

Concord.EE Plugins "How To"

Concord.EE is supporting extended functionality by writing custom "plugins".

The plugins can be divided into three main categories:

1. Events Plugins:

By implementing a provided interface you get "entry points" on all kind of events during the signing cycle. Application started, document open, signature signed, document saved, document closed etc. on those events you will get the required data in order to implement the business logic required by the customer demands.

2. Special Purpose Plugins:

The special purpose plugins allows you to make some specific tasks in the application differently. For example by implementing the single-method "IPDFParser" interface:

```
XDocument ParsePDF(string pdfPath);
```

The implementation of this method controls how the application will load signature, stamp and image boxes (place holders) into the document. The Concord.EE installation comes with predefined parser plugin (WN.Concord.DipParserPlugin.dll) that implements this interface in order to work with the "DIPs" tags system (see user manual for details). You could implement your own logic for this interface to override the application behavior regarding the PDF parsing.

3. GUI Plugins:

By implementing WPF user control template with a provided interface, you can create custom buttons (and other controls) that will be displayed on the Concord.EE main toolbar. Those control events (clicking a button, choosing an item from a combobox etc.) could be used to custom actions (opening a custom window for input data) as well as to activate Concord.EE actions (open document, save document, close document, minimize or close application etc.).

Requirements:

1. Events Plugins should be written as a class library project (dll) with the name *Plugin.dll under Microsoft .NET framework 4.5.2.
2. GUI plugin should be written as a WPF User Control library (dll) with the name *Plugin.dll under WPF Microsoft .NET framework 4.5.2.
3. All the plugins interfaces are located in "Concord.EE.Extensions.dll" library. This file is in the Concord.EE installation directory and should be referenced in the plugin project.
4. System.ComponentModel.Composition is a framework assembly also needed to be referenced to in the project.
5. Using the NLog logging library (available in nugget) with the same version used by Concord.EE (currently 4.3.9) will allow you to log custom plugin data directly to the main application log for debugging or auditing purposes without any configuration settings required.
6. Written correctly, copying the plugin files into the Plugins directory under the Concord.EE installation directory and restarting the application is all you need in order to activate them.

Example Projects:

Two Visual Studio (2105) example projects will be provided (under the same solution), one for Event Plugin and one for GUI Plugin.

1. The event plugin will use the "SignatureSigned" event, to check whether all the signatures of the document have been signed. If so, the user will be prompt by a message box to save the document. Additionally, a new boolean configuration key "AutoSavePrompt" will be added, allowing to disable the message box prompt and saving the document automatically upon the last signature without prompting.
2. The GUI plugin will be a "Save and Upload" button: *(Under Construction)*