Plugin Development for the AutomationML-Editor

Josef Prinz

inpro

AutomationML PlugFest

Oct. 15th 2015

AutomationML Editor Plugin Development

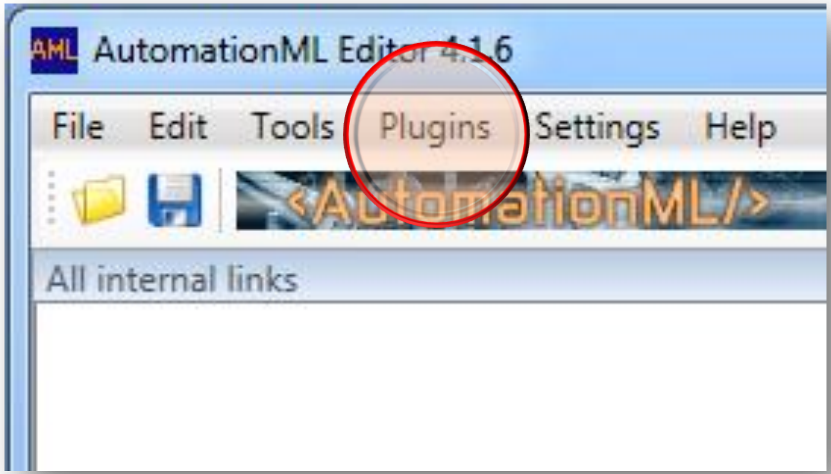
- Motivation
- Plugin Representation
- Plugin Concept
- Implementationresources
- Implementation (getting started)
- Deployment
- Use Cases

Motivation

- Allow others to use the Viewing and Editing Capabilities of the AutomationML-Editor together with some custom Data-Modelling Tools.
- Configurable Extensions of the Viewing and Editing Capabilities of the AutomationML-Editor.
- Synchronized Editing of the same AutomationML-Document.

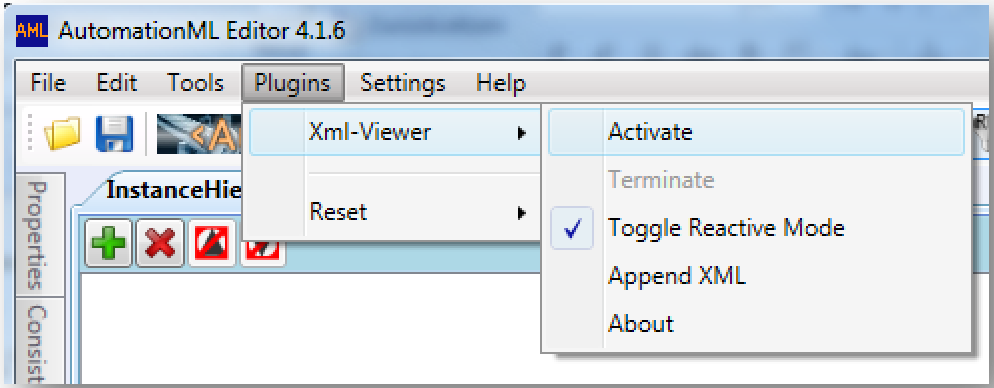
Plugin Representation

- Extra Menu Item “Plugins”
- Available since Version 4.0 of the AutomationML-Editor



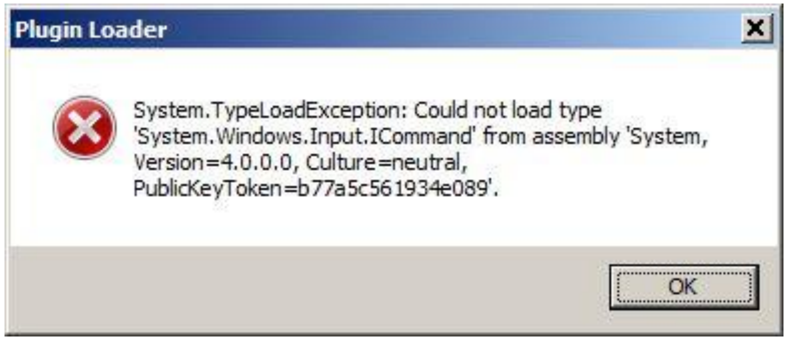
Plugin Representation

- Each Plugin provides a custom Sub-Menu
  - only “Activate” and “Terminate” are mandatory (each plugin has to implement these commands)
- “Reset” Commands are provided by the AutomationML-Editor to force a Termination (when a plugin has no reaction)



Plugin Representation

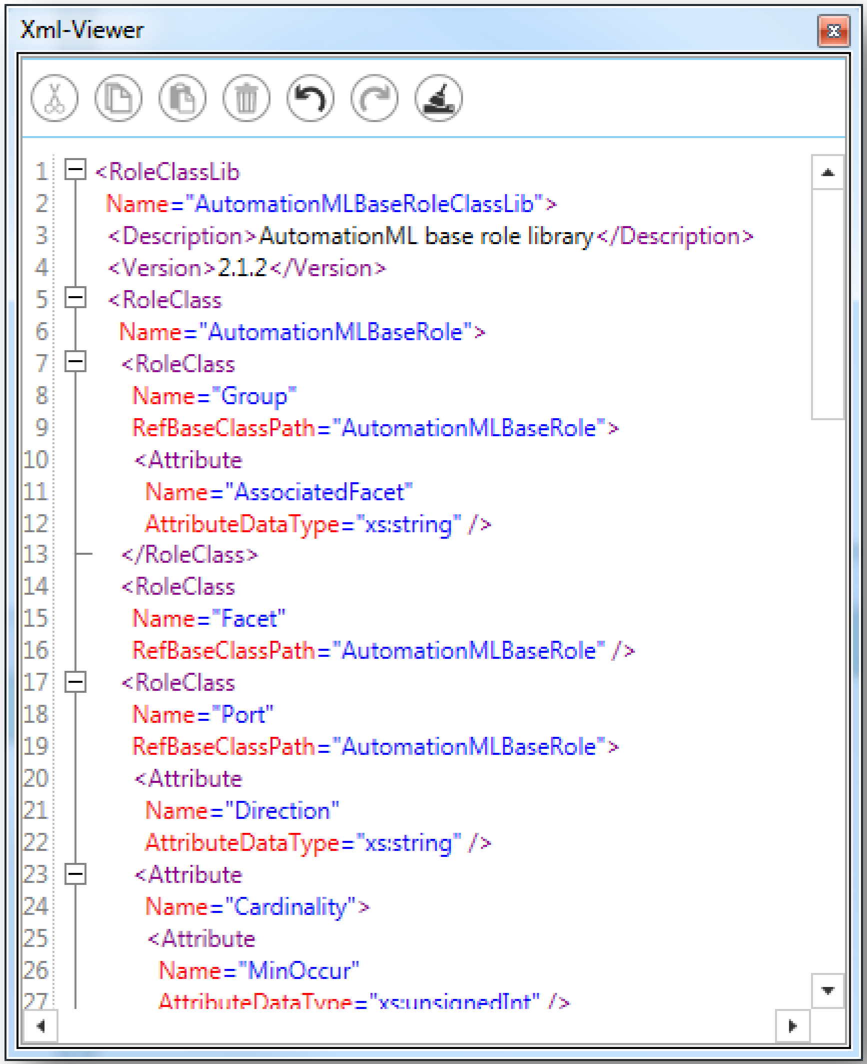
- Possible problems
  - Incompatible libraries will cause a Plugin Loader Exception.
  - The AutomationML-Editor will start, but without plugins.
  - Restart of the AutomationML-Editor due to errors always will be without plugins.



```
System.TypeLoadException: Could not load type
'System.Windows.Input.ICommand' from assembly 'System,
Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089'.
```

Concept

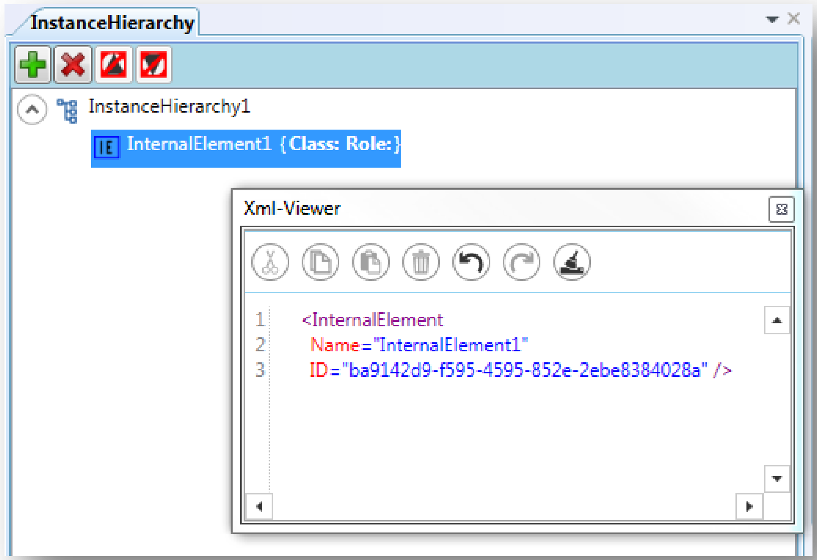
- **ReadOnly plugins**
  - No changes of the AutomationML-Document by the plugin.
  - AutomationML-Editor UI keeps interactive
- **Editing plugins**
  - Changes of the AutomationML-Document by the plugin.
  - AutomationML-Editor UI is freezed while plugin is active.
  - AutomationML-Editor asks user to reload the changed document when plugin terminates.



```
<RoleClassLib
 Name="AutomationMLBaseRoleClassLib">
 <Description>AutomationML base role library</Description>
 <Version>2.1.2</Version>
 <RoleClass
  Name="AutomationMLBaseRole">
  <RoleClass
   Name="Group"
   RefBaseClassPath="AutomationMLBaseRole">
   <Attribute
    Name="AssociatedFacet"
    AttributeDataType="xs:string" />
  </RoleClass>
  <RoleClass
   Name="Facet"
   RefBaseClassPath="AutomationMLBaseRole" />
  <RoleClass
   Name="Port"
   RefBaseClassPath="AutomationMLBaseRole">
   <Attribute
    Name="Direction"
    AttributeDataType="xs:string" />
   <Attribute
    Name="Cardinality">
    <Attribute
     Name="MinOccur"
     AttributeDataType="xs:unsignedInt" />
```

Concept

- **Reactive plugins**
  - Communication via “selection” like in the XMLViewer-Plugin.
- **Passive plugins**
  - AutomationML-Editor “Load State” is communicated to the plugin on activation only.



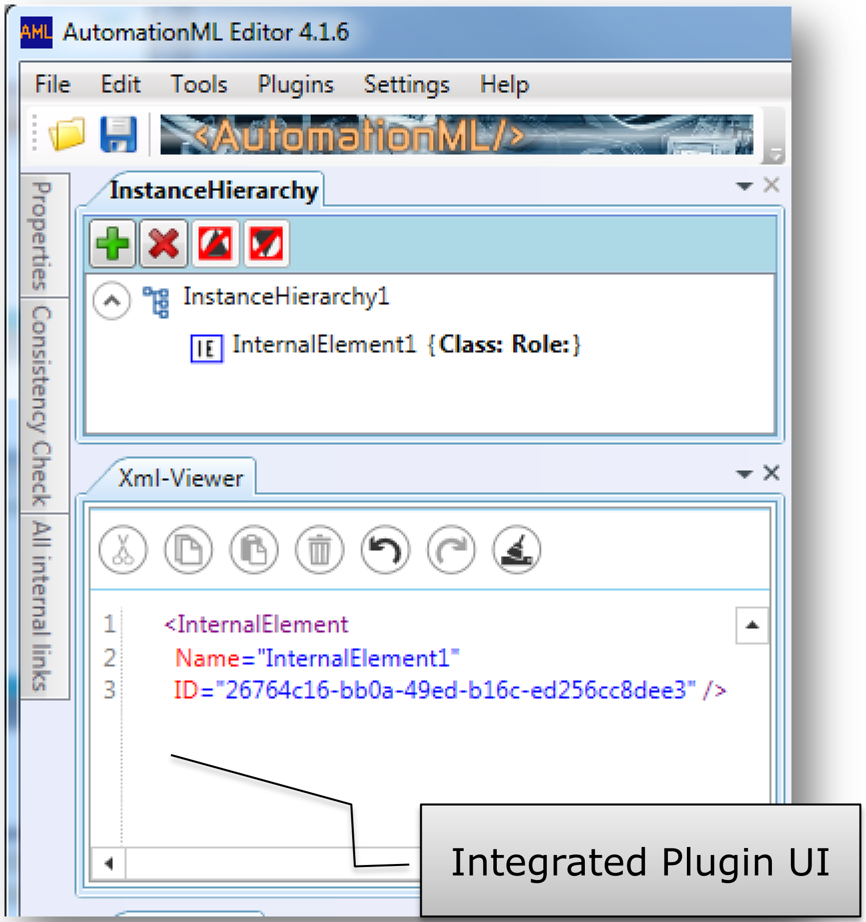
```
<InternalElement
 Name="InternalElement1"
 ID="ba9142d9-f595-4595-852e-2ebe8384028a" />
```

Selected node communicated to reactive plugin.

Concept

- **UI Integrated plugins**
  - Plugin UI (Window) is managed by the layout manager of the AutomationML-Editor.

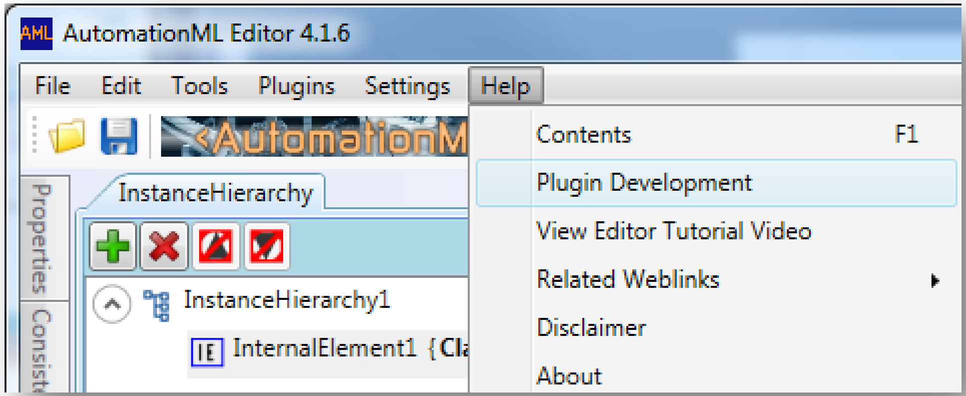
- **Stand alone plugins**
  - Plugin UI (Window) runs on its own Dispatcher Thread.



```
<InternalElement
 Name="InternalElement1"
 ID="26764c16-bb0a-49ed-b16c-ed256cc8dee3" />
```

Implementationresources

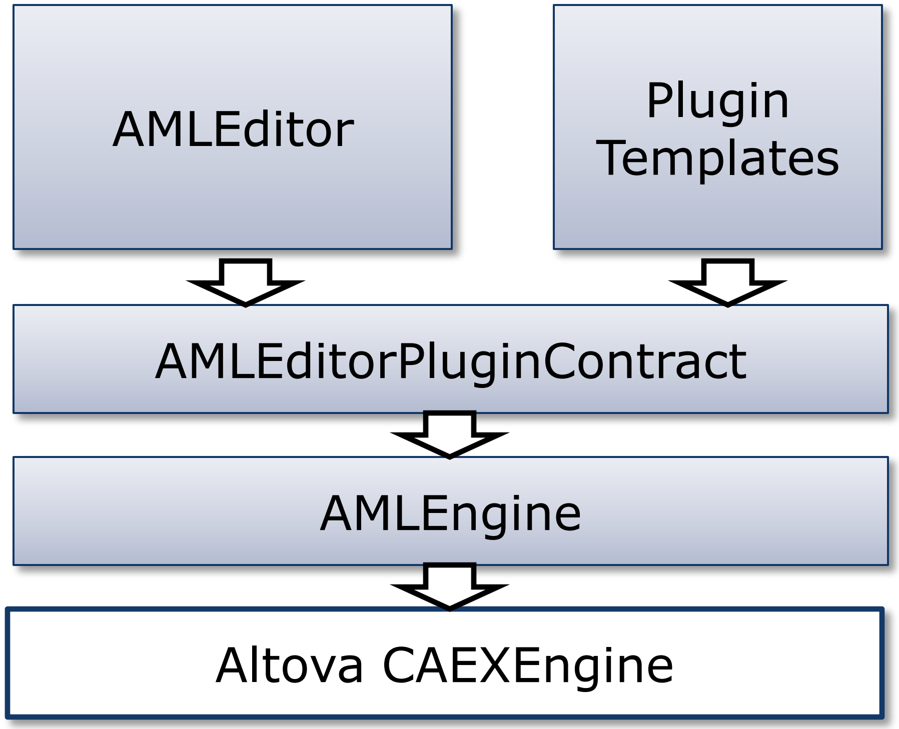
- **Available Documentations**
  - Plugin Development Help
  - Documented examples usable as templates



- **Implementationresources**
  - AMLEditorPluginContract.dll
    - deployment via nuget: https://www.nuget.org/packages/AMLEditorPluginContract
  - Templates (example plugins)
    - “SimpleWPFUserControl” (UI-Integrated readonly plugin)
    - “EditingCAEXApplication” (Standalone editing plugin)
    - deployment via github: https://github.com/AutomationML/AMLEditorPluginContract

Implementationresources

- **Dependencies**
  - All Implementations are based on the AMLEngine (minimal version requirement 3.0.1).
  - The AMLEngine is based on the Altova CAEXEngine, generated for CAEX-Schema Version 2.15.



Implementationresources

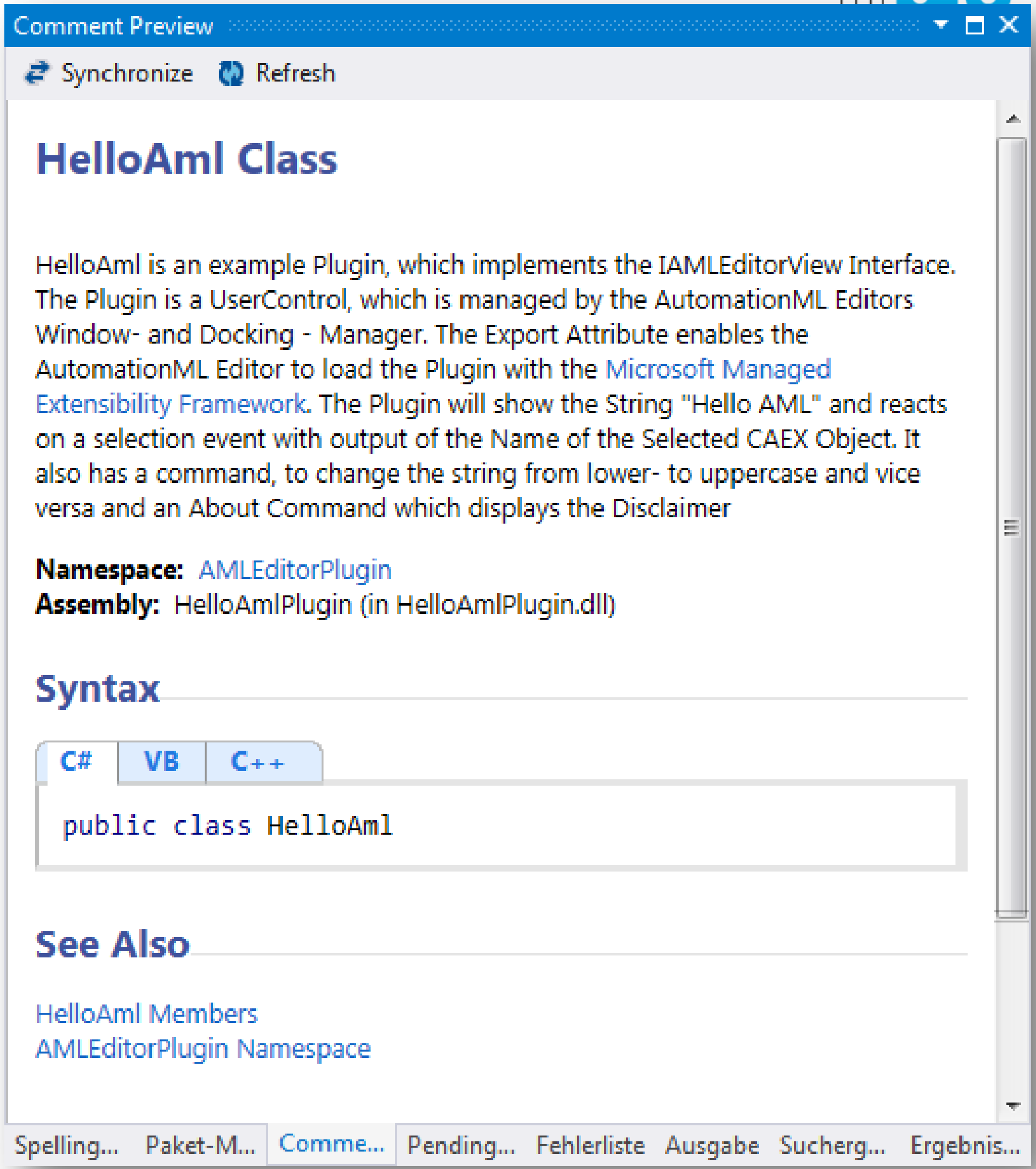
- **AMLEditorPluginContract.dll**
  - Contains interface definitions
    - ***IAMLEditorPlugin***
      Interface, implemented by any plugin
    - ***IAMLEditorView***
      Interface, implemented by UI-Integrated plugins only
  - Contains some Type-Definitions

- **Managed Extensibility Framework (MEF)**
  - AMLEditor Plugin Interfaces are imported/ exported via MEF

Implementation

Getting started

1. Concept Decision
2. Template selection and download
3. Customization
4. Implementation of custom commands
5. Local Test
6. Deployment
7. Integration test



HelloAml Class

HelloAml is an example Plugin, which implements the IAMLEditorView Interface. The Plugin is a UserControl, which is managed by the AutomationML Editors Window- and Docking - Manager. The Export Attribute enables the AutomationML Editor to load the Plugin with the Microsoft Managed Extensibility Framework. The Plugin will show the String "Hello AML" and reacts on a selection event with output of the Name of the Selected CAEX Object. It also has a command, to change the string from lower- to uppercase and vice versa and an About Command which displays the Disclaimer

Namespace: AMLEditorPlugin
Assembly: HelloAmlPlugin (in HelloAmlPlugin.dll)

Syntax

C# VB C++

```
public class HelloAml
```

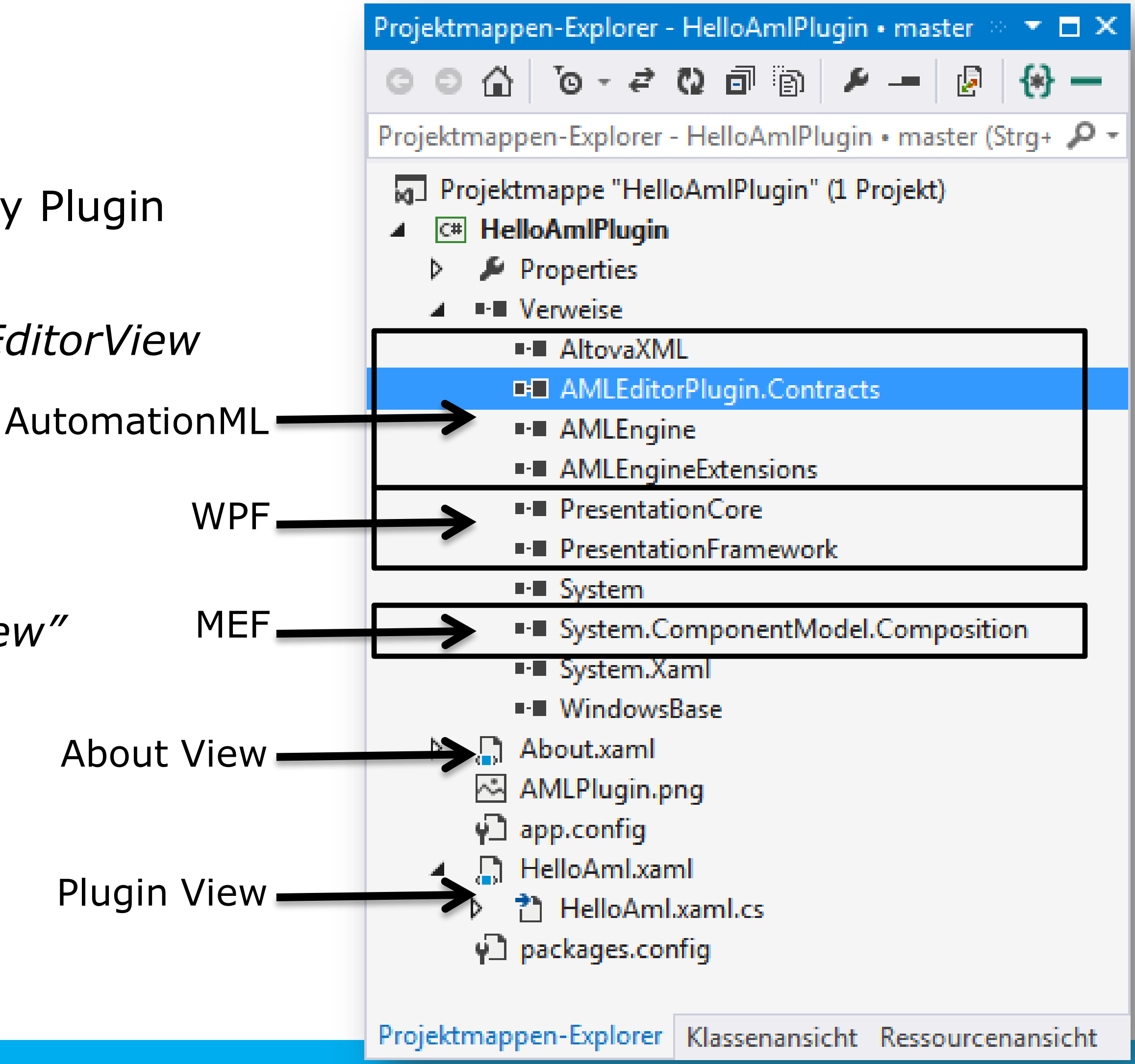
See Also

HelloAml Members
AMLEditorPlugin Namespace

Spelling... Paket-M... Comme... Pending... Fehlerliste Ausgabe Sucherg... Ergebnis...

Implementation - Customization

- **Solution Template**
  - UI Integrated ReadOnly Plugin "HelloAmlPlugin"
  - Implements the *IAMLEditorView* Contract
  - Usage of WPF
  - Usage of MEF
  - Contains an "*About View*"



Implementation - Customization

- **Change ClassName**
  - HelloAml -> …
- **Change build in Commands**
  - CommandName
  - CommandToolTip
- **Change the DisplayName**

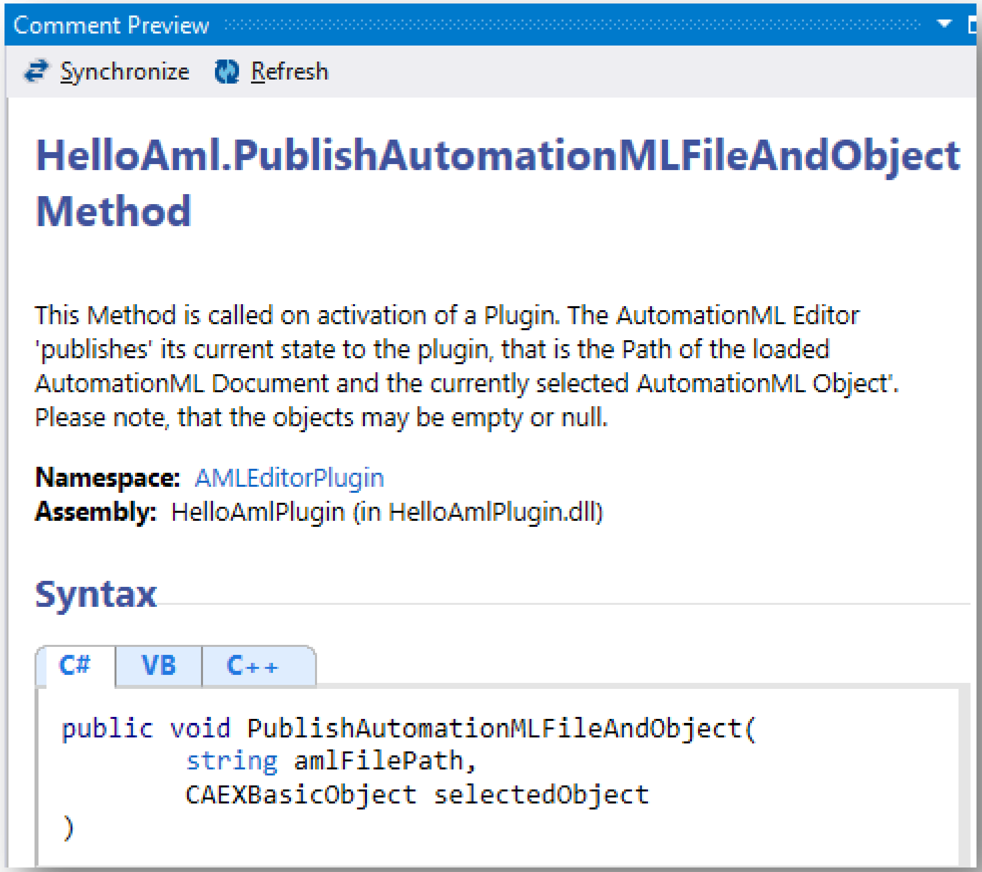
```
[Export(typeof(IAMLEditorView))]
public partial class HelloAml:
     UserControl, IAMLEditorView
```

```
…
ActivatePlugin = new PluginCommand() {
  Command = new RelayCommand<object>
     (this.StartCommandExecute,
      this.StartCommandCanExecute),
  CommandName = "Start",
  CommandToolTip = "Start the Plugin“ };
```

```
public string DisplayName
{
   get { return "Hello AML"; }
}
```

Implementation - Customization

- **Customize the communication methods**
  - AutomationML-Editor communicates its "current state", when the plugin becomes active with the method:
    - **PublishAutomationML-FileAndObject**
  - AutomationML-Editor communicates a "changing state", while the plugin is active with the methods:
    - **ChangeAMLFilePath**
    - **ChangeSelectedObject**
  - A "changing state" is only communicated to a "reactive" plugin.



HelloAml.PublishAutomationMLFileAndObject Method

This Method is called on activation of a Plugin. The AutomationML Editor 'publishes' its current state to the plugin, that is the Path of the loaded AutomationML Document and the currently selected AutomationML Object'. Please note, that the objects may be empty or null.

Namespace: AMLEditorPlugin
Assembly: HelloAmlPlugin (in HelloAmlPlugin.dll)

Syntax

```
public void PublishAutomationMLFileAndObject(
        string amlFilePath,
        CAEXBasicObject selectedObject
)
```

Implementation - Customization

AutomationML-Editor-State

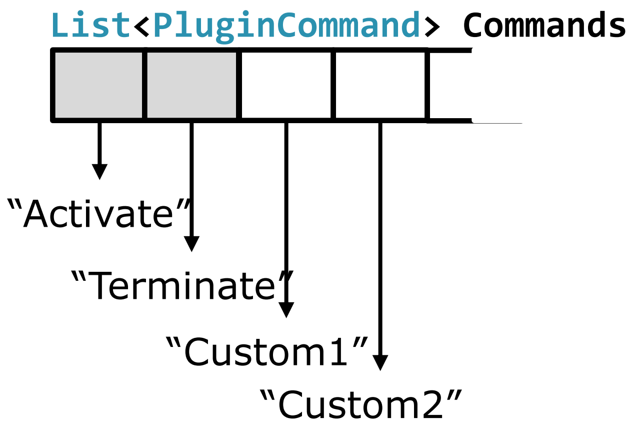
• the loaded AutomationML-Document	(the filepath)
• the selected CAEXObject	(the last selected node in a treeview)

AutomationML-Editor-State communicated to plugins

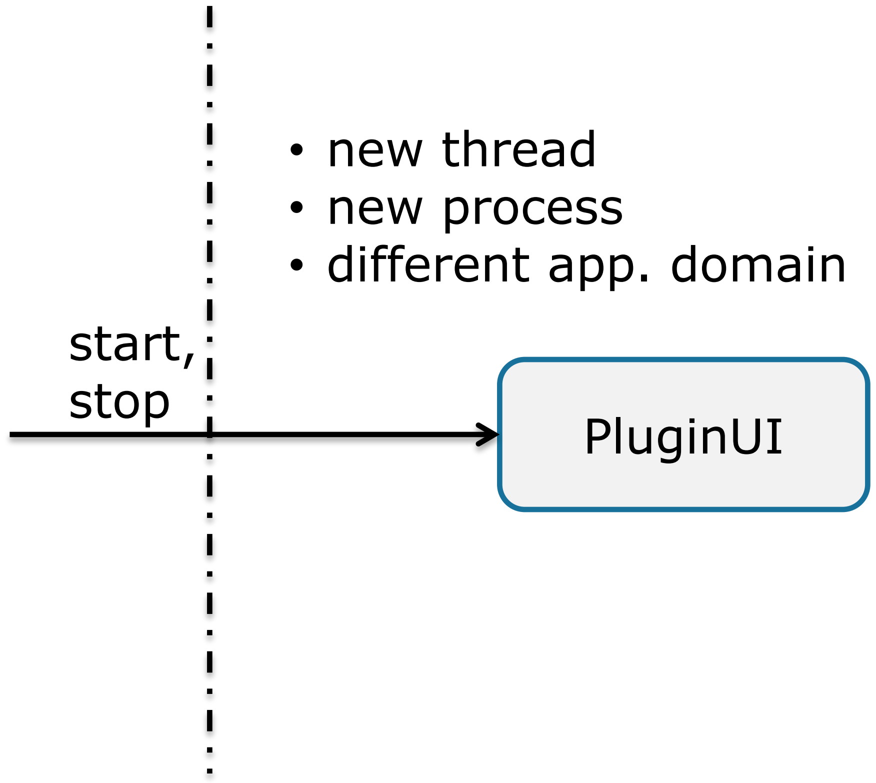
• no AutomationML-Document is loaded	(empty state)
• AutomationML-Document is loaded, no selection	(filepath, empty selection)
• AutomationML-Document is loaded and Node is selected	(filepath, CAEXObject)
• unsaved Changes document	(the user is asked to save the when the plugin is activated)

Implementation – Custom commands

- **Implement plugin commands**
  - Execute method
  - CanExecute method
  - CommandName
  - CommandToolTip
  - Add to list of commands
- **Terminate / Activate commands can be triggered by the AutomationML-Editor**
- **Custom commands are only triggered by a user**



Implementation – Using the Stand Alone template



Implementation – Using the Stand Alone template

- **Communication between Threads**
  - Usage of the .NET Synchronization Context
  - Termination and Activation Events are posted on the Synchronization Context, owned by the AutomationML-Editor
- **Plugin Methods**
  - Execution of methods on the Plugin-UI Dispatcher Thread

<u>Logic:</u>

AutomationML-Editor
    Activate Plugin

Plugin
    A: Get Current Synch. Context
    Start a new Thread

Plugin Thread
    B: Create new Synch. Context
    Create Plugin-UI
    Register a Close Handler for the UI
    Show the UI
    Post "Activated Event" on Sync. "*A*"
    Run the Dispatcher (will use "*B*")

Plugin Deployment

- **Plugin Folder**
  - Plugin libraries should be copied to the “Plugin”-Folder of the AutomationML-Editor executable
  - AML-base-libraries (AMLEngine, AMLPluginContract) need not be copied
- **Plugin Offer**
  - Plugins are offered by the AutomationML organization.
  - For a possible impairment of fitness by plugins, no liability is accepted (AutomationML-Editor disclaimer).
  - User defined plugins may be offered by the AutomationML organization under the following conditions (*):
    - Development by an AutomationML member
    - Source code and disclaimer are provided
    - Usability is tested and approved by the AutomationML Workshop

(*) **needs clarification by the AutomationML Members**

Plugin Use Cases

- **Existing**
  - XML Viewer
- **Ideas for future developments**
  - InternalLink Viewer
  - InternalLink Graphic-Editor
  - ChangeMode Viewer
  - Difference Viewer (compared to a second AutomationML-Document)
  - OCL-Editor (AutomationML <-> OCL Integration)

Thank you for your kind attention!

Please ask questions?

inpro
Innovationsgesellschaft für fortgeschrittene
Produktionssysteme in der Fahrzeugindustrie mbH

Steinplatz 2

D-10623 Berlin

www.inpro.de

Josef Prinz, josef.prinz@inpro.de