Image credits: www.gira.de

Content

What is a plug-in?

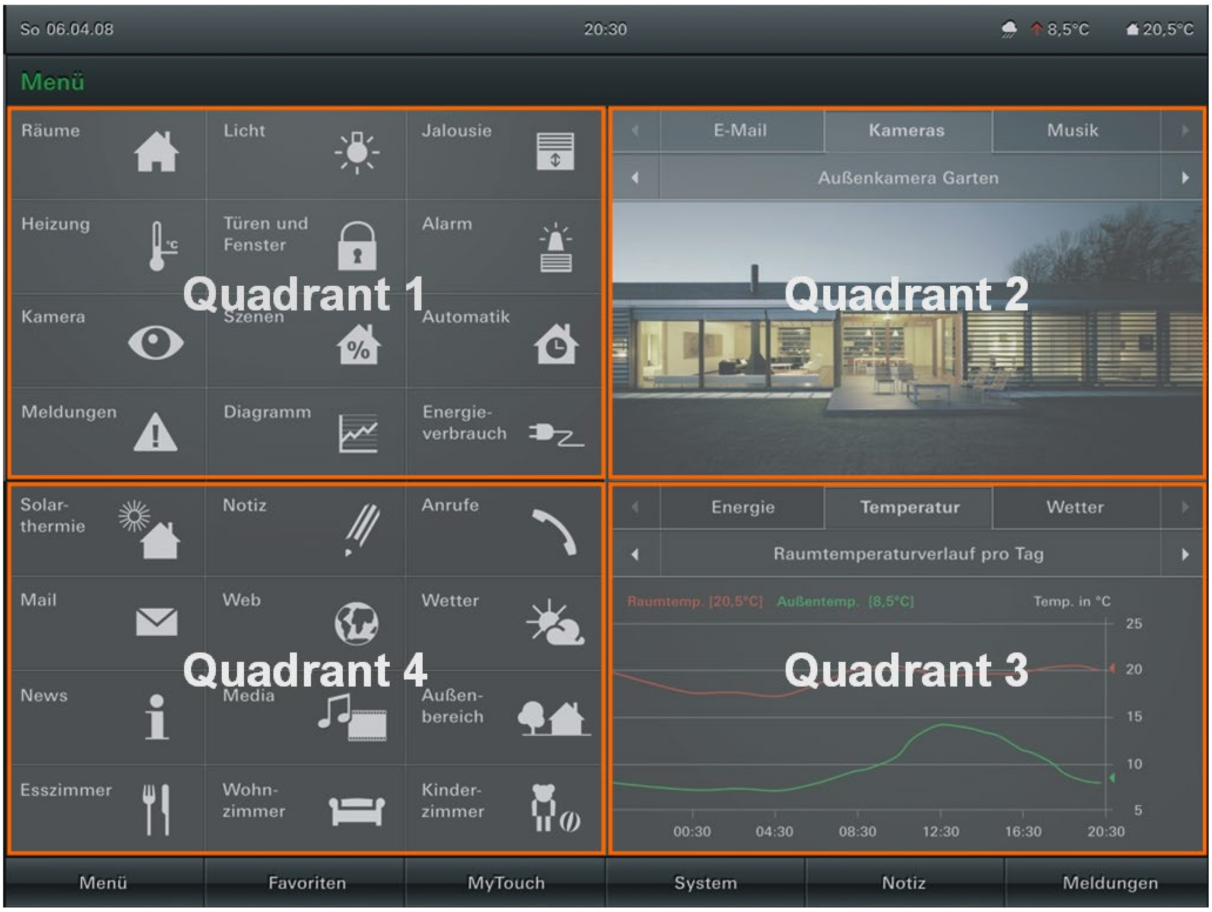


Fig. 1

Self-operating programs (known as 'plug-ins') can be embedded in the QuadClient.
Quadrants 2 and 3 are available for this purpose. Combining quadrants 2 and 3 into one quadrant is not possible in this version.
The plug-ins are parametrised using HomeServer Expert/QuadConfig.
The parameters and the plug-in are saved on the HomeServer/FacilityServer and called up by QuadClient (touch screen/PC).

The following plug-ins are currently included as standard:

- RSS feeds
- HS message archives
- HS graphs
- Cameras
- Audio archive/Audio player
- Email
- Weather
- Energy graph/Energy traffic light
- Camera archive

Definition

A plug-in consists of the following files.
These are packaged together in a single ZIP file. The file extension is hsp.

Naming convention:

Example: `0--1_RSS Feed Reader.hsp`

0-	Developer ID, is managed/assigned by DaCom Database Computing GmbH.
1_	Sequential number of the plug-in.
Dimmer	Any descriptive name for the plug-in. Note: Is displayed in the file dialogue of QC system during import.

Note: 0-1 is called the plug-in ID.

Number ranges of the developer ID

0-	DaCom Database Computing GmbH
1-	IISN Thorsten Neitzke-Dannecker
2-	Thorsten Dreiner, netyard Intelligente Gebäudetechnik GmbH

Plug-in files

version.xml	Basic information about the plug-in C, T
interface.xml	Programming interface definition C, T
"plug-in ID"_lang.xml	Contains the translation data for the plug-in HS
"plug-in-ID" _client.dll	Contains the assembly for the client plug-in Note: In the QuadClient configuration file (config.xml), there is a parameter especially for developers where a path is specified in which this assembly is located. In this case, QuadClient does not use the assembly from the HS/FS but rather the one from the specified path. This means that the assembly does not have to be transferred to the HS/FS after each recompilation. The parameter is: `<assembly path="..\quad_assembly\" />` HS
plugin_config.dll	Contains the assembly for the Config plug-in C
config_lang.xml	This file contains all the language versions for the Config program (QuadConfig) C
config_data.xml	This file contains all the necessary data for the Config program (QuadConfig) C
thumbs\	Subfolder
prev_"language".png	Preview image for the Config program (QuadConfig) Example: thumbs\prev_en.png Note: A separate file is necessary for each language supported by QuadConfig. C
help\	Subfolder

help\p "plug-in ID"_"language".chm	Help for the plug-in in the Config program (QuadConfig). Example: help\p0-1_en.chm The Microsoft HTML Help Workshop can be used as a tool. C

Abbreviations:

C: The file is needed to define the meta information of the plug-in for QuadConfig.

T: The file is needed for the transfer between Expert and HS/FS.

HS: The file is loaded to the HS/FS.

[design_"design_key"]	Folder for the design HS Note: Currently there is only one design with the constant 1: QuadClient displays a continuous list/menu in quadrants 1 and 4 (Fig. 1). Quadrants 2 and 3 can be used. The aspect ratio is 4:3
	Content of the folder:
	"plug-in ID"_design.xml Design information of the plug-in
	"plug-in ID"_icon.zip Xaml icons for the plug-in

version.xml	Basic information about the plug-in Structure:
	<header>
	<version> Version number for error report, also appears in QC system and QuadConfig
	<id> Unique plug-in ID across all plug-ins
	<name> Name of the plug-in This name can also be a translation key from *config_lang.xml.*
	<manufacturer> Name of the developer, appears in QC system and QuadConfig
	<contact> Contact to the developer, appears in QC system and QuadConfig
	<support_gipc> Indicates whether the plug-in is also implemented on the Gira iPhone® Client. true or false
	</header>

Example:

```
<plugin>
<header>
<id>0-1</id>
<version>1.0.080924</version>
<name>#0-1_pluginname</name>
<manufacturer>DaCom Database Computing GmbH</manufacturer>
<contact>http://www.dacom-homeautomation.de</contact>
<support_gipc>true</support_gipc>
</header>
</plugin>
```

interface.xml	Programming interface definition for the integration of the plug-in into QuadConfig and QuadClient
Example: <plugin> <class nsconfig="feedReader.CFeedReader" nsclient="hs_client_quad_rssReader.CQuadRssReader" remanent="1" max_instance_count="0" /> </plugin>	
	<plugin>
	<class
	{nsconfig} Class name including name space of plugin_config.dll (notation: Namespace.classname, e.g. feedReader.CFeedReader). This is specified in the library source code.
	{nsclient} Notation like *nsconfig* Class name incl. name space of the "plug-in ID"_client.dll
	{remanent} = 0/1 (Plug-in may store configuration data on the HS/FS
	{max_instance_count} = maximum number of plug-in instances (quadrants) per user. 0 = no maximum set.
	<master This tag is optional. It is only used if a master plug-in exists (see following line).
	{id} = plug-in ID of the master plug-in. The plug-in does not have its own assembly, but uses the assembly of the master plug-in. Example: Audio archive as master plug-in of the audio player plug-in <plugin> <class nsconfig="audio_player_config.CAudioPlayer" nsclient="hs_client_quad_audio.CQuadAudioPlayer" remanent="0" max_instance_count="1" /> <master id="0-5" /> </plugin>
	</plugin>

"plug-in ID"_lang.xml	Contains the translation data for the plug-in
	<language>
	<text>
	key="#language" default="German">
	<lang id="de" value="Deutsch" /> <lang id="en" value="english" /> <lang id="ru" value="нке" />
	</text>
	</language>

Example:

```
<?xml version="1.0" encoding="utf-8"?>

<language>

  <text key="#language" default="German">
    <lang id="de" value="Deutsch" />
    <lang id="en" value="english" />
    <lang id="ru" value="нке" />
  </text>

</language>
```

"plug-in ID"_client.dll	Contains the assembly for the Client plug-in More information can be found in additional documentation for the SDK: *client_framework.chm*

plugin_config.dll	Contains the assembly for the Config plug-in for Expert/QuadConfig More information can be found in additional documentation for the SDK: *config_framework.chm*

config_lang.xml	This file contains all the necessary language versions for the Config program The structure corresponds to the file "plug-in ID"_lang.xml (see above)

config_data.xml	This file contains configuration data for the plug-in. The QuadConfig program transfers this data to the plug-in when the plug-in is called up.

design_"design-ID"\"plug-in-ID"_design.xml	Design information for the plug-in This file is transferred in QuadClient during start-up and is used to define the interface. The QuadClient framework provides several classes for GUI components. The plug-in can transfer the XML elements to the QuadClient framework during readout and use them to generate and parametrise the components.

Classes

The classes of the framework can process the following XML structures/attributes :

	CTextData: • {left} • {top} • {width} • {align} • 0=left-justified • 1=centred • 2=right-justified • {color} • Text colour (e.g. #000000) • {text} • {size} • Text size in points • {shadow_color} • Shadow colour (e.g. #FFFFFF) • {shadow_x} • Offset of the shadow in X direction • {shadow_y} • Offset of the shadow in Y direction
	CButtonData: • {left} • {top} • {width} • {height} • {ico_up} • Icon name for the button in the normal state • {ico_down} • Icon name for the button in the pressed state • {ico_inactive} • Ico nname for the button in the inactive state • <btn_text> • a CTextData structure • Position information relative to the associated button
	CClickAreaData: • {left} • {top} • {width} • {height} • {ico_down} • Icon name for the pressed state

<table>
<tr><td></td><td><em>CIconPixelData:</em><ul><li>{left}</li><li>{top}</li><li>{width}</li><li>{height}</li><li>{id}<ul><li>Icon name</li></ul></li></ul></td></tr>
<tr><td></td><td><em>CIconVectorData:</em><ul><li>{left}</li><li>{top}</li><li>{width}</li><li>{height}</li><li>{ico}<ul><li>Xaml icon name</li></ul></li></ul></td></tr>
<tr><td></td><td><em>CPanelScrollData:</em><ul><li>{left}</li><li>{top}</li><li>{width}</li><li>{height}</li></ul><ul><li><ico_bg><ul><li>a CIconVectorData structure</li><li>Position information relative to the associated PanelScroll</li><li>Represents the background of the scrollbar</li></ul></li><li><mark><ul><li>a CIconVectorData structure</li><li>Position information relative to the associated PanelScroll</li><li>Represents the slider of the scrollbar</li><li>Has the following additional attributes:<ul><li>{range}<ul><li>Size of the dynamic area in pixels</li></ul></li></ul></li></ul></li></ul></td></tr>
</table>

CPopupClickAreaData:

- {key}
    - Key to identify the ClickArea
- {left}
- {top}
- {width}
- {height}
- {up_ico}
    - Icon name for the normal state
- {down_ico}
    - Icon name for the pressed state
- {sec_ico}
    - Icon name for the secondary state (e.g. inactive)
- {sec_down_ico}
    - Icon name for the pressed secondary state (e.g. inactive)

- <text>
    - a CTextData structure
    - Position information relative to the associated ClickArea
    - Has the following additional attributes:
        - {sec_text}
            - Text for the secondary state
        - {sec_color}
            - Colour for the secondary state (e.g. #FFFFFF)

CAlphaPopupHeaderData:

- <title>
    - a CTextData structure
    - Position information relative to the input mask
- <hint>
    - a CTextData structure
    - Position information relative to the input mask

	CAlphaPopupFieldData: • {is_password} • (0/1) Input field is displayed as password field • <text> • a CTextData structure • Position information relative to the input mask • Contains the title for the input field • <field> • a CPopupClickAreaData structure • Position information relative to the input mask • Contains the input field with background icons.
	CQuadSubControlData: • {top} • {left} • {width} • {height} • <btn_left> • a CButtonData structure • Position information relative to the SubControl • <btn_right> • a CButtonData structure • Position information relative to the SubControl • <txt> • a CTextData structure • Position information relative to the SubControl • <ico_bg> • a CIconVectorData structure Position information relative to the SubControl

Example:

```
<?xml version="1.0" encoding="utf-8"?>

<design>

  <img_bg left="0" top="0" width="507" height="314" />
  <page left="0" top="0" width="507" height="314" />

  <value left="100" top="100" width="200" shadow="1" align="1" font="13" />
  <hint left="100" top="50" width="200" shadow="1" align="1" font="13" />

  <btn_dialog ico_up="play.xaml" ico_down="play_down.xaml" left="205"
top="159" width="95" height="79" />

  <dialog>
    <login_header>
      <title left="10" top="10" width="1000" align="0" font="980"
text="#dialog_header" />
      <hint left="10" top="25" width="1000" align="0" font="981"
text="#dialog_hint" />
    </login_header>

    <input_1>
      <text left="10" top="50" width="1000" align="0" font="980"
text="#field_1" />
      <field left="10" top="70" width="300" height="25"
sec_ico="field_focus.xaml" >
        <text left="10" top="5" width="280" align="0" font="980" />
      </field>
    </input_1>

    <input_2 is_password="1">
      <text left="10" top="150" width="1000" align="0" font="980"
text="#field_2" />
      <field left="10" top="170" width="300" height="25"
sec_ico="field_focus.xaml" >
        <text left="10" top="5" width="280" align="0" font="980" />
      </field>
    </input_2>

    <click_ok key="1" left="5" top="200" width="110" height="50"
down_ico="popup_down_1.xaml" >
      <text font="980" left="5" top="25" width="100" align="0" text="#ok"
/>
    </click_ok>

    <click_esc key="2" left="105" top="200" width="110" height="50"
down_ico="popup_down_1.xaml" >
      <text font="980" left="5" top="25" width="100" align="0" text="#esc"
/>
    </click_esc>

  </dialog>

</design>
```

Technical requirements

.net framework 3.0 or higher
Microsoft Visual Studio 2008 (VS) is recommended

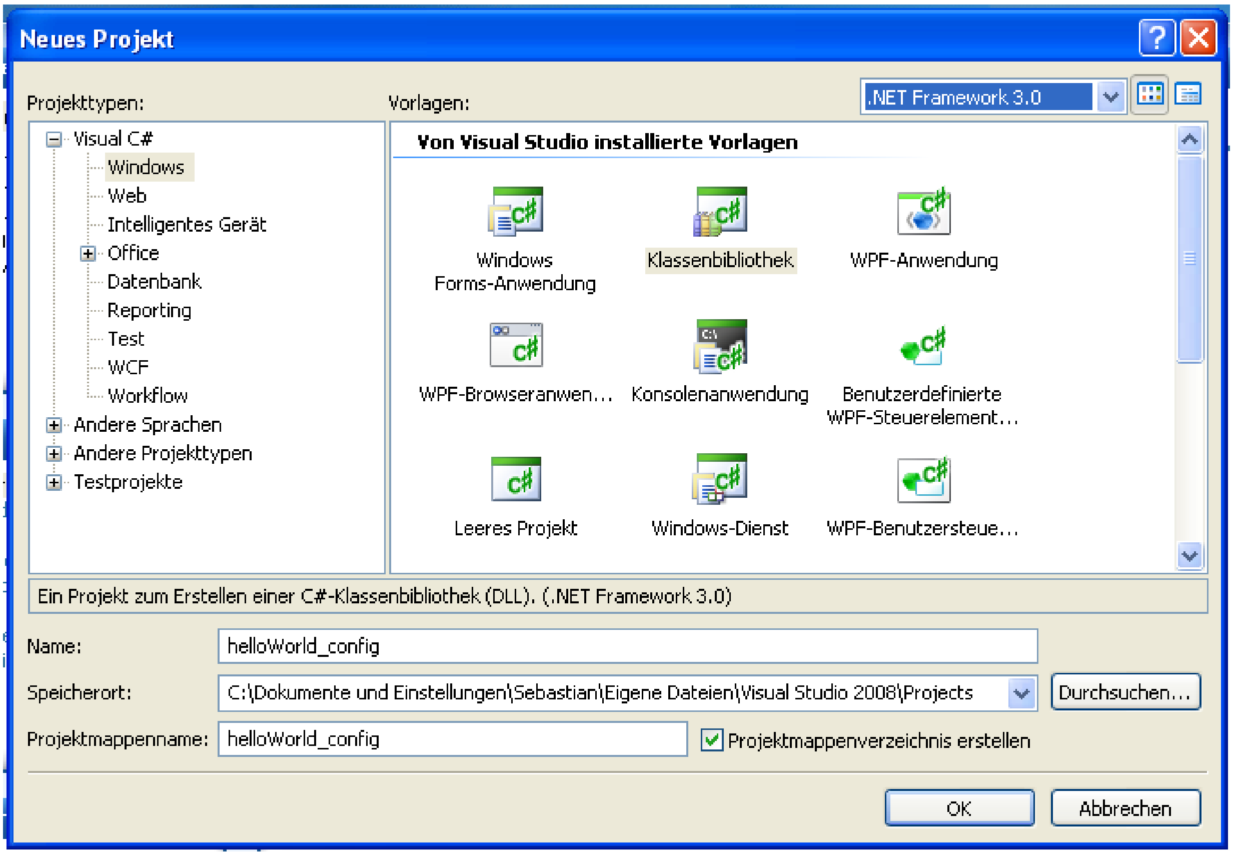
DaCom libraries:
config_framework.dll (directory: Experte\quad_config\config)

and

hs_client_framework.dll
(directory: e.g. Expert\tools\QuadClient)

DaCom-Doku: *client_framework.chm* and *config_framework.chm*

How To – First steps in VS (Assembly creation)



.NET Framework 3.0

Ein Projekt zum Erstellen einer C#-Klassenbibliothek (DLL). (.NET Framework 3.0)

Name: helloWorld_config

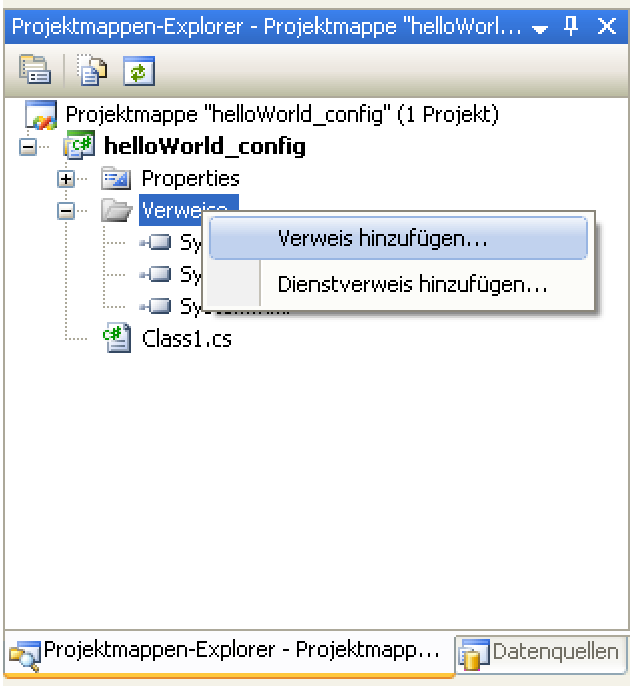
Speicherort: C:\Dokumente und Einstellungen\Sebastian\Eigene Dateien\Visual Studio 2008\Projects

Projektmappenname: helloWorld_config

☑ Projektmappenverzeichnis erstellen

1. Create a new class library. For the QuadConfig plug-in and the QuadClient plug-in, you need a separate project for each.

2. Add a reference to config_framework.dll or hs_client_framework.dll to the project.


Projektmappen-Explorer - Projektmappe "helloWorl...
Projektmappe "helloWorld_config" (1 Projekt)
helloWorld_config
Properties
Verweise
Sy
Sy
Sy
Class1.cs
Verweis hinzufügen...
Dienstverweis hinzufügen...
Projektmappen-Explorer - Projektmapp...
Datenquellen

Note:
The config_framework.dll file is located in QuadConfig's root directory (EXP\quad_config\config\).

The file hs_client_framework.dll is located in QuadClient's root directory.

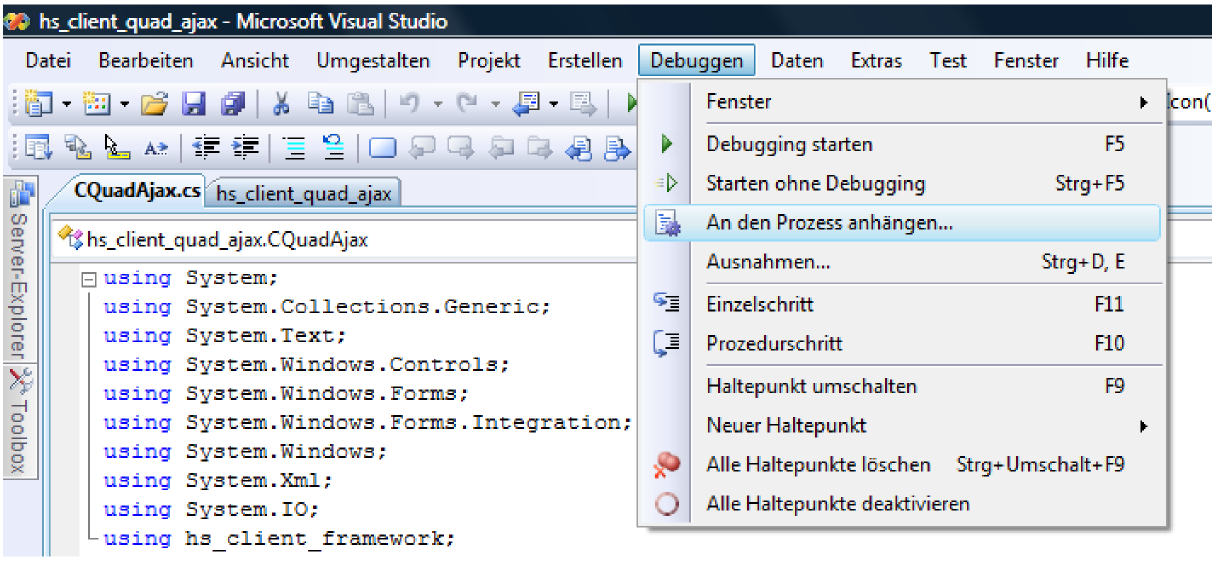
3. Define project properties
4. Define classes
5. Start programming ;)

How To - First steps (HelloWorld)

1. Unzip the file helloworldlplugin.zip into any folder.
2. Open the file \hs_client_quad_helloworld\hs_client_quad_helloWorld.sln with VS 2008.
3. Select the hs_client_quad_helloworld project in Solution Explorer.
4. Expand the "References" item and remove the incorrect reference.
5. Add the reference to hs_client_framework (e.g. '\Experte\tools\QuadClient').
6. Compile the project.
7. Select the helloworld_config project in Solution Explorer.
8. Expand the "References" item and remove the incorrect reference.
9. Add the reference to hs_config_framework ('\Experte\quodconfig\config').
10. Compile the project.
11. Copy the plug-ins folder to the EXP/quad/ folder (Overwrite: Yes). Note: Make sure that the output paths of the projects point to the corresponding subdirectory of the plug-ins folder. If this is not the case, the assemblies must be manually moved from the debug/release folder to the corresponding subdirectory of the plug-in (see also "List of files" above).
12. Create a quadrant of the type HelloWorld in QuadConfig, assign a KO to it and transfer the project to the HS.

How To - Troubleshooting

In the QuadClient configuration file (config.xml), there is a parameter especially for developers where a path is specified in which the assembly of the plug-in is located. In this case, QuadClient does not use the assembly from the HS/FS but rather the one from the specified path.
This means that the assembly does not have to be transferred to the HS/FS after each recompilation.
The parameter is: `<assembly path="..\quad_assembly\" />`
With the VS, it is possible to attach to the QuadClient process and so debug the plug-in "live" (see the following figure). To do so, the process "hs_client_net.exe" must be selected in the selection menu "Attach to process". In order for the debugger to be attached to the QuadClient process, it should be in the login mask, since all the plug-ins are started immediately after login.



```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Controls;
using System.Windows.Forms;
using System.Windows.Forms.Integration;
using System.Windows;
using System.Xml;
using System.IO;
using hs_client_framework;
```

Example source code

The following source codes are included in the SDK:

- HelloWorld (helloworldlplugin.zip)
  The plug-in displays the value of a K-object and can switch between 0 and 1 via a switch button.

- Message archive*(hs_client_quad_meldungsarchive.sln*)
  Display of the HS/FS message archive.

Notes

It is mandatory to use the references to the hs_client_framework ('\Experte\tools\QuadClient') and the config_framework ('Expert\quad_config\config')

- 'Solution Explorer' (located at the top right by default) contains the item References
- Expand references
- Remove faulty reference (represented by a yellow warning symbol)
- Add reference -> Search -> select config_framework.dll or hs_client_framework.dll

Notes on the sample source codes

Plug-in 0-4_message_archive:

Folder structure:

\hs_client_quad_message_archive	Project for the Client plug-in
\message_archives_config	Project for the QuadConfig plug-in
\plugins	Plug-in directory into which the compiled assemblies are copied

XML files

In the first line of an XML file, a code page is specified (usually this is "UTF-8").

This line looks something like this:

`<?xml version="1.0" encoding="utf-8"?>`

If the XML file is NOT saved in the mode specified in the header, an error may occur if a character with an ASCII code greater than 127 is used in the XML file (e.g. ö,ä,ü, etc.)
If you are using an editor that is unable to use a specific code page when saving, and if you are using a Windows operating system for programming, you should enter the code page "ISO-8859-1" in the header, since this is almost identical, as far as printable characters are concerned, to the Windows ANSI code page 1252 used by "normal" editors such as Notepad, the standard Windows editor.

You can read more about this in this Wikipedia entry.

The following generally applies:
The XML file must be saved with the code page specified in the header.

Changes in the framework

09 October 2008	Change in config_framework.dll: IplugIn has changed: - SaveToArray(): Attribute _lst is omitted (the GetLinks methods is used as a replacement). - GetLinks(): Returns a list with all the references to project data contained in the plug-in - Validate(): Method is now called up by QuadConfig. New parameter available. Validate is used to display error messages or warnings in QuadConfig. - New class CPluginErrorItem

Text versions

18 November 2010	Chapter "XML Files" added
11 March 2010	<support_gipc> added to version.xml
01 December 2009	Section "How To – Troubleshooting" added
22 December 2008	First version for SDK developers
09 October 2008	Notes on the sample project Changes in the framework
08 October 2008	config_data.xml changed help.chm added Sample project
07 October 2008	1st pre-release version