as you can see below:



You can define further icon paths if needed, by adding the path string separated by a comma.

# Set the private variables for the settings in the class MD5

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

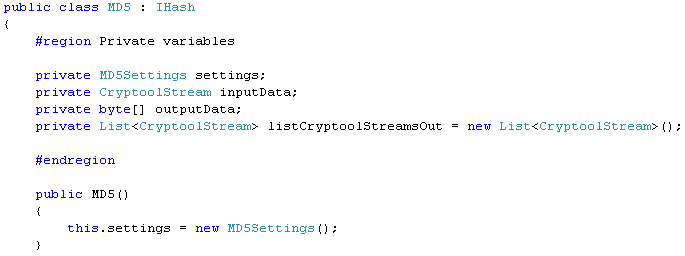


Please notice the sinuous line at the type “CryptoolStream” of the variable inputData and the list listCryptoolStreamsOut. As explained above, you have to include the namespace “Cryptool.PluginBase.IO” to use the CrypTool own stream type. Also notice, that we use streams for input and output to perform a file handling in CrypTool with huge files. The class “IHash” provides an additional byte array output.

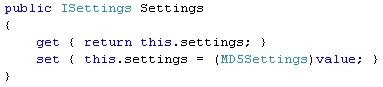
# Define the code of the class MD5 to fit the interface

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

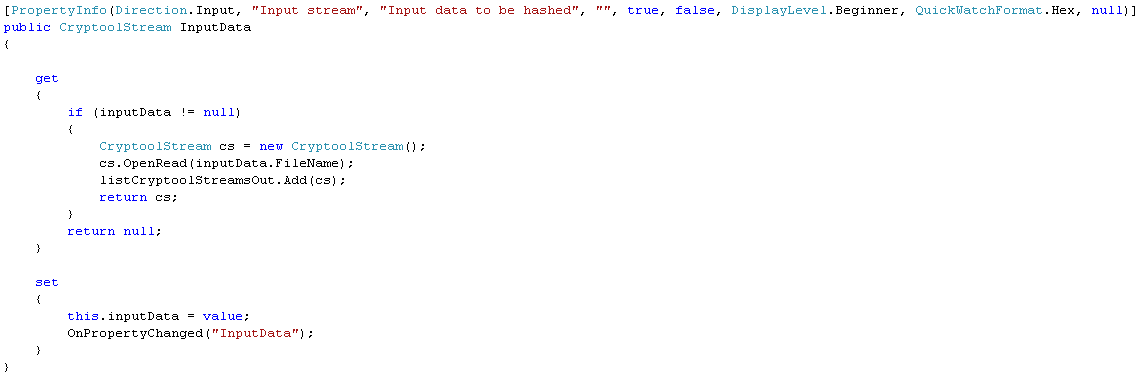
First we add a constructor to our class where we can create an instance of our settings class:



Secondly, we have to implement the property “Settings” defined in the interface:



Thirdly we have to define two properties with their according attributes. This step is necessary to tell Cryptool that these properties are input/output properties used for data exchange with other plugins. First we define the “InputData” property getter and setter:



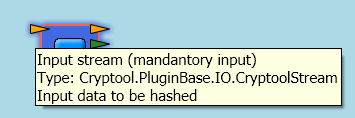
In the getter we first prove if the input data is not null. If input data is filled, we declare a new CryptoolStream, open it and add it to our list where all output streams are stored. Finally the new stream will be returned.

The setter sets the new input data and announces the data to the Cryptool 2.0 environment by using the expression “OnPropertyChanged(“<Property name>”).

This step is important (especially for output properties), because this is the only way to inform Cryptool 2.0 of the data update.

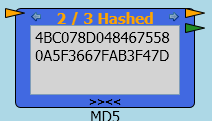
As you can see, the “InputData” property also needs an attribute named “PropertyInfo”. The attribute-constructor expects 9 parameters:

* direction = defines if this property is an input or output property like input or output data
  + Direction.Input
  + Direction.Output
* caption = caption of the property (e.g. shown at the input on the dropped icon in the editor), see below:



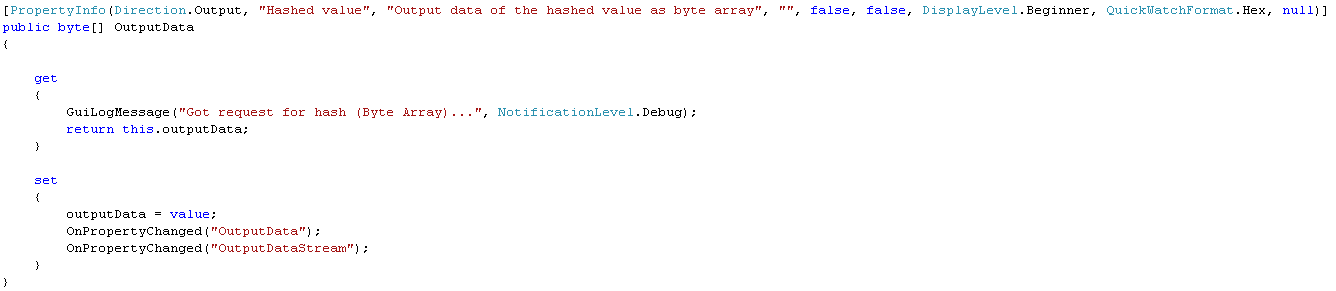
* tooltip = tooltip of the property (e.g. shown at the input arrow on the dropped icon in the editor), see above
* descriptionUrl = not used right now
* mandatory = this value can be set to true or false and defines if the user is forced to set the input data (it only applies to Direction.input, when using Direction.output it is ignored). If set to true, there has to be an input connection that provides data. If no input data is provided for mandatory input, your plugin will not be executed. If set to false, the plugin may be executed, even if this input is not connected.
* hasDefaultValue = if this flag is set, CrypTool treat this plugin as though the input has already input data.
* DisplayLevel = define in which display levels your property will be shown in CrypTool. CrypTool provides the following display levels:
  + DisplayLevel.Beginner
  + DisplayLevel.Experienced
  + DisplayLevel.Expert
  + DisplayLevel.Professional
* QuickWatchFormat = defines if the content of the property has to be shown in the quick watch. CrypTool accepts the following quick watch formats:
  + QuickWatchFormat.Base64
  + QuickWatchFormat.Hex
  + QuickWatchFormat.None
  + QuickWatchFormat.Text

A quick watch could look like this:

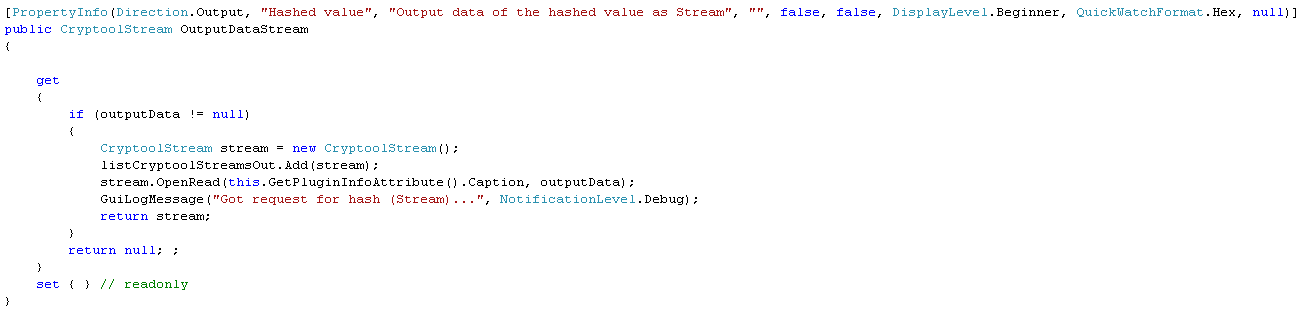


* quickWatchConversionMethod = this string points to a conversion method; the most plugins can use a “null” value here, because no conversion is necessary. The QuickWatch function uses system “default” encoding to display data. So only if your data is in some other format, like Unicode or UTF8, you have to provide a conversion method. The method header has to look like this:  
  object YourMethodName(string PropertyNameToConvert)

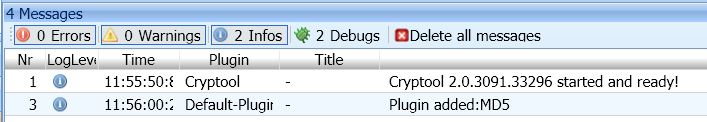
The output data property could look like this:



You can also provide additionally output data types if you like. For example we provide also an output data of type CryptoolStream:



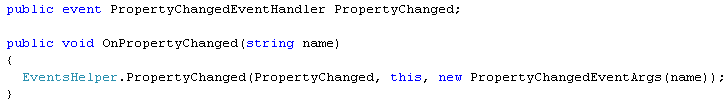
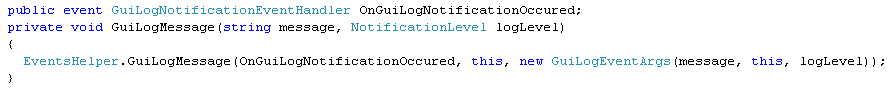
Notice the method “GuiLogMessage” in the source codes above. This method is used to send messages to the CrypTool status bar. This is a nice feature to inform the user what your plugin is currently doing.



The method takes two parameters which are:

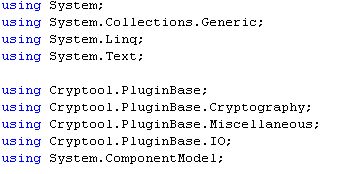
* Message = will be shown in the status bar and is of type string
* NotificationLevel = to group the messages to their alert level
  + NotificationLevel.Error
  + NotificationLevel.Warning
  + NotificationLevel.Info
  + NotificationLevel.Debug

As we can recognize we have two methods named “OnPropertyChanged” and “GuiLogMessage” which are not defined. So we have to define these two methods as you can see below:



To use the “PropertyChangedEventHandler” you have to include the namespace “System.ComponentModel”.

Our whole included namespaces looks now like this:



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

* PresentationCore
* PresentationFramework
* WindowsBase

The function “Presentation” is served for the PluginBase if a plugin developer wants to provide his own graphic visualization of the plugin algorithm which has to been shown in CrypTool. Take a look at the PRESENT plugin to see how a graphic visualization can be realized.

# Complete the actual code for the class MD5

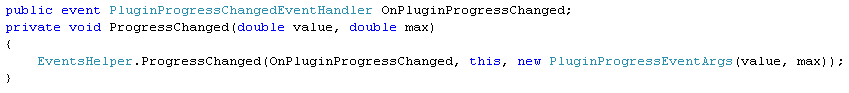
Up to now, the plugin is ready for the CrypTool base application to be accepted and been shown correctly in the CrypTool menu. What we need now, is the implementation of the actual algorithm in the function “Execute()” which is up to you as the plugin developer.

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



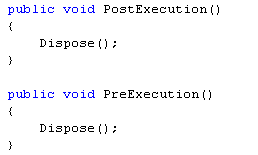
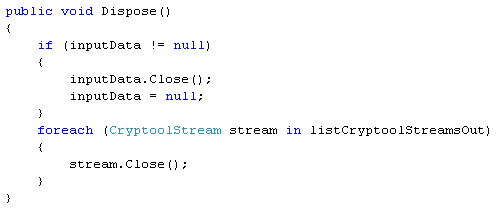
Certainly you have seen the unknown method “ProgressChanged” which you can use to show the current algorithm process as a progress on the plugin icon.

To use this method you also have to declare this method to afford a successful compilation:

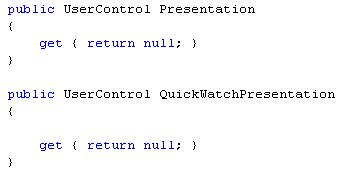


1. Keep this in mind

Be sure you have closed and cleaned all your streams before working with or after disposing your plugin:



To avoid unnecessary exceptions you should replace the automatically placed “NotImplementedException” expressions by the compiler to “null” if you want to stay these methods empty:



1. Import the plugin to CrypTool and test it

To test our new plugin in CrypTool after you have build it, there are now three ways to do this:

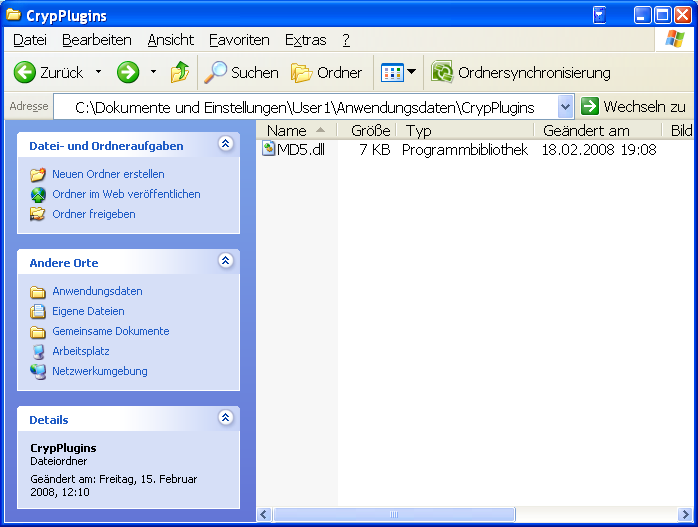
1. Copy your plugin DLL file in the folder “CrypPlugins” which has to be in the same folder as the CrypTool executable, called “CrypWin.exe”. If necessary, create the folder “CrypPlugins”. This folder is called “Global storage” in the CrypTool architecture. Changes in this folder will take effect for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

Drag&Drop



1. 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“ and in Vista the path will look like „C:\Users\<user>\Application Data\CrypPlugins“.



1. 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.

1. Source code and source template

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

username: anonymouspassword: not required

<https://www.cryptool.org/svn/CrypTool2/trunk/CrypPlugins/MD5/>

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/downloads/template/hashplugin.zip>

1. Provide a workflow file of your plugin

Every plugin developer should provide a workflow file which shows his algorithm working in CrypTool2. You will automatically create a workflow file by saving your project which was created on CrypTool2 work space. Here is an example how a workflow could looks like:

