# CrypTool 2.0
## Developer Manual

S. Przybylski, A. Wacker, M Wander and F Enkler
{*przybylski*|*wacker*|*wander*|*enkler*}*@cryptool.org*

Version: 0.1
December 7, 2009

## Contents

**Abstract.** This document is intended for plugin developers.

# 1 Developer Guidelines

CrypTool 2.0 uses state-of-the-art technologies like .NET 3.5 and WPF. In order to make your first steps towards developing something in the context of this project a few things need to be considered. In order to not get stuck, please follow the instructions on this page. If you encouter a problem/error, which is not described here, please let us know, so we can add this information to this guide.

In the following we describe all steps necessary in order to compile CrypTool 2.0 on your own. This is always the first thing you need to do before you go on developing own plugins and extensions. The basic steps are

- Getting all pre-requisites and installing them
- Accessing and downloading the source code with SVN
- Compiling the source-code for the first time

## 1.1 Pre-requisites

Since CrypTool 2.0 is based on Microsoft .NET 3.5, you first need a Microsoft Windows environment. (Right now no plans exist for porting this project to mono and therefore other platforms.) We successfully tested with **Windows XP** and **Windows Vista**.

Since you're reading the developer guidlines, you probably want to develop something. Hence you need a developer environment. In order to compile our sources you need **Microsoft Visual Studio 2008 Professional**. Please make sure you always install the latest service packs for Visual Studio too. Unfortunately it does not work (smoothly) with the freely available Visual Studio Express (C#) versions. This is due to the fact, that CrypWin uses a commercial component, and is therefore distributed only as binary. The C# Express version unfortunately cannot handle a binary as a start project, hence debugging becomes cumbersome.

Usually the installation of Visual Studio also installs the .NET framework. In order to run/compile our source code you need (at the time of this writing) at least **Microsoft .NET 3.5 with Service Pack 1 (SP1)**. You can get this freely from Microsofts webpage.

After the last step, your development environment should be ready for our source-code. Hence, now you need a way of accessing and downloading the entire sources. In the CrypTool 2.0 project we use SVN (subversion control) for version control, hence you need an **SVN client** of you're choice, e.g. **TortoiseSVN** or the **svn commandline from cygwin**. If you never worked with SVN before, we suggest to download and install TortoiseSVN, since it offers a nice windows explorer integration of SVN and any windows user should feel right away at home.

## 1.2 Accessing Subversion Control (SVN)

This section describes how to access our SVN and the basic settings you need.

### SVN URL

Our code repository is accessable under the following url:
`https://www.cryptool.org/svn/CrypTool2/`

If you are a guest and just want to download our source code you can use ”anonymous” as the username and an empty password. If you are a registered developer, just use your provided username and password (which is the same as for this wiki).

### Accessing the SVN with TortoiseSVN

As already mentioned, in order to use the SVN repository one of the best options is TortoiseSVN. Please install TortoiseSVN (unfortunately it will ask you to reboot, which you need to do) and then create somewhere on your computer a directory (for instance ”Cryptool2”) for storing the local working files. Right click on this directory and select ”SVN Checkout” from the context menu. In the new appearing window you must enter the URL of the repository as given above. The ”Checkout directory” should be filled in correctly. After that, just hit ok, accept the certificate (if necessary) and enter your user credentials or ”anonymous” for guests. Also mark the checkbox for saving your crendentials otherwise you will be asked about them for every single file. Then hit ok, and now the whole CrypTool2 repository should be checked out into the given directory.

Later on, if you just want to update (if there where changes in the repository) you can do this with right click on any directory within the working files and choose ”SVN Update” from the context menu. If you changed a file you should choose ”SVN Commit” from the context menu in order to upload your changes. Please always provide *meaningfull descriptions* of your updates.

A TortoiseSVN Tutorial can be found here.

### Ignore Patterns

In order to checkin only clean code, please use the following **ignore patterns**: *”obj bin debug release *.pdb *.suo *.exe *.dll”*

This basically means that you should never check-in compiled and user-generated files. As an example please do not check-in the entire bin/ and obj/ directories which Visual Studio generates. If you want to submit a component (binary file) despite the ignore patterns you can still add *.dll files by using the context menu and add that file explicitely – but please be absolutely sure, that you know what you are doing.

## 1.3 Compiling the sources

At this point you should have checked out the entire CrypTool repository. Then compiling is pretty easy, you just go to the directory *trunk/* and open the **CrypTool 2.0.sln** Visual Studio solution. Now Visual Studio should open with all working plugins and all components nicely arranged. In case you started Visual Studio now for the very first time, you must choose a certain settings – just select either ”most common” or ”C#” – you can change this at any time later. In the right hand you get the project explorer, where you see all the subprojects included in the solution. You have to look for the project **CrypWin.exe** there. When you found it, you need to right-click it and select **”Set as startup-project”** from the context menu. After you have done this, just go to the menu *Build* and select *Build solution* (clearly you can also use the hotkeys if you memorized them). Then go to *Debug* and click *Start debugging* – now CrypTool 2.0 should start for the first time with your own compiled code – clearly you did not change yet anything, however, you have now an own build of all components (with the exception of CrypWin and AnotherEditor, since they are available only as binary). In case it does not compile or start, please consult our F.A.Q. and let us know if you found a bug.

As a core-developer, hence somebody who can also compile CryWin and AnotherEditor, you should use the **CrypTool 2.0.sln** solution from the trunk/CoreDeveloper/ directory (this directory is **not** visible to you if you are not a core developer). As a core developer you should know, that when compiling you **change** the CryWin.exe which is visible to everybody else. Hence, when doing a checkin, please make sure you *really* want to checkin a new binary.

# 2 Plugin Implementation

In this chapter we provide step-by-step instructions for implementing your own CrypTool 2.0 plugin. The given instructions refer to the usage of MS Visual Studio 2008 Professional, hence before starting you should have installed your copy of MS Visual Studio 2008.

## 2.1 New project

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: "Caesar"), 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:

## 2.2 Interface selection

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 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.

Besides the CrypPluginBase you need to add three assembly references to provide the necessary "Windows" namespace for your user control functions called "Presentation" and "QuickWatchPresentation". Select the following .NET components:

- PresentationCore
- PresentationFramework
- WindowsBase

Figure 2.1: Figure 1

Figure 2.2: Figure 2

Figure 2.3: Figure 3

Afterwards your reference tree view should look like this:
[IMAGE]
If your plugin will be based on further libraries, you have to add them in the same way.

## 2.3 Create the classes for the algorithm and for its settings

In the next step we have to create two classes. The first class named "Caesar" has to inherit from IEncryption to provide an ecryption plugin. If you want to develop a Hash plugin your class has to inherit from IHash. The second class named "CaesarSettings" has to inherit from ISettings.

### 2.3.1 Create the class for the algorithm (Caesar)

Visual Studio automatically creates a class which has the name "Class1.cs". There are two ways to change the name to "Caesar.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:
[IMAGE]
Now make a right click on the project item "Caesar" and select "Add-¿Class...":
[IMAGE]
Now give your class a unique name. We call the class as mentioned above "Caesar.cs" and make it public to be available to other classes.
[IMAGE]

### 2.3.2 Create the class for the settings (MD5Settings)

Add a second public class for ISettings in the same way. We call the class "CaesarSettings". 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.
[IMAGE]

### 2.3.3 Add namespace for the class MD5 and the place from where to inherit

Now open the "Caesar.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 Caesar algorithm which means we need to include the following namespaces:

-"Cryptool.PluginBase" to provide "ISettings" for the CaesarSettings class

-"Cryptool.PluginBase.Cryptography" to provide "IEncryption" for the Caesar class

-"Cryptool.PluginBase.Miscellaneous" to use the entire CrypTool event handler

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

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

[IMAGE]

Next let your class "Caesar" inherit from IHash by inserting of the following statement:

[IMAGE]

### 2.3.4 Add the interface functions for the class Caesar

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

[IMAGE]

Choose the item "Implement interface 'IEncryption'". 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:

[IMAGE]

### 2.3.5 Add namespace and interfaces for the class CaesarSettings

Let's now take a look at the second class "CaesarSettings" by double clicking at the "CaesarSettings.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 Caesar class. Visual Studio will here also automatically place code from the CrypTool interface if available.

[IMAGE]

### 2.3.6 Add controls for the class CaesarSettings (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.

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

Before we go back to the code of the Caesar class, we have to add an icon image to our project, which will be shown in the CrypTool ribbon bar or/and navigation pane. As there is no default, using an icon image is mandatory. Note: This will be changed in future. A default icon will be used if no icon image has been provided. For testing purposes you may create a simple black and white PNG image with MS Paint or Paint.NET. As image size you can use 40x40 pixels for example, but as the image will be scaled when required, any size should do it. Place the image file in your project directory or in a subdirectory. Then make a right click on the project item "Caesar" within the Solution Explorer, and select "Add–>Existing Item...":

[IMAGE]

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

[IMAGE]

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":

[IMAGE]

In the "Properties" panel you have to set the "Build Action" to "Resource" (not embedded resource):

[IMAGE]

## 2.5 Set the attributes for the class Caesar

Now let's go back to the code of the Caesar class ("Caesar.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 optional, 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.

[IMAGE]

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 optional. The developer decides what he wants to publish. Unused elements shall be set to null or a zero-length string (""). Our author attribute should look now as you can see below:

[IMAGE]

*[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:

[IMAGE]

This attribute expects the following elements:

o 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. This element is mandatory.

o caption = from type string, the name of the plugin (e.g. to provide the button content). This element is mandatory.

o toolTip = from type string, description of the plugin (e.g. to provide the button tool tip). This element is optional.

o descriptionUrl = from type string, define where to find the whole description files (e.g. XAML files). This element is optional. o 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). This element is mandatory.

Unused optional elements shall be set to null or a zero-length string ("").

Note 1: It is possible to use the plugin without setting a caption though it is not recommended. This will be changed in future and the plugin will fail to load without a caption.

Note 2: Currently a zero-length toolTip string appears as empty box. This will be changed in future.

Note 3: Tooltip and description currently do not support internationalization and localization. This will be changed in future.

In our example the first parameter called "startable" has to be set to "false", because our hash algorithm is neither an input nor generator plugin.

[IMAGE]

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

[IMAGE]

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:

[IMAGE]

Accordingly the attribute parameter has to be set to:

[IMAGE]

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

[IMAGE]

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:

[IMAGE]

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

## 2.6 Set the private variables for the settings in the class Caesar

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

[IMAGE]

Please notice the sinuous line at the type "CryptoolStream" of the variable inputData and the list listCryptoolStreamsOut. "CryptoolStream" is a data type for input and output between plugins and is able to handle large data amounts. To use the CrypTool own stream type, include the namespace "Cryptool.PluginBase.IO" with a "using" statement as explained in chapter 3.3.

The following private variables are being used in this example:

-CaesarSettings settings: required to implement the IPlugin interface properly

-CryptoolStream inputData: stream to read the input data from
-byte[] outputData: byte array to save the output hash value
-List<CryptoolStream> listCryptoolStreamsOut: list of all streams being created by Caesar plugin, required to perform a clean dispose

## 2.7 Define the code of the class Caesar 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:
[IMAGE]
Secondly, we have to implement the property "Settings" defined in the interface:
[IMAGE]
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.
The attribute is named "PropertyInfo" and consists of the following elements:
-direction = defines whether this property is an input or output property, i.e. whether it reads input data or writes output data
o Direction.Input
o Direction.Output
-caption = caption of the property (e.g. shown at the input on the dropped icon in the editor), see below:
[IMAGE]
-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 flag defines whether an input is required to be connected by the user. 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 in the workflow chain. If set to false, connecting the input is optional. This only applies to input properties. If using Direction.Output, this flag is ignored.
-hasDefaultValue = if this flag is set to true, CrypTool treats 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:
o DisplayLevel.Beginner
o DisplayLevel.Experienced
o DisplayLevel.Expert
o DisplayLevel.Professional
-QuickWatchFormat = defines how the content of the property will be shown in the quick watch. CrypTool accepts the following quick watch formats:
o QuickWatchFormat.Base64
o QuickWatchFormat.Hex
o QuickWatchFormat.None
o QuickWatchFormat.Text
A quick watch in Hex could look like this:
[IMAGE]
-quickWatchConversionMethod = this string points to a conversion method; 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 the name of a conversion method as string. The method header has to look like this: object YourMethodName(string PropertyNameToConvert)

First we define the "InputData" property getter and setter:

[IMAGE]

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

Note 1: It is currently not possible to read directly from the input data stream without creating an intermediate CryptoolStream.

Note 2: The naming may be confusing. The new CryptoolStream is not an output stream, but it is added to the list of output streams to enable a clean dispose afterwards. See chapter 9 below.

The setter sets the new input data and announces the data to the Cryptool 2.0 environment by using the expression "OnPropertyChanged("<Property name>"). For input properties this step is necessary to update the quick watch view.

The output data property could look like this:

[IMAGE]

CrypTool does not require implementing output setters, as they will never be called from outside of the plugin. Nevertheless in this example our plugin accesses the property itself, therefore we chose to implement the setter.

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

[IMAGE]

This property's setter is not called and therefore not implemented.

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.

[IMAGE]

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

- o NotificationLevel.Error
- o NotificationLevel.Warning
- o NotificationLevel.Info
- o 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:

[IMAGE]

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

Our whole included namespaces looks now like this:

[IMAGE]

## 2.8 Complete the actual code for the class Caesar

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:

[IMAGE]

It is important to make sure that all changes of output properties will be announced to the CrypTool environment. In this example this happens by calling the setter of OutputData which in turn calls ”OnPropertyChanged” for both output properties ”OutputData” and ”OutputDataStream”. Instead of calling the property’s setter you can as well call ”OnPropertyChanged” directly within the ”Execute()” method.

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:

[IMAGE]

## 2.9 Perform a clean dispose

Be sure you have closed and cleaned all your streams after execution and when CrypTool decides to dispose the plugin instance. Though not required, we run the dispose code before execution as well:

[IMAGE]

## 2.10 Finish implementation

When adding plugin instances to the CrypTool workspace, CrypTool checks whether the plugin runs without any exception. If any IPlugin method throws an exception, CrypTool will show an error and prohibit using the plugin. Therefore we have to remove the ”NotImplementedException” from the methods ”Initialize()”, ”Pause()” and ”Stop()”. In our example it’s sufficient to provide empty implementations.

[IMAGE]

The methods ”Presentation()” and ”QuickWatchPresentation()” can be used if a plugin developer wants to provide an own visualization of the plugin algorithm which will be shown in CrypTool. Take a look at the PRESENT plugin to see how a custom visualization can be realized. For our Caesar example we don’t want to implement a custom visualization, therefore we return ”null”:

[IMAGE]

Your plugin should compile without errors at this point.

## 2.11 Sign the created plugin

## 2.12 Import the plugin to Cryptool and test it

After you have built the plugin, you need to move the newly created plugin DLL to a location, where CrypTool can find it. To do this, there are the following ways:

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.

[IMAGE]

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

[IMAGE]

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

4. Use post-build in your project properties to copy the DLL automatically after building it in Visual Studio. Right-click on your plugin project and select "Properties":

[IMAGE]

Select "Build Events":

[IMAGE]

Enter the following text snippet into "Post-build event command line":

```
cd "$(ProjectDir)"
cd ..\..\CrypWin\$(OutDir)
if not exist "./CrypPlugins" mkdir "./CrypPlugins"
del /F /S /Q /s /q "Caesar*.*"
copy "$(TargetDir)Caesar*.*" "./CrypPlugins"
```

You need to adapt the yellow marked field to your actual project name.

## 2.13 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: anonymous
password: not required

https://www.cryptool.org/svn/CrypTool2/trunk/CrypPlugins/Caesar/

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/encryptionplugin.zip

## 2.14 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 look like:

[IMAGE]