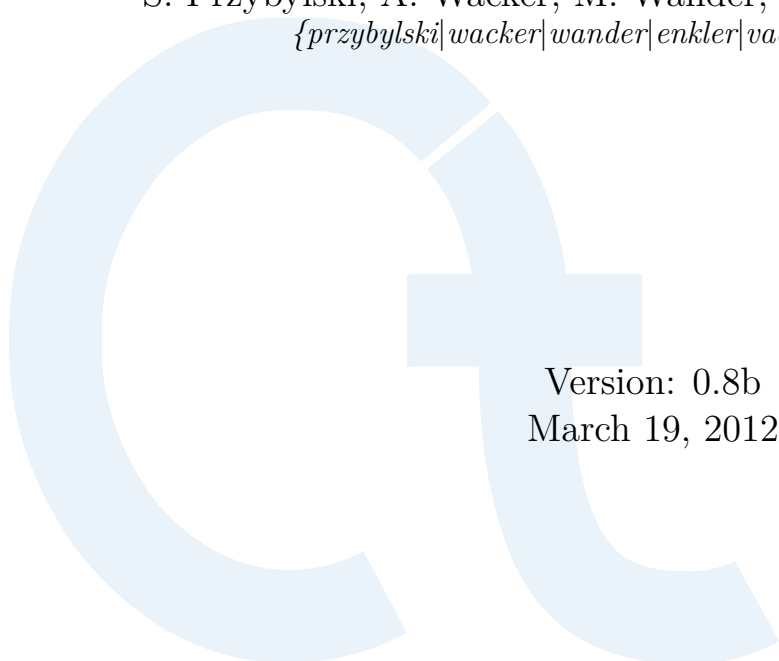CrypTool 2.0

Plugin Developer Manual

– How to build your own plugins for CrypTool 2.0 –

S. Przybylski, A. Wacker, M. Wander, F. Enkler and P. Vacek

{przybylski|wacker|wander|enkler|vacek}@cryptool.org

Version: 0.8b

March 19, 2012

CrypTool 2 is the successor of the well-known e-learning platform for cryptography and cryptanalysis CrypTool[1], which is used for educational purposes at schools and universities as well as in companies and agencies. As of January 2010, CrypTool 2 consists of over 7000 lines of C# code in the core application and over 250,000 lines of C# code in about 100 plugins.

CrypTool 2 is built using the following technologies and development tools:

- .NET (a modern software framework from Microsoft with solutions to common programming problems)

- C# (a modern object-oriented programming language, comparable to Java)

- WPF (a modern vector-based graphical subsystem for rendering user interfaces in Windows-based applications)

- Visual Studio 2010 (a development environment)

- Subversion (a source code and documentation version management system)

- trac (a lightweight web-based software project management system)

This document is intended for plugin developers who want to contribute a new plugin to CrypTool 2 which implements a cryptographic algorithm or similar functionality. Please note that CrypTool 2 is an alive project in development. Certain information may be outdated or missing. If you want to stay up-to-date, we recommend checking out the CrypTool 2 development wiki[2] and website[3].

---

[1] http://www.cryptool.org/
[2] https://www.cryptool.org/trac/CrypTool2/wiki
[3] http://www.cryptool.org/cryptool2/

Contents

List of Figures

1 Developer Guidelines

CrypTool 2.0 is built upon state-of-the-art technologies such as .NET 4.0 and the Windows Presentation Foundation (WPF). Before you can start writing code and adding to the development of the project, a few things need to be considered. To make this process easier, please read through this document[1] and follow the instructions closely. This document exists to help get you started by showing you how CrypTool 2 plugins are built in order to successfully interact with the application core. We have tried to be very thorough, but if you encounter a problem or error that is not described here, please let us know[2]. Not only do we want to help get you up and running, but we also want to add the appropriate information to this guide for the benefit of other future developers.

In this first chapter we will describe all steps necessary in order to compile CrypTool 2 on your own computer. This is always the first thing you need to do before you can begin developing your own plugins and extensions. The basic steps are:

- Getting all prerequisites and installing them

- Accessing and downloading the source code with SVN

- Compiling the latest version of the source code

1.1 Prerequisites

Since CrypTool 2 is based on Microsoft .NET 4.0, you will need a Microsoft Windows environment. (Currently no plans exist for porting this project to Mono or other platforms.) We have successfully tested with **Windows XP**, **Windows Vista** and **Windows 7**.

Since you are reading the developer guidelines, you probably want to develop something. Hence, you will need a development environment. In order to compile our sources you need **Microsoft Visual Studio 2010** or **Microsoft Visual C# 2010 Express**. Make sure to always install the latest service packs for Visual Studio.

In order to run or compile our source code you will need at least the **Microsoft .NET 4.0**. Usually the installation of Visual Studio also installs the .NET framework, but if you do not have the latest version, you can get it for free from Microsoft's website. Once the framework has been installed, your development environment should be ready for our source code.

---

[1]Download the most current version of this document here.

[2]Contact us here or write an email to mailto:ct2contact@cryptool.org

1.2 Access the Subversion (SVN) repository

Next you will need a way of accessing and downloading the source code. For the CrypTool 2 project we use **Subversion (SVN)** for version control, and hence you will need an SVN client, i.e. **TortoiseSVN**, **AnkhSVN** or the **svn commandline from cygwin**, to access our repository. It does not matter which client you use, but if SVN is new to you, we suggest using TortoiseSVN, since it offers a handy, straightforward Windows Explorer integration. We will guide you through how to use TortoiseSVN, although you should be able to use any SVN client in a similar fashion.

1.2.1 Check out the sources

First, download and install TortoiseSVN. This will require you to reboot your computer, but once it is back up and running, create a directory (for instance, *CrypTool2*) somewhere on your computer for storing the local working files. Right-click on this directory; now that TortoiseSVN has been installed, you should see a few new items in the context menu (Figure 1.1). Select *SVN Checkout*.
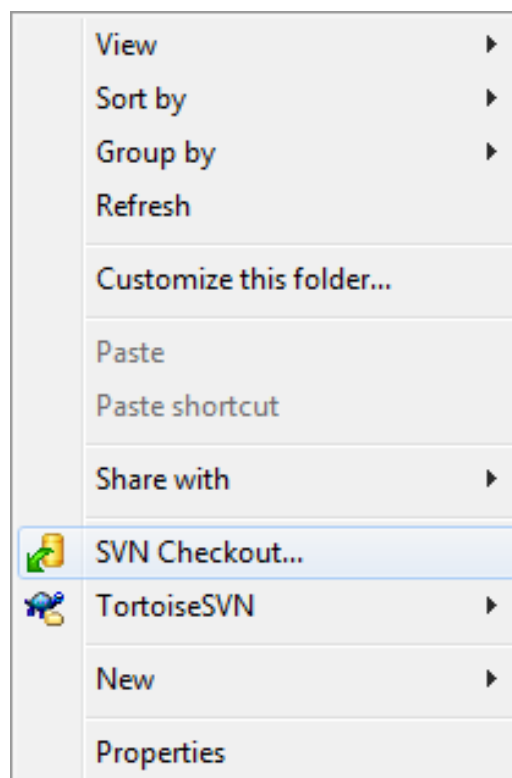

Figure 1.1: Selecting *SVN Checkout* from the context menu after installing TortoiseSVN.

A window will now appear that will ask you for the URL of the repository that you would like to access. Our code repository is stored at `https://www.cryptool.org/svn/CrypTool2/trunk/`, and this is what you should enter in the appropriate field. The *Checkout directory* should already be filled in correctly with your new folder, and you shouldn't need to change any other options.
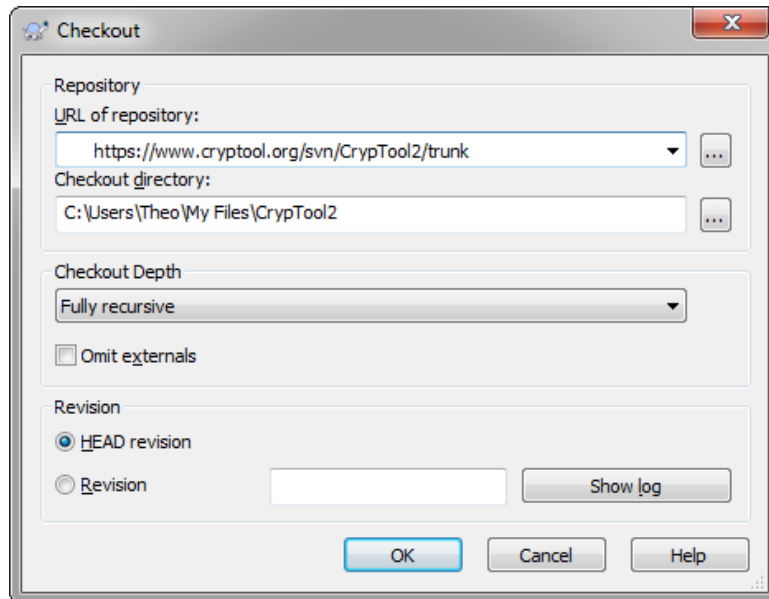

Figure 1.2: Checking out the CrypTool 2 repository.

Then just hit *OK*. You may be asked to accept a certificate (which you should accept), and you will certainly be asked for login information. If you are a registered developer, you should have already been given a username and password, and you should enter them here. (These are the same username and password that you can use for the CrypTool 2 development wiki.) If you are a guest and only need read-only access, you can use "anonymous" as the username and an empty password. Mark the checkbox for saving your credentials if you don't want to enter them every time you work with the repository. (Your password will be saved on your computer.) Finally, hit *OK*, and the whole CrypTool 2 repository will begin downloading into your chosen local directory.

Since CrypTool 2 is a collaborative project with many developers, changes are made to the repository rather frequently. You should maintain a current working copy of the files to ensure your interoperability with the rest of the project, and thus you should update to the latest version as often as possible. You can do this by right-clicking on any directory within the working files and choosing *SVN Update* from the context menu.

A TortoiseSVN tutorial can be found at `http://www.mind.ilstu.edu/research/robots/iris5/developers/documentation/svntutorial/`.

1.2.2 Adjust the SVN settings

If you are a registered developer, you can commit your file changes to the public CrypTool 2 repository. However, before you do, you should edit your settings to make sure that you only check in proper source code. First, bring up the TortoiseSVN settings window:
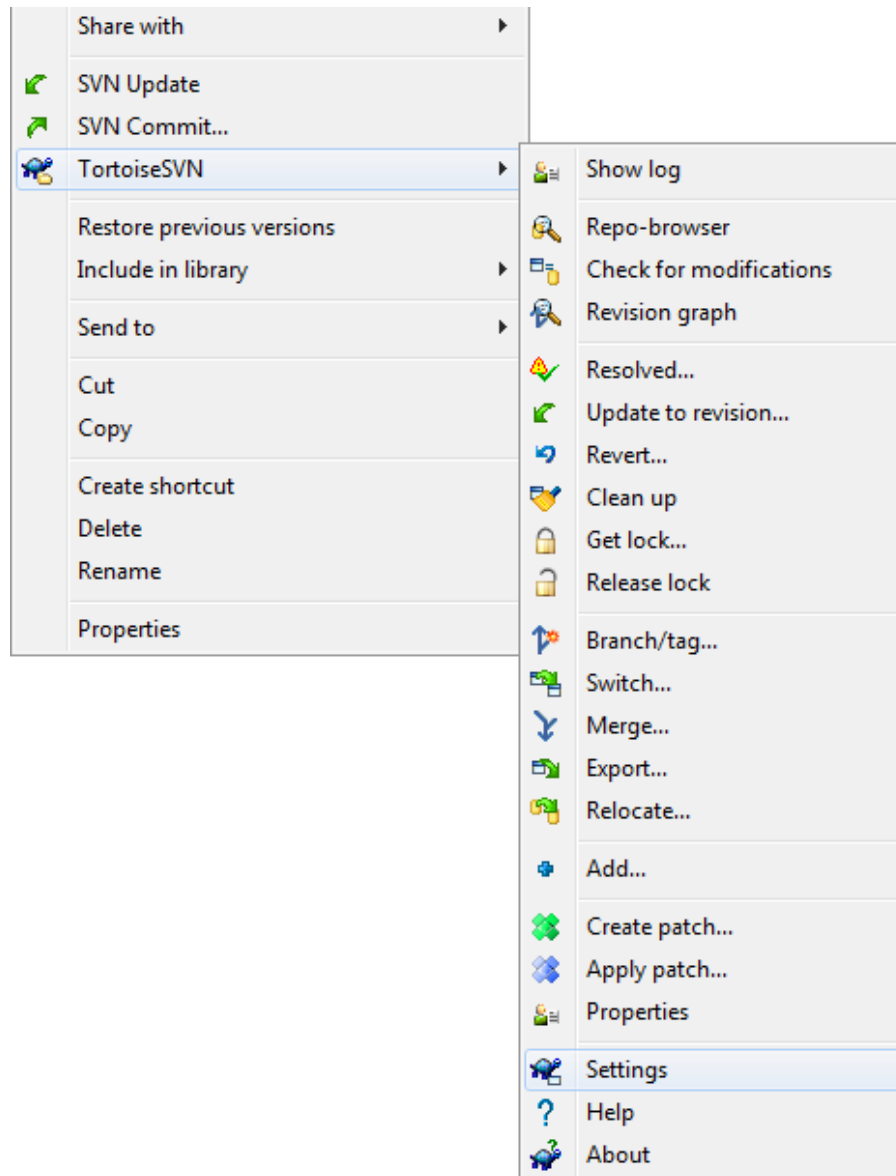
Figure 1.3: Getting to the TortoiseSVN settings.

The settings window will look something like this:
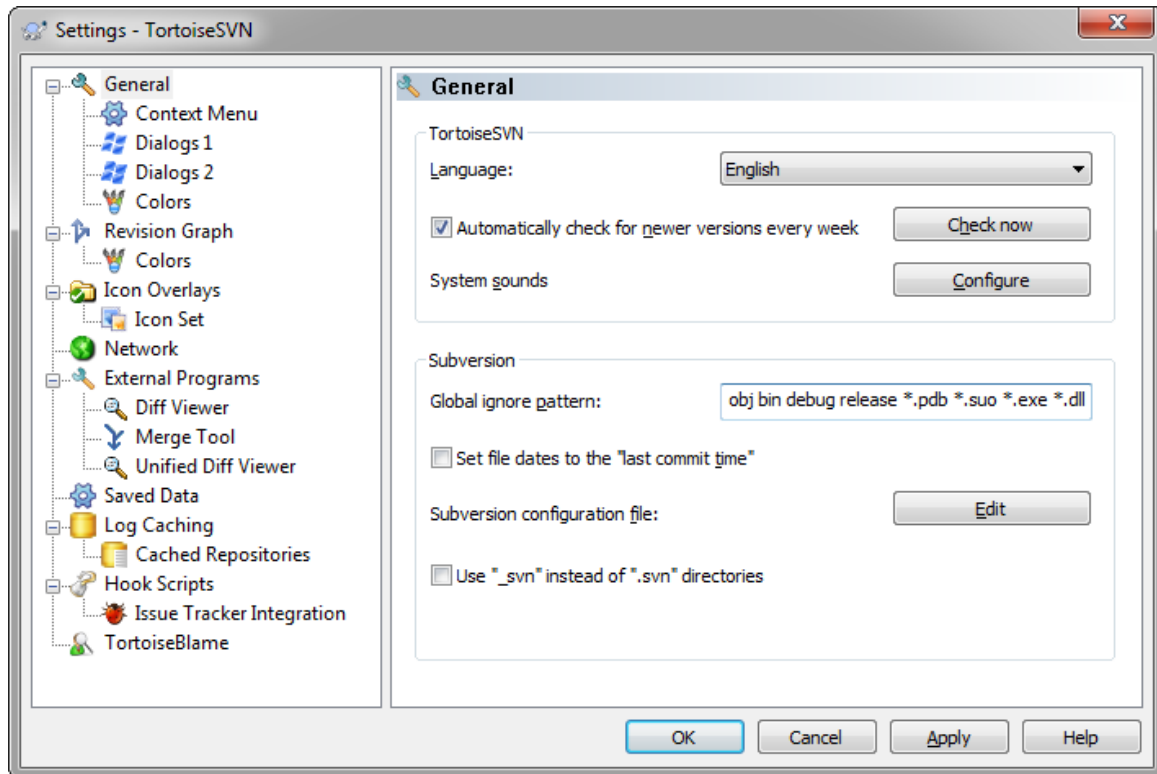
Figure 1.4: The TortoiseSVN settings window with the proper ignore pattern.

Then in the *Global ignore pattern* field, please enter the following text:

*obj bin debug release *.pdb *.suo *.exe *.dll *.aux *.dvi *.log *.bak *.bbl *.blg *.user*

You are free to also leave in any default pattern text or to write your own additions; this pattern serves simply to tell TortoiseSVN what kinds of files to ignore. You can now click *OK* to save your settings and close the window.

1.2.3 Commit your changes

Once you start writing code and developing your plugin, you should check your work into the project repository. If you are reading this document in sequence, you are probably not ready to do this, but while we are on the topic of SVN we will describe the process. To upload your changes, right-click on a directory within the working files that contains your changes and select *SVN Commit* from the context menu:
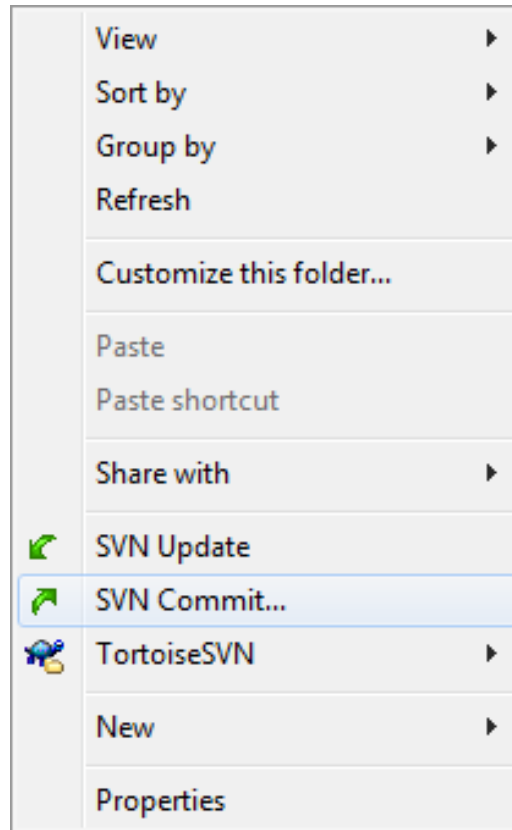

Figure 1.5: Selecting *SVN Commit* from the context menu.

When you commit your code, you must enter a comment to describe what you have changed. *Meaningful descriptions* will help other developers comprehend your updates. You can also select exactly which files you want to check in. The ignore pattern that we recommended should prevent most undesirable files from being included, but double-check to make sure everything you want to upload is included and nothing more. In general, you should never check in compiled or automatically generated files. For example, do not check in the entire `bin\` and `obj\` directories that Visual Studio generates. The server will reject your commits if you try to do so. You should commit your sources to our SVN repository as often as you can, even if your work is unfinished or there are bugs. However, your committed code should not break any part of the existing project, so please make sure the public solution still compiles and runs successfully.
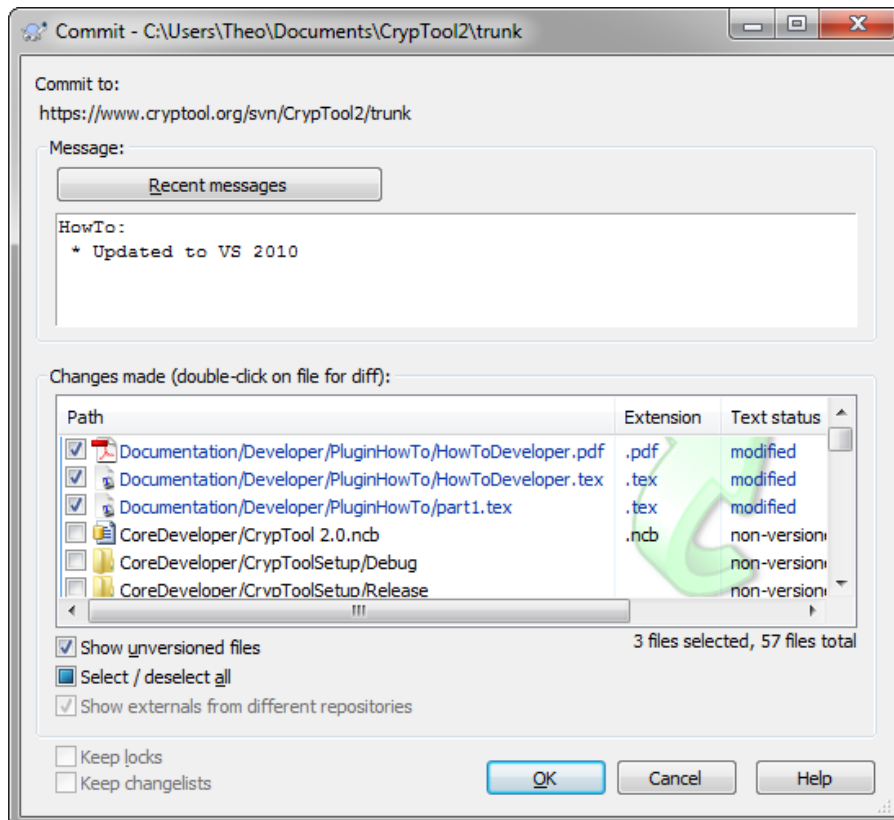
Figure 1.6: Providing comments for a commit.

You can use the SVN comments to link to your changes to a particular issue or bug ticket on the CrypTool 2 development wiki. (The list of active tickets can be found here.) The following commands are supported (note that there are multiple variations of each command that are functionally identical):

```
closes, fixes:
```
The specified ticket will be closed and the contents of this commit message will be added to its notes.

```
references, refs, addresses, re:
```
The contents of this commit message will be added to the specified ticket's notes, but the status will be left unaltered.

You can apply the commands to multiple tickets simultaneously. The command syntax is as follows (again note that there are multiple variations that are functionally identical):

```
command #1
command #1, #2
command #1 & #2
command #1 and #2
```

You can also use more than one command in a message if necessary. For example, if you want to close tickets #10 and #12, and add a note to #17, you could type the following:

```
Changed blah and foo to do this or that. Fixes #10 and #12, and refs #17.
```

The comments can also be used to override the ignore pattern that the server is designed to block. However, please do not do this unless you are absolutely sure that you know what you are doing. If you are, you must use the *override-bad-extension* command and provide an explicit list of the file and directory names that you want to upload that need to override the ignore pattern. For example, if you want to check in a library file named *someLib.dll*, you must write something like the following:

```
This library is referenced by project xy.

override-bad-extension: someLib.dll
```

Note that any text after the colon and the whitespace will be treated as the file name. Therefore, do not use quotation marks and do not write any text after the file name.

1.3 Compile the sources with Visual Studio 2010

By this point you should have checked out a copy of the entire CrypTool 2 repository. Compiling is pretty easy; just go to the `trunk\` directory and open the ***CrypTool 2.0.sln*** Visual Studio solution. The Visual Studio IDE should open with all the working plugin components nicely arranged. If you are now starting Visual Studio for the first time, you will have to choose your settings. Just select either *most common* or *C#* — you can change this at any time later. On the right side is the project explorer, where you can see all the subprojects included in the solution. Look for the project ***CrypWin*** there and make sure it is selected as startup project (right-click on it and select *Set as StartUp Project* from the context menu). Then click *Build → Build Solution* in the menubar to start the build process.

You may have to wait a while for the program to compile. Once it is finished, select *Debug → Start Debugging*. CrypTool 2 should now start for the first time with your own compiled code. Presumably you have not changed anything yet, but you now have your own build of all the components. If the program does not compile or start correctly, please consult our FAQ and let us know if you found a bug.

If you are a **core developer**, you can use the ***CrypTool 2.0.sln*** solution from the `CoreDeveloper\` directory (which will *not* be visible to you if you are not a core developer). We often refer to this solution as the core-developer solution or as the internal solution. The core-developer solution is used for building the nightly build. Thus, if a plugin should become delivered with the nightly builds it needs to be added to this solution.

1.4 Compiling the sources with Visual C# 2010 Express

With Visual C# Express the build process is basically the same as with Visual Studio. When opening the solution file, you will receive two error messages. The first is because Visual C# does not support

solution folders and shows all plugin projects as a flat list in the solution explorer. However, this is merely a visual defect. The second error message occurs because Visual C# does not support unit tests and thus is not able to load the project *UnitTests*. Again, this does not interfere with opening, writing, compiling, running, or debugging any other plugins.

1.5 Download the plugin template

Before you can start implementing a new plugin, you will need to download the CrypTool 2 plugin template. The most current version of this template is located in our CrypTool 2 repository, and can be downloaded via our wiki interface at `https://www.cryptool.org/trac/CrypTool2/browser/trunk/` `Documentation/CrypPluginTemplate/CrypTool%202.0%20Plugin.zip`. Save the template zip file in your documents folder in the subdirectory `Visual Studio 2010\Templates\ProjectTemplates\`. (This applies to both Visual Studio and Visual C# Express.) Do not unpack the zip file.
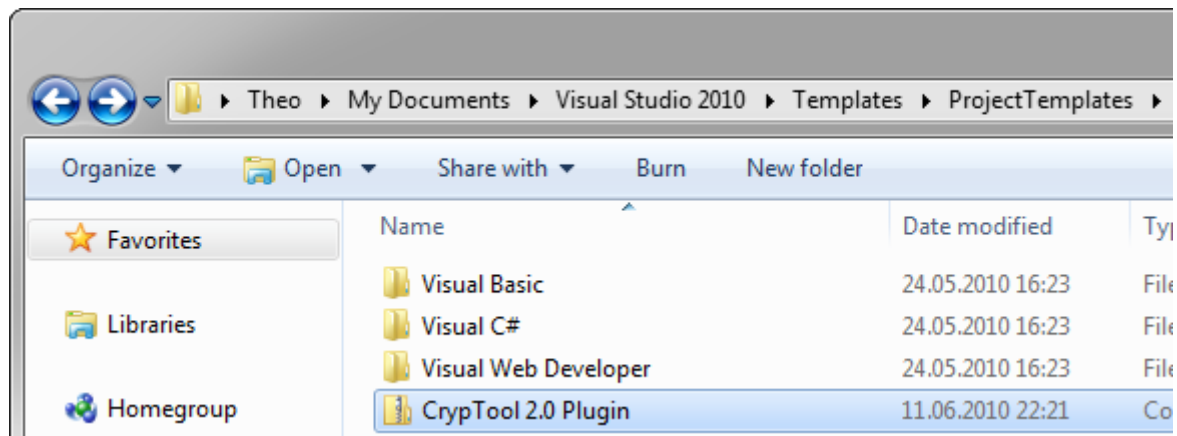

Figure 1.7: Saving plugin template.

2 Plugin Implementation

In this chapter we provide step-by-step instructions for implementing your own CrypTool 2 plugin. We shall assume that you have already retrieved the CrypTool 2 source code from the SVN repository, set up Visual Studio 2010 or Visual C# 2010 Express to build CrypTool 2, and you have placed the plugin template in the right place.

2.1 Create a new project

Open the CrypTool 2 solution, right click in the solution explorer on *CrypPlugins* (or on the top solution item, if you're using Visual C# Express) and select *Add → New Project.* In the dialog window, select *Visual C# → CrypTool 2.0 Plugin* as project template and enter a unique name for your new plugin project (such as *Caesar* in our case). The **next step is crucial** to ensure that your plugin will compile and run correctly: select the subdirectory `CrypPluginsExperimental\` as the location of your project (Figure 2.1).
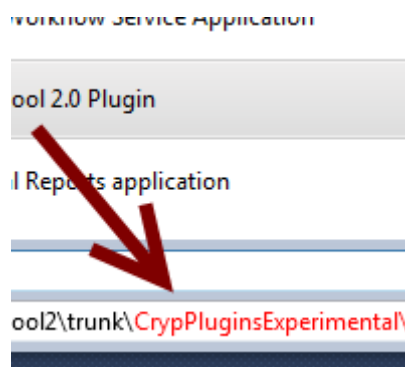

Figure 2.1: Creating a new CrypTool 2 plugin project.

As the project basics are already correctly set up in the template, you should now be able to compile the new plugin. First of all, rename the two files in the solution explorer to something more meaningful, for example `Caesar.cs` and `CaesarSettings.cs`.
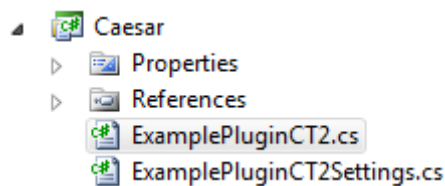

Figure 2.2: Items of a new plugin project.

2.2 Adapt the plugin skeleton

If you look into the two C# source files (Figure 2.2), you will notice a lot of comments marked with the keyword `HOWTO`. These are hints as to what you should change in order to adapt the plugin skeleton to

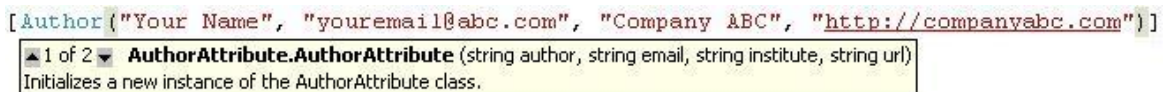your custom implementation. Most `HOWTO` hints are self-explanatory and thus we won't discuss them all.

2.3 Defining the attributes of the Caesar class

The next thing we will do in our example *Caesar.cs* is define the attributes of our class. These attributes are used to provide necessary information for the CrypTool 2 environment. If they are not properly defined, your plugin won't show up in the application user interface, even if everything else is implemented correctly.

Attributes are used for declarative programming and provide metadata that can be added to the existing .NET metadata. CrypTool 2 provides a set of custom attributes that are used to mark the different parts of your plugin. These attributes should be defined right before the class declaration.

2.3.1 The *[Author]* attribute

The *[Author]* attribute is optional, meaning that we are not required to define it. The attribute can be used to provide additional information about the plugin developer (or developers, as the case may be). This information will appear in the TaskPane. We will define the attribute to demonstrate how it should look in case you want to use it in your plugin.

Figure 2.3: The defintion for the *[Author]* attribute.

All of these elements are optional and may be null:

- *Author* — the name of the plugin developer.

- *Email* — the email address of the plugin developer, should he or she wish to be available for contact.

- *Institute* — the organization, company or university with which the developer is affiliated.

- *URL* — the website of the developer or of his or her institution.

2.3.2 The *[PluginInfo]* attribute

The *[PluginInfo]* attribute provides necessary information about the plugin, and is therefore mandatory. The information defined in this attribute appears in the caption and tool tip window. The attribute is defined as follows:

Figure 2.4: Multilingual defintion of *[PluginInfo]* attribute.

This attribute has the following parameters:

- *Resource File* — the namespace and class name of an associated resource file *.resx.* This element is optional and used only if the plugin provides multilingual strings (see also Section 3).

- *Caption* — the name of the plugin, or, if using a resource file, the name of the field in the file with the caption data. This element is mandatory.

- *ToolTip* — a description of the plugin, or, if using a resource file, the name of the field in the resource file with the toolTip data. This element may be null.

- *DescriptionURL* — the local path of the user documentation file (XML file, see Section 4). This element may be null.

- *Icons* — an array of strings to define all the paths of the icons used in the plugin (i.e. the plugin icon described in Section 2.4). This element may be null.

For your first plugin, it's recommended to skip the resource file and use English strings for *Caption* and *ToolTip*. If you're ready to add multi-language support to your plugin, take a look at Section 3.

The *DescriptionURL* element defines the location path of the user documentation file (custom XML format), e.g. `"assemblyname/path/filename.xml"`. Take a look at Section 4 to see how to create a documentation file.

The *Icons* parameter is an array of strings and should be provided in format:
`new[] { "assemblyname/path/filename.png" }`
Don't forget to add all files to your project in Visual Studio. See Section 2.4 how to do this.

2.3.3 Set algorithm category

In the CrypTool 2 user interface plugins are grouped by their algorithm category, for example hash function, symmetric cipher, and so on. To set the category, you must specifiy the attribute *[ComponentCategory]*. Multiple categories are allowed.

```
1    [ComponentCategory(ComponentCategory.CiphersClassic)]
2    public class ExamplePluginCT2 : ICrypComponent
3    {
```

2.3.4 Adding parameters to the CaesarSettings class

If your component provides user-configurable parameters, you can set them up in the settings class derived from *ISetting*. This comprises text input fields, number fields, combo boxes, radio buttons and so on. Take a look at *CaesarSettings.cs* to see some examples for user-configurable parameters.

If you don't want to provide any parameters, delete the class and return null for the *Settings* property:

```
1    public ISettings Settings
2    {
3      get { return null; }
4    }
```

2.4 Add an icon file

Your component is represented in the CrypTool 2 user interface by an icon. You should add a custom icon file, but if you don't, the template uses a generic default one.

The proper image size is 40x40 pixels, but since the image will be rescaled if necessary, any size is technically acceptable. Once you have saved your icon, you should add it directly to the project or to a subdirectory with it. Right-click on your plugin project or any subfolder and select *Add → Existing Item*.
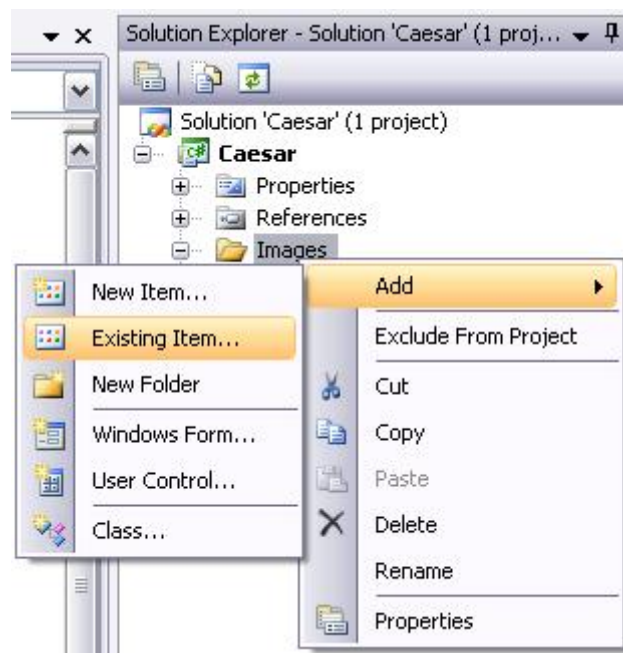
Figure 2.5: Adding an existing item.

Choose *Image Files* as the file type and select your newly-created icon for your plugin.
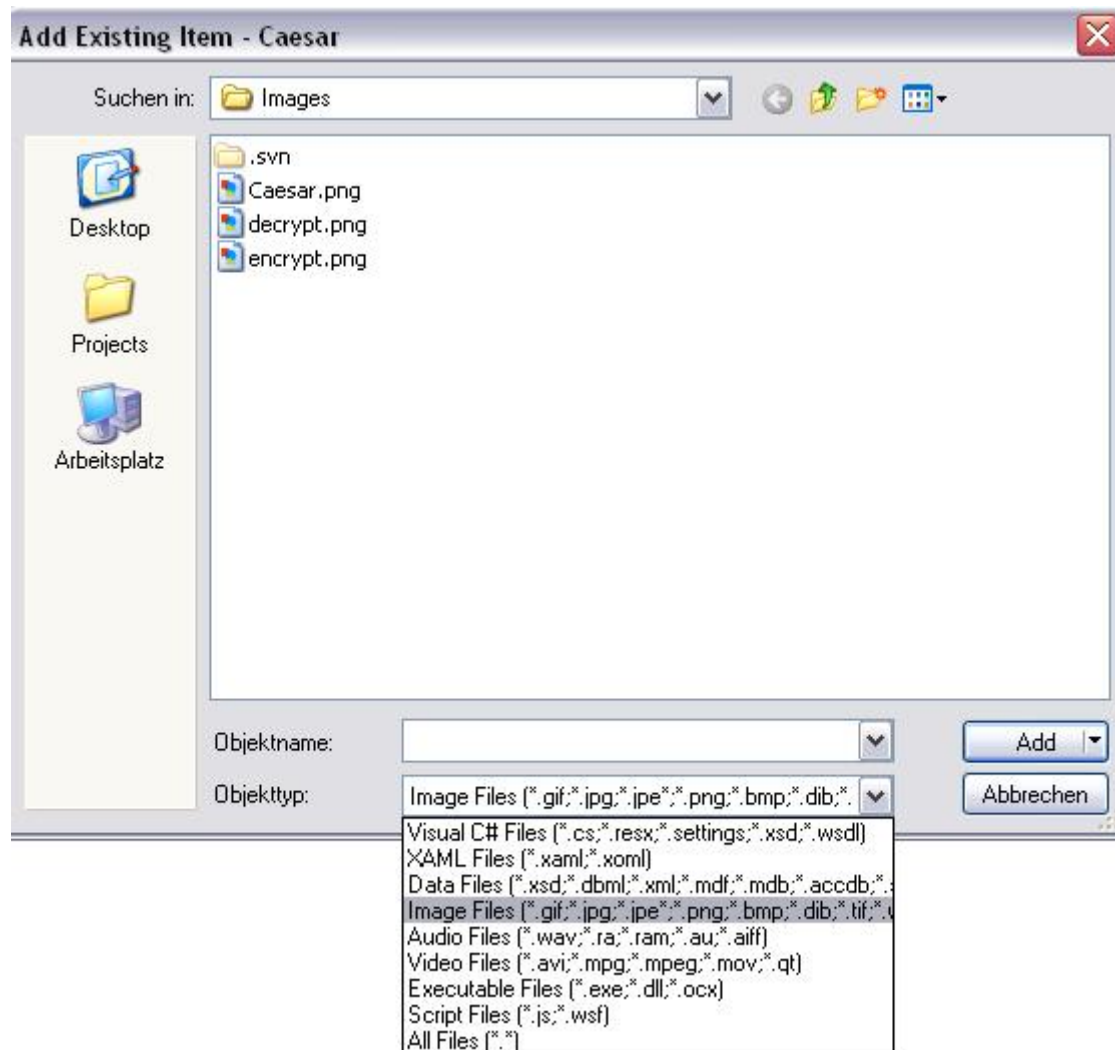

Figure 2.6: Selecting the image file.

It's **necessary** to set the icon file as a *Resource* in the file properties. Right-click on your icon file, click *Properties* and set the *Build Action* to *Resource.*
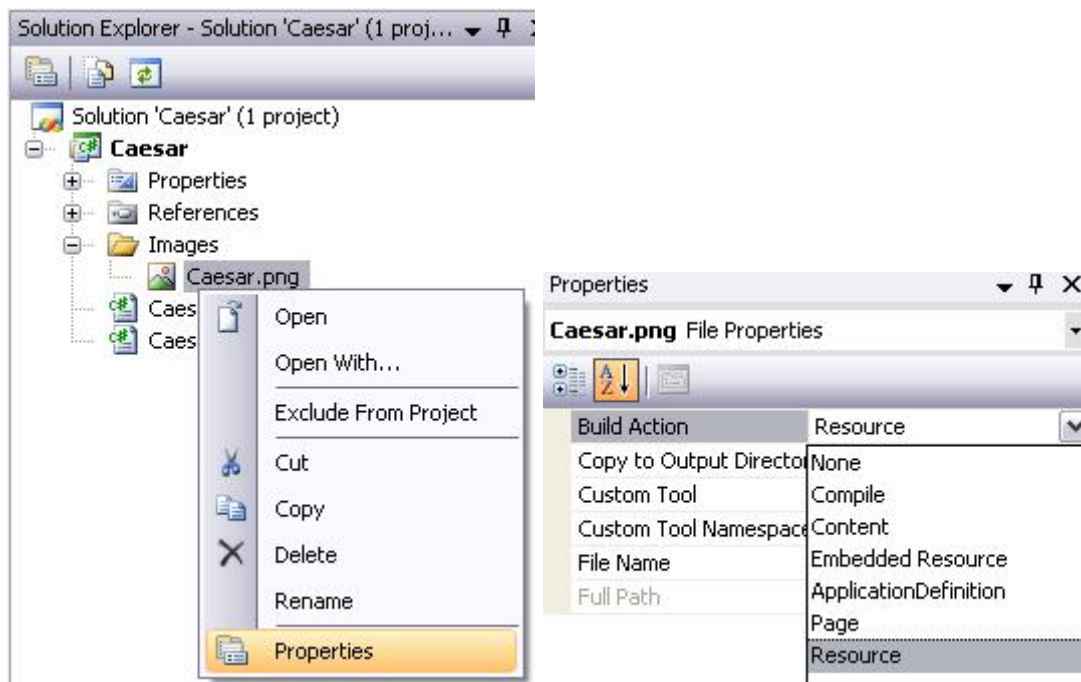

Figure 2.7: Go to *Properties*, select *Build Action* to *Resource.*

2.5 Input and output dockpoints

Next we will define some class properties to be used as data input and output connectors of the component. Each property is defined by a *[PropertyInfo]* attribute:

- *direction* — defines whether this property is an input or output property:
    - `Direction.Input`
    - `Direction.Output`

- *caption* — a short caption of the data property shown in the editor. May be either a text or a key refering to a multilingual resource file (see Section 3).

- *toolTip* — similar as a caption, but more descriptive. May be either a text or a key refering to a multilingual resource file.

- *mandatory* — this flag determines whether an input must be attached by the user to use the plugin. If set to `true`, an input connection will be required or else the plugin will not be executed in the workflow chain. If set to `false`, connecting an input is optional. As this only applies to input properties, if the direction has been set to `Direction.Output`, this flag will be ignored.

Here is an example:

```
1 [PropertyInfo(Direction.InputData, "Text input", "Input a string to be
      processed by the Caesar cipher", false)]
2 public string InputString
3 {
4   get;
5   set;
6 }
```

The output data property (which handles the input data after it has been encrypted or decrypted) may look as follows. CrypTool 2 does not require implementing set methods for output properties, as they will never be called from outside the plugin (it won't hurt, though).

```
1 [PropertyInfo(Direction.OutputData, "Text output", "The string after
      processing with the Caesar cipher", false)]
2 public string OutputString
3 {
4   get;
5   set;
6 }
```

You can basically use any data type. If your component deals with potentially large amounts of binary data, you may want to use the *ICryptoolStream* data type instead of *byte[]*. More information about how to use the *ICryptoolStream* can be found in the CrypTool 2 wiki: https://www.cryptool.org/trac/CrypTool2/wiki/ICryptoolStreamUsage. You will need to include the namespace *Cryptool.PluginBase.IO*. Here's an example how to use *ICryptoolStream* for an output property:

```
1 [PropertyInfo(Direction.OutputData, "CryptoolStream output", "The raw
      CryptoolStream data after processing with the Caesar cipher", false
      )]
2 public ICryptoolStream OutputData
3 {
4   get
5   {
6     if (OutputString != null)
7     {
8       return new CStreamWriter(Encoding.UTF8.GetBytes(OutputString));
9     }
10     return null;
11   }
12 }
```

2.6 Implement your algorithm

Algorithmic processing should be done in the *Execute()* function. Here's a boilerplate example of how it could look like (example simplified for demonstration purposes):

```
1 public void Execute()
2 {
3   if (string.IsNullOrEmpty(InputString))
4     return;
```

```
 5
 6    try
 7    {
 8      ProgressChanged(0, 100);  // set progress bar to 0%
 9      if (settings.Action == CaesarMode.encrypt)
10      {
11        OutputString = Encrypt(InputString);
12      }
13      else
14      {
15        OutputString = Decrypt(InputString);
16      }
17      ProgressChanged(100, 100); // set progress bar to 100%
18
19      OnPropertyChanged("OutputString"); // push output to editor
20    }
21    catch(Exception ex)
22    {
23      // log error
24      GuiLogMessage("Failure: " + ex.Message, NotificationLevel.Error);
25    }
26 }
```

You **must** announce all changes to output properties by calling *OnPropertyChanged* with the correct property name. This step is crucial to correctly pass output data to other components.

You should set the progress of an execution by calling *ProgressChanged*. You may use interim progress updates for long-running computations in a loop.

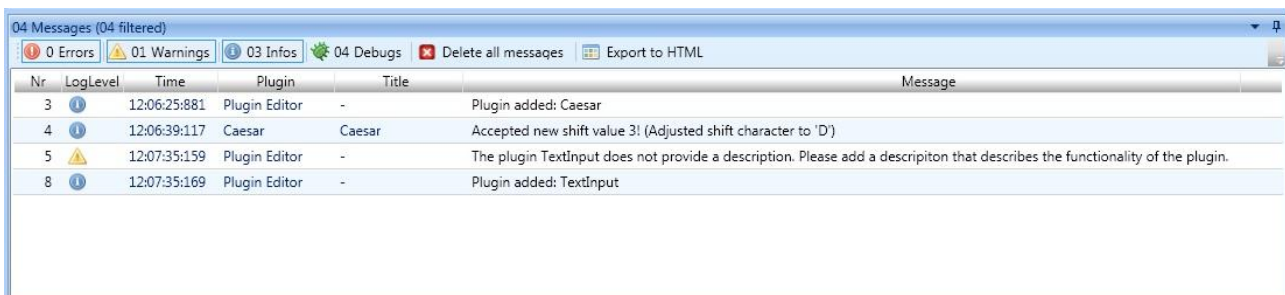You may use *GuiLogMessage* to show errors, warnings or informational messages to the user.

Figure 2.8: An example status bar.

2.7 Create an editor template

Each plugin should have an associated workflow file to show the algorithm in action in CrypTool 2. These workflow files are saved as *Compressed Workspace Manager* files with file extension *.cwm*. You can view the example files from other plugins by opening any of the files in the `Templates\` folder with CrypTool 2. Below is an example workflow (image shows old editor, but the new one looks similar). You should provide such a template .cwm file to demonstrate a typical use case of your plugin. Please place it into the `Templates\` folder and make sure to commit the file to the SVN repository (see Section 1.2.3).
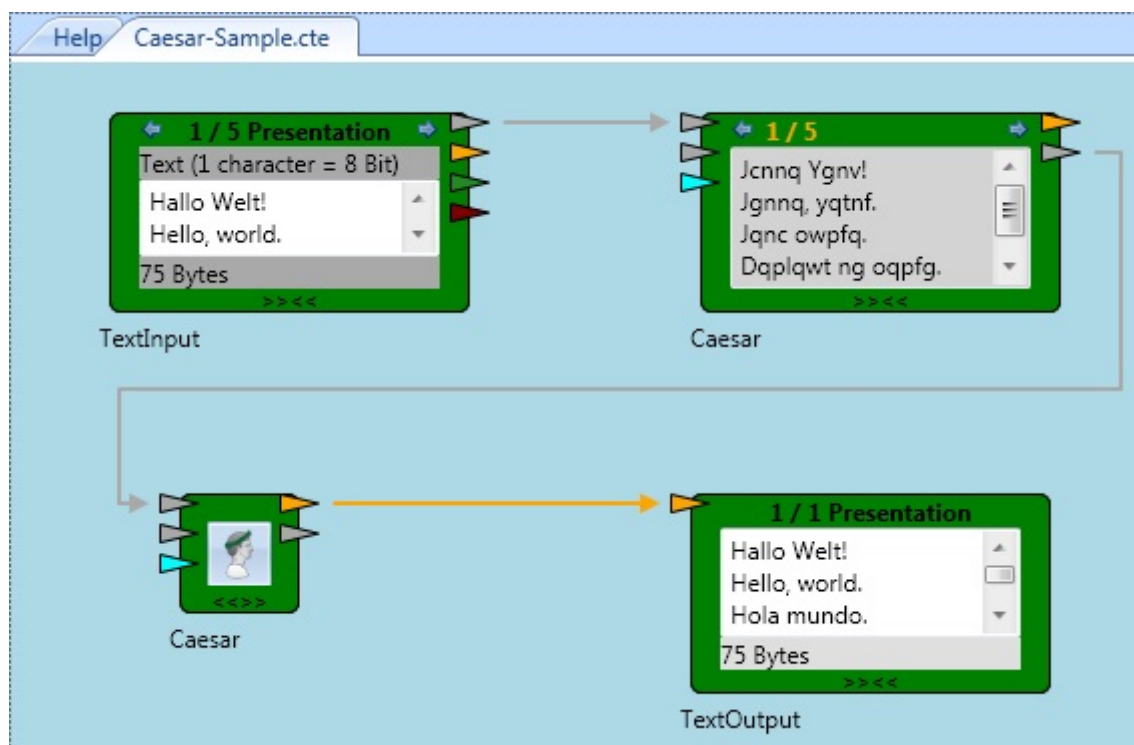
Figure 2.9: A sample workflow diagram for the Caesar algorithm.

3 Internationalization

If you'd like to know how to add multi-language support to your component, please see our Wiki:
https://www.cryptool.org/trac/CrypTool2/wiki/Internationalization

4 Documentation

If you'd like to know how to provide a user documentation file (custom XML format), please see our Wiki: https://www.cryptool.org/trac/CrypTool2/wiki/Documentation