SDK for UniversalPlantViewer

Manual

Table of Contents

Introduction

SDK for UniversalPlantViewer Builder allows you to create custom plugins, which will be executed by the **UniversalPlantViewer Builder**. With those plugins you can create new, or read, update and delete existing attributes and graphics.

Setup

Requirements:

- Hardware (minimum):
  - CPU: double-core, 2.5 GHz
  - RAM: 3 GB
- Hardware (recommended):
  - CPU: quad-core, >=3.4 GHz
  - RAM: >=8GB
  - OS, %TEMP%, input and output folders on SSD
  - Full HD resolution
- Supported operating systems:
  - Microsoft Windows 7-10
- Microsoft .NET Framework 4.0

Administrator rights are required on each computer to install CAXperts **SDK for UniversalPlantViewer**. “Run as” is not supported.

Uninstall

To uninstall the software, select **Control panel** from the start menu. Then on large or small icon view, click **Programs and features**. If you are using category view, under **Programs**, click **Uninstall a program**.

Select the program you want to remove, and click Uninstall/Remove. Alternatively, right-click the program and select **Uninstall**.

Setup command line (for administrators)

The following command line options are supported by the installer:

/S:<<optional ini file>>
Allow an install to be run in silent mode. As a result, no screens or dialogs will be shown.

This command line option also has an optional INI file that can be passed containing session variable values. For example:

```
"C:\output\setup.exe" "/S:C:\setupvars.ini"
```

This will cause the session variables in the INI file to be used for the setup. The INI file should be in the format:

```
[SetupValues]

%AppFolder%=C:\Program Files\MyProduct
```

/U:<<XML config file>>
This command line option must be used when calling the uninstall program from the command line. This command also has an optional XML file that can be passed containing session variable values. For example:

```
"C:\Program Files\MyProduct\uninstall.exe" "/U:C:\Program Files\MyProduct\irunin.xml"
```

Licence

A UniversalPlantViewer API licence is required to run this product.

CAXperts software supports two types of licences:

Site licence (local)

Domain based licences are restricted to machines within a 5 miles (8 kilometres) radius at a specific geographic location (building) for which the licence key file was issued for. There is no limitation regarding the number of users or machines.

CAXperts will need your **Current domain name** to issue a licence file (.lic).

The licence file has to be stored on a local or network location accessible by the machine running the CAXperts product; a licence server is not required.

The location of the licence file should be defined on every machine with the CAXperts product installed, which can be done

- Either by setting the environment variable
  `CAXPERTS_LICENSE_FILE=C:\Program Files (x86)\CAXperts\Licensing\`
- Or by setting the registry key
  `HKEY_CURRENT_USER\Software\FLEXlm License Manager\CAXPERTS_LICENSE_FILE=C:\Program Files (x86)\CAXperts\Licensing\`

Multiple licence key file locations should be separated by semicolons (";").

If the licence key file location is not defined, "C:\Program Files (x86)\CAXperts\Licensing\" is used

The licence key file may be renamed (including the extension). The CAXperts product will check the content of all files in the licence folder(s) for valid keys.

Changes to the key path (in the environment variable or registry key) are read during the next start of the CAXperts product.

Floating (server) licence

Floating licences require a FlexNET server (lmadmin or lmgrd) accessible by the machine running the CAXperts product. The licence key file is typically located on the FlexNET server machine. The licence key file defines the maximum number of seats which can be used at the same time.

> *Every running instance of the CAXperts software product will require one seat, no matter if the application is running multiple times on the same or different machine(s).*

New instances can only be started as long as seats are available and the FlexNET server is accessible.

The location of the licence server must be defined on every machine with the CAXperts product installed, which can be done

- Either by setting the environment variable
  `CAXPERTS_LICENSE_FILE=@YourServerName`
- Or by setting the registry key
  `HKEY_CURRENT_USER\Software\FLEXlm License Manager\CAXPERTS_LICENSE_FILE=@YourServerName`

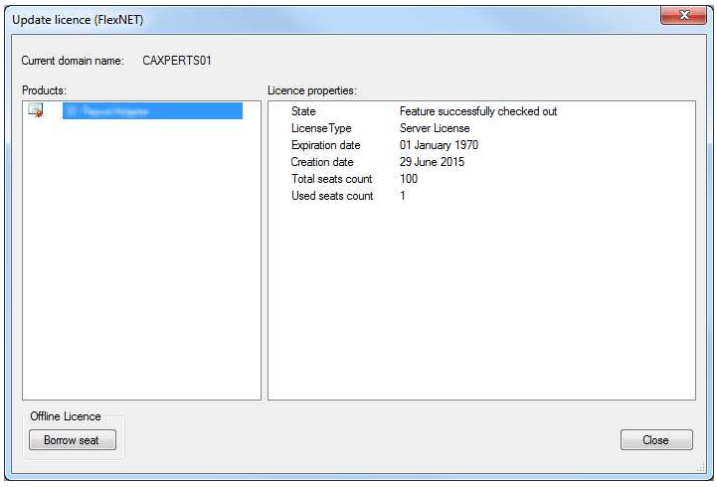
Every computer name must be preceded by the @ symbol. Multiple licence server locations should be separated by semicolons (";").

YourServerName is the Windows computer name of the machine running lmadmin (FlexNET server). If lmadmin is running on a non-default port, the port number should be defined like this: port@YourServerName.

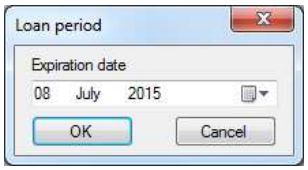
> *Changes to the server path (in the environment variable or registry key) are read during the next start of the CAXperts product.*

Borrowed licences

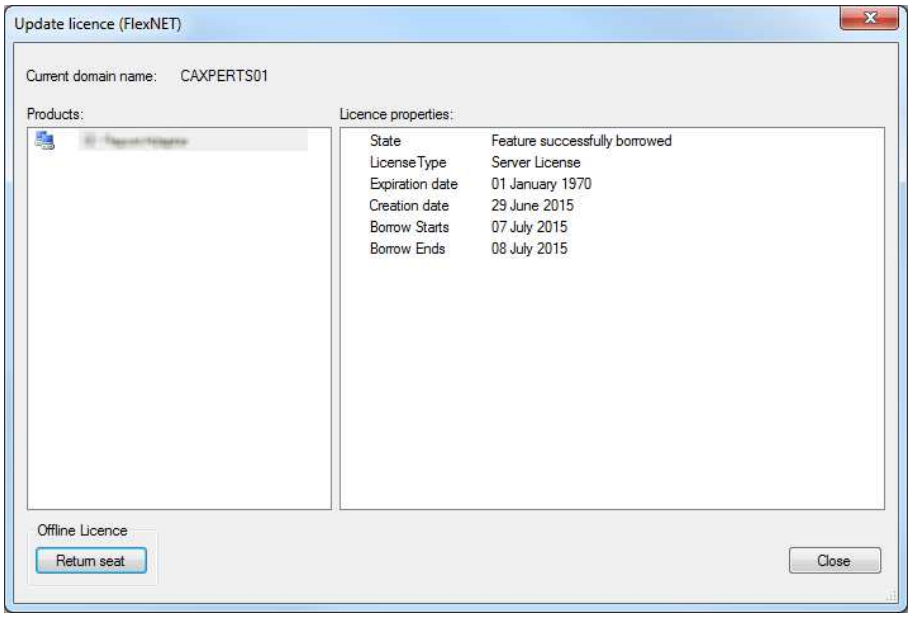
Licence seats can be borrowed for a period of time from the server and saved to the local machine. Once the seat has been borrowed the CAXperts application can be used without any connection to the FlexNET server.



To borrow a licence seat press the **Borrow seat** button, define the loan period (up to 7 days) and press OK. When the borrowed seat expires it gets automatically returned back to the FlexNET server.



The seat can be returned earlier by pressing the **Return borrowed seat** button.



Without a valid licence file the software will run in demo mode.

Working with SDK for UniversalPlantViewer Builder

Creating plugins

In order to develop a custom plugin, you first need to reference the "UniversalPlantViewerBuilderPlugin.dll" file in your plugin project. Your custom plugin class must implement the interface "IPluginVersion1" and its method "BeforeExport". This method is then called by the UniversalPlantViewer Builder. The "PreExportHook" that is passed in as a parameter into the "BeforeExport" method, is the single entry-point which contains various accessors and methods to create or modify attributes and graphics on the fly (see API documentation for details).

Prerequisites for debugging

If you want to debug your custom plugin, you have to adjust both the "Start Action" as well as the "Build Events" of your project:

Start action

Here you have to enter the path to your "UniversalPlantViewer Builder" executable file, i.e. "%Program Files%\CAXperts\UniversalPlantViewer Builder\UniversalPlantViewer Builder.exe"

Build events

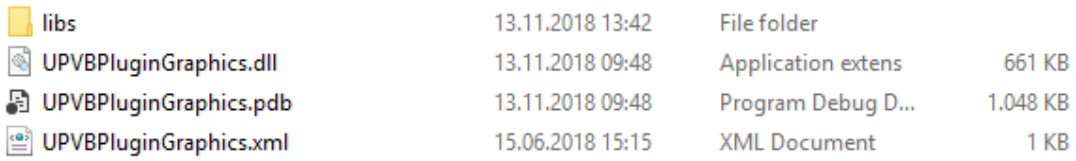
Under Post-Build Events please enter the following:

```
COPY /Y "$(TargetPath)" "%AppData%\CAXperts\UniversalPlantViewer Builder\Plugins"
```

Background: The UniversalPlantViewer Builder will execute every DLL which can be found in the "%AppData%\CAXperts\UniversalPlantViewer Builder\Plugins" directory.

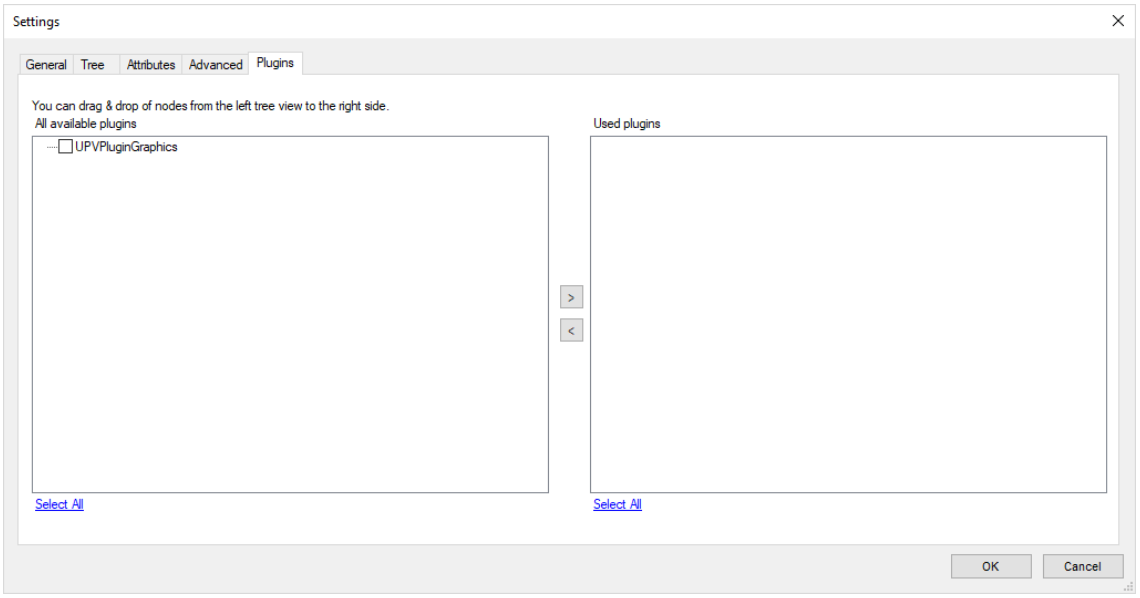
Dependencies

If your plugin requires any third party dependencies or other DLL files, you have to place them inside a libs folder next to your custom plugin DLL files.



Running your plugins

Once you have deployed your custom plugin DLLs to the folder where the SDK will look for plugins "%AppData%\CAXperts\UniversalPlantViewer Builder\Plugins", you can select the plugins which should be executed in the Settings - Plugins tab:



API

The various accessors below can be reached through the "PreExportHook" object that is passed in as a parameter in the "BeforeExport" method. The "BeforeExport" method is the single entry point for a custom plugin. A plugin class must implement the interface "IPluginVersion1" and its method "BeforeExport". This method is then called by the SDK in the UniversalPlantViewer Builder.

BuilderContext

The builder context is used for logging information to the UniversalPlantViewer Builder output file.

void Log(string message)
Prints a message to the UniversalPlantViewer Builder output file. Requires a string message that should be logged to the output file as a parameter and returns void.

void LogError(string message, Exception e)
Prints a message and an exception to the UniversalPlantViewer Builder output file. Requires a string message and an exception object as parameters and returns void.

void CancelExecution()
Throws a new CanceledExecutionException.

void CancelExecution(string message)
Throws a new CanceledExecutionMessage with the message. Requires the string message as parameter.

BuilderElementAccessor
This accessor contains methods for accessing elements and session objects.

IEnumerable<ISession> Sessions { get; }
This property is used to get the available session objects. Returns an IEnumerable of ISession interfaces.

IEnumerable<IElement> IterateElements()
Iterates over all elements and returns an IEnumerable of IElement.

IElement GetElement(Guid uid)
Access a single element by its uid. Requires the uid as parameter and returns either the found element or null if no element was found.

IElement CreateElement(Guid uid, ISession session)
Create a new builder element. Requires a an uid and a session object as parameters and returns the newly created element.

void DeleteElement(Guid uid, ISession session)
Completely deletes an existing element including all its attributes and graphics. Requires the uid and the session object of the element as parameters and returns void.

BuilderAttributeAccessor
Contains various methods to interact with attributes.

IEnumerable<string> IterateAttributes()
Iterate over all attribute keys. Returns an IEnumerable of strings.

IEnumerable<IElement> FindElementsByAttributes(string conditionQuery)
Find all elements by a certain condition. Requires the query condition string as a parameter and returns an IEnumerable of IElement. The query condition is of the following form: “Name=Value”. Several conditions can be combined with an “&” character.

void RenameAttribute(string originalName, string newName)
Change an attribute key for all elements. Requires the old and new attribute key as parameters. Returns void.

void DeleteAttribute(string name)
Completely removes an attribute. Requires the attributes name as parameter and returns void.

IDictionary<string, string> GetAttributes(IElement element, HashSet<string> attributes = null)
Gets all attributes for a specific element. Requires the element as parameter and returns a dictionary of attributes. Optionally, a hash set with the attribute keys to extract could be passed as a second parameter.

void AddAttribute(IElement element, string name, string value)
Adds a new attribute to an existing element. The required parameters are the element, the attribute name and value. Returns void.

void UpdateAttribute(IElement element, string name, string value)
Updates an existing attribute of an element. The required parameters are the element, the attribute name and value. Returns void.

void RenameAttribute(IElement element, string oldName, string newName)
Renames an existing attribute of an element. The required parameters are the element, the old attribute name and the new attribute name. Returns void.

void DeleteAttribute(IElement element, string name)
Removes an existing attribute of an element. Requires the element and the attributes name as parameters. Returns void.

BuilderGraphicAccessor
Contains various methods for getting, creating and changing graphics.

ISessionTransformation GetSessionTransformation(ISession session)
Returns the transformation information for the corresponding session object. Requires the session object as input parameter.

IEnumerable<string> GetAspects()
Access all available aspects. Returns an IEnumerable of strings.

void CreateAspect(string aspectName)
Creates a new aspect. Requires the new aspect name as parameter. Returns void.

void DeleteAspect(string aspectName)
Deletes an existing aspect. Requires the aspect name as input and returns void.

IEnumerable<IGraphicElement> GetGraphics(IElement element)
Get all graphic objects of an element. Returns an IEnumerable of graphic elements. Requires the element as input parameter.

IGraphicElement CreateGraphic(IElement element, string aspectName, IEnumerable<string> primitives)
Create a new graphic. The required parameters are the element, the aspect name and the primitives. Returns the newly created graphic element.

IEnumerable<string> GetPrimitives(IGraphicElement graphicElement)
Access the primitives of a certain graphic element. Returns an enumerable of strings.

void SetPrimitives(IGraphicElement graphicElement, IEnumerable<string> primitives)
Sets the primitives for a certain graphic element. Required parameters are the graphic element and the new primitives. Returns void.

void SetColor(IGraphicElement graphicElement, Color color)
Change the color of a graphic element. Paramters are the graphic element and the new color. Returns void.

BuilderLinkAccessor
Used to attach links to elements.

ILink CreateLink(string url, Color color, string name = null, LinkType type = LinkType.Default)
Create a new link. Use the returned identifier to register elements to the link.

void AddRelation(ILink link, IElement element)
Register an element to the link.

BuilderDocumentAccessor
Contains methods to register and link documents with elements.

IDocument CreatePdf(byte[] fileContent, string name, string treePath)
Register a new pdf file which will be written to the general output location.

void AddRelation(IDocument document, IElement element)
Link an element to a created document.

BuilderIntelliPidElementAccessor
This accessor contains methods for accessing IntelliPid elements and session objects.

IEnumerable<ISession> Sessions { get; }
This property is used to get the available session objects. Returns an IEnumerable of ISession interfaces.

IEnumerable<IIntelliPidElement> IterateElements()
Iterates over all elements and returns an IEnumerable of IIntelliPidElement.

IIntelliPidElement GetElement(Guid uid)
Access a single element by its uid. Requires the uid as parameter and returns either the found element or null if no element was found.

BuilderIntelliPidAttributeAccessor
Contains various methods to interact with IntelliPid attributes.

IEnumerable<string> IterateAttributes()
Iterate over all IntelliPid attribute keys. Returns an IEnumerable of strings.

IEnumerable<IIntelliPidElement> FindElementsByAttributes(string conditionQuery)
Find all IntelliPid elements by a certain condition. Requires the query condition string as a parameter and returns an IEnumerable of IIntelliPidElement. The query condition is of the following form: “Name=Value”. Several conditions can be combined with an “&” character.

void RenameAttribute(string originalName, string newName)
Change an IntelliPid attribute key for all IntelliPid elements. Requires the old and new attribute key as parameters. Returns void.

void DeleteAttribute(string name)
Completely removes an IntelliPid attribute. Requires the attributes name as parameter and returns void.

IDictionary<string, string> GetAttributes(IIntelliPidElement element, HashSet<string> attributes = null)
Gets all IntelliPid attributes for a specific IntelliPid element. Requires the IntelliPid element as parameter and returns a dictionary of attributes. Optionally, a hash set with the attribute keys to extract could be passed as a second parameter.

void AddAttribute(IIntelliPidElement element, string name, string value)
Adds a new attribute to an existing IntelliPid element. The required parameters are the IntelliPid element, the attribute name and value. Returns void.

void UpdateAttribute(IIntelliPidElement element, string name, string value)
Updates an existing attribute of an IntelliPid element. The required parameters are the IntelliPid element, the attribute name and value. Returns void.

void RenameAttribute(IIntelliPidElement element, string oldName, string newName)
Renames an existing attribute of an IntelliPid element. The required parameters are the IntelliPid element, the old attribute name and the new attribute name. Returns void.

void DeleteAttribute(IIntelliPidElement element, string name)
Removes an existing attribute of an IntelliPid element. Requires the element and the attributes name as parameters. Returns void.

Contact

Contact CAXperts' support by email, online, or phone:

CAXperts GmbH
Carl-Zeiss-Ring 4
85737 Ismaning
Germany

https://www.caxperts.com/contact/
Phone: +49 (89) 969772-0
Email: info@caxperts.com

Helpdesk

Available Monday to Friday 08.00 a.m. – 5.00 pm (UTC +1)
Phone: +49 (89) 969772-250
support@caxperts.com