Bildnachweis: www.gira.de

Inhalt

Was ist ein PlugIn?

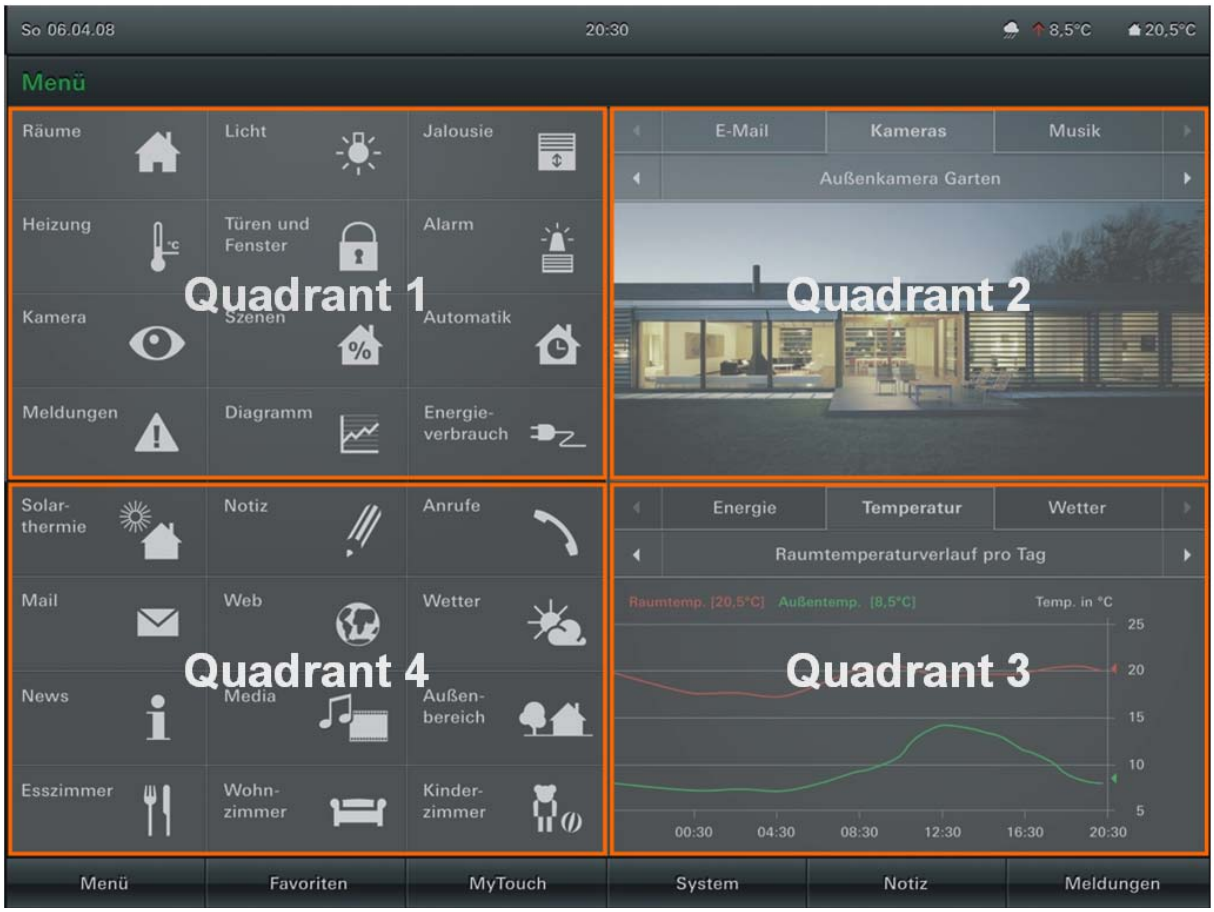


Abb. 1

In den QuadClient können selbstständig arbeitende Programme (sogenannte PlugIns) eingebettet werden.
Hierzu stehen die Quadranten 2 und 3 zur Verfügung. Eine Zusammenfassung der Quadranten 2 und 3 in einen Quadranten ist in dieser Version nicht möglich.
Die PlugIns werden mit dem HomeServer-Experten/QuadConfig parametrisiert. Die Parameter und das PlugIn werden auf dem HS/FS abgelegt und von QuadClient (Touchscreen/PC) aufgerufen.

Aktuell sind folgende PlugIns im Lieferumfang:

- RSS-Feeds
- HS-Meldungsarchive
- HS-Diagramme
- Kameras
- Audio-Archiv/AudioPlayer
- eMail
- Wetter
- Energiegraph/Energieampel
- Kamera-Archiv

Definition

Ein PlugIn besteht aus den folgenden Dateien.
Diese werden komplett in einer ZIP-Datei zusammengefasst. Als Dateiendung ist hsp vorgesehen.

Namenskonvention:

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

0-	Entwickler ID, wird von DaCom Database Computing GmbH verwaltet/vergeben.
1_	Fortlaufende Nummer des PlugIns.
Dimmer	Beliebige, sprechende Bezeichnung des PlugIns. Anmerkung: Wird beim Import im Dateidialog von QC-System angezeigt.

Anmerkung: 0-1 wird als PlugIn-ID bezeichnet.

Nummernkreise der Entwickler ID

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

Dateien eines PlugIns

version.xml	Grundsätzliche Angaben zum PlugIn C, T
interface.xml	Programmierschnittstelle-Definition C, T
„PlugIn-ID“_lang.xml	Enthält die Übersetzungsdaten für das PlugIn HS
„PlugIn-ID" _client.dll	Enthält das Assembly für das Client-PlugIn Anmerkung: In der Konfigurationsdatei des QuadClients (config.xml) gibt es einen Parameter speziell für Entwickler. Hier wird ein Pfad angegeben, in dem dieses Assembly liegt. Der Quad Client verwendet dann nicht das Assembly vom HS/FS, sondern das aus dem angegebenen Pfad. Somit muss das Assembly nicht nach jeder Neucompilierung auf den HS/FS übertragen werden. Der Parameter lautet: <assembly path="..\quad_assembly\" /> HS
plugin_config.dll	Enthält das Assembly für das Config-PlugIn C
config_lang.xml	Diese Datei enthält alle Sprachversionen für das Config-Programm (QuadConfig) C
config_data.xml	Diese Datei enthält alle nötigen Daten für das Config-Programm (QuadConfig) C
thumbs\	Unterordner
prev_“Sprache“.png	Vorschaubild für das Config-Programm (QuadConfig) Beispiel: thumbs\prev_de.png Anmerkung: Für jede Sprache, die vom QuadConfig unterstützt wird, ist eine eigene Datei notwendig. C
help\	Unterordner

help\p„PlugIn-ID"_“Sprache“.chm	Hilfe für das PlugIn im Config-Programm (QuadConfig). Beispiel: help\p0-1_de.chm Als Werkzeug kann der Microsoft HTML Help Workshop verwendet werden. C

Abkürzungen:

C: Die Datei wird zur Definition der Metainformationen des PlugIns für den QuadConfig benötigt.

T: Die Datei wird für den Transfer zwischen Experte und HS/FS benötigt.

HS: Die Datei wird auf den HS/FS geladen.

[design_"design_key"]	Ordner für das Design HS Anmerkung: Aktuell existiert nur ein Design mit der Konstante 1: Im QuadClient wird in den Quadranten 1 und 4 (Abb.1) eine durchgehende Liste/Menü angezeigt. Quadrant 2 und 3 können belegt werden. Das Seitenverhältnis ist 4:3
	Inhalt des Ordners:
	„PlugIn-ID“_design.xml Design-Informationen des PlugIns
	„PlugIn-ID“_icon.zip Xaml-Icons für das Plugin

version.xml	Grundsätzliche Angaben zum PlugIn Struktur:
	<header>
	<version> Versionsnummer für Fehlerreport, erscheint auch im QC-System und QuadConfig
	<id> Eindeutige PlugIn-ID über alle PlugIns hinweg
	<name> Bezeichnung des PlugIns Dieser Name kann auch ein Übersetzungsschlüssel aus der *config_lang.xml* sein.
	<manufacturer> Name des Entwicklers, erscheint im QC-System und QuadConfig
	<contact> Kontakt zum Entwickler, erscheint im QC-System und QuadConfig
	<support_gipc> Gibt an ob das PlugIn auch auf dem Gira iPhone® Client implementiert ist. true oder false
	</header>

Beispiel:

```
<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	Programmierschnittstelle-Definition für die Einbindung des PlugIns in den QuadConfig und QuadClient
Beispiel: <plugin> <class nsconfig="feedReader.CFeedReader" nsclient="hs_client_quad_rssReader.CQuadRssReader" remanent="1" max_instance_count="0" /> </plugin>	
	<plugin>
	<class
	{nsconfig} Klassenname inkl. Namespace der plugin_config.dll (Schreibweise: Namespace.Klassename, z. B. feedReader.CFeedReader). Diese wird im Quellcode der Bibliothek festgelegt.
	{nsclient} Schreibweise wie *nsconfig* Klassenname inkl. Namespace der „PlugIn-ID“_client.dll
	{remanent} = 0/1 (Plugin darf Konfigurationsdaten auf dem HS/FS speichern
	{max_instance_count} = maximale Anzahl der PlugIn-Instanzen (Quadranten) pro Benutzer. 0 = kein Maximum gesetzt.
	<master Dieser Tag ist optional. Dieser wird nur dazu benutzt wenn ein Master-PlugIn vorhanden ist (siehe folgende Zeile).

	{id} = PlugIn-ID des MasterplugIns Das PlugIn hat kein eigenes Assembly, sondern benutzt das Assembly des MasterplugIns. Beispiel: Audio-Archiv als Master-PlugIn des Audio-Player-Plugins <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>

„PlugIn-ID“_lang.xml	Enthält die Übersetzungsdaten für das PlugIn
	<language>
	<text>
	key="#Sprache" default="Deutsch">
	<lang id="de" value="Deutsch" /> <lang id="en" value="english" /> <lang id="ru" value="нке" />
	</text>
	</language>

Beispiel:

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

<language>

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

</language>
```

"PlugIn-ID "_client.dll	Enthält das Assembly für das Client-PlugIn Nähere Infos gibt es in einer Zusatzdoku zum SDK: *client_framework.chm*

plugin_config.dll	Enthält das Assembly für das Config-PlugIn für den Experten/QuadConfig Nähere Infos gibt es in einer Zusatzdoku zum SDK: *config_framework.chm*

config_lang.xml	Diese Datei enthält alle nötigen Sprachversionen für das Config-Programm Der Aufbau entspricht der Datei „PlugIn-ID“_lang.xml (siehe oben)

config_data.xml	Diese Datei enthält Konfigurationsdaten für das PlugIn. Das QuadConfig-Programm übergibt diese Daten beim Aufruf des PlugIns an das PlugIn.

design_“design-ID“\"PlugIn-ID"_design.xml	Design-Informationen für das PlugIn Diese Datei wird beim Start im QuadClient übergeben und dient zur Definition der Oberfläche. Das QuadClient-Framework stellt mehrere Klassen für GUI-Komponenten zur Verfügung. Das PlugIn kann die XML-Elemente beim Auslesen dem QuadClient-Framework übergeben und damit die Komponenten generieren und parametrisieren.

Klassen

Die Klassen des Frameworks können folgende XML-Strukturen/Attribute verarbeiten :

	CTextData:
	• {left}
	• {top}
	• {width}
	• {align}
	  • 0=linksbündig
	  • 1=zentriert
	  • 2=rechtsbündig
	• {color}
	  • Textfarbe (z.B. #000000)
	• {text}
	• {size}
	  • Textgrösse in Punkten
	• {shadow_color}
	  • Farbe des Schattens (z.B. #FFFFFF)
	• {shadow_x}
	  • Ofset des Schattens in X-Richtung
	• {shadow_y}
	  • Ofset des Schattens in Y-Richtung

	CButtonData:
	• {left}
	• {top}
	• {width}
	• {height}
	• {ico_up}
	  • Iconname für die Taste im Normalzustand
	• {ico_down}
	  • Iconname für die Taste im gedrückten Zustand
	• {ico_inactive}
	  • Iconname für die Taste im inaktiven Zustand
	• <btn_text>
	  • eine CTextData-Struktur
	  • Positionsangaben relativ zum zugehörigen Button

	CClickAreaData: • {left} • {top} • {width} • {height} • {ico_down} • Iconname für den gedrückten Zustand

	CIconPixelData: • {left} • {top} • {width} • {height} • {id} • Name des Icons
	CIconVectorData: • {left} • {top} • {width} • {height} • {ico} • Name des Xaml-Icons

	CPanelScrollData:
	• {left}
	• {top}
	• {width}
	• {height}
	
	• <ico_bg>
	  • eine CIconVectorData-Struktur
	  • Positionsangaben relativ zum zugehörigen PanelScroll
	  • Stellt den Hintergrund des Scrollbars dar
	• <mark>
	  • eine CIconVectorData-Struktur
	  • Positionsangaben relativ zum zugehörigen PanelScroll
	  • Stellt den Slider des Scrollbars dar
	  • Hat folgende Zusatzattribute:
	  • {range}
	  • Größe des dynamischen Bereiches in Pixel

	CPopupClickAreaData:
	• {key}
	  • Schlüssel zur Identifizierung der ClickArea
	• {left}
	• {top}
	• {width}
	• {height}
	• {up_ico}
	  • Iconname für den Normalzustand
	• {down_ico}
	  • Iconname für den gedrückten Zustand
	• {sec_ico}
	  • Iconname für den Sekundärzustand (z.B. Inaktiv)
	• {sec_down_ico}
	  • Iconname für den gedrückten Sekundärzustand (z.B. Inaktiv)
	
	• <text>
	  • eine CTextData-Struktur
	  • Positionsangaben relativ zur zugehörigen ClickArea
	  • Hat folgende Zusatzattribute:
	  • {sec_text}
	  • Text für den Sekundärzustand
	  • {sec_color}
	  • Farbe für den Sekundärzustand (z.B. #FFFFFF)

	CAlphaPopupHeaderData:
	• <title>
	  • eine CTextData-Struktur
	  • Positionsangaben relativ zur Eingabemaske
	• <hint>
	  • eine CTextData-Struktur
	  • Positionsangaben realtiv zur Eingabemaske

	CAlphaPopupFieldData:
	• {is_password}
	  • (0/1) Eingabefeld wird als Passwordfeld dargestellt
	• <text>
	  • eine CTextData-Struktur
	  • Positionsangaben relativ zur Eingabemaske
	  • Enthält den Titel für das Eingabefeld
	• <field>
	  • eine CPopupClickAreaData-Struktur
	  • Positionsangaben relativ zur Eingabemaske
	  • Enthält das Eingabefeld mit Hintergrundicons.

	CQuadSubControlData:
	• {top}
	• {left}
	• {width}
	• {height}
	• <btn_left>
	  • eine CButtonData-Struktur
	  • Positionsangaben relativ zum SubControl
	• <btn_right>
	  • eine CButtonData-Struktur
	  • Positionsangaben relativ zum SubControl
	• <txt>
	  • eine CTextData-Struktur
	  • Positionsangaben relativ zum SubControl
	• <ico_bg>
	  • eine CIconVectorData-Struktur
	Positionsangaben relativ zum SubControl

Beispiel:

```
<?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>
```

Technische Voraussetzungen

.net-Framework ab 3.0
Empfohlen wird Microsoft Visual Studio 2008 (VS)

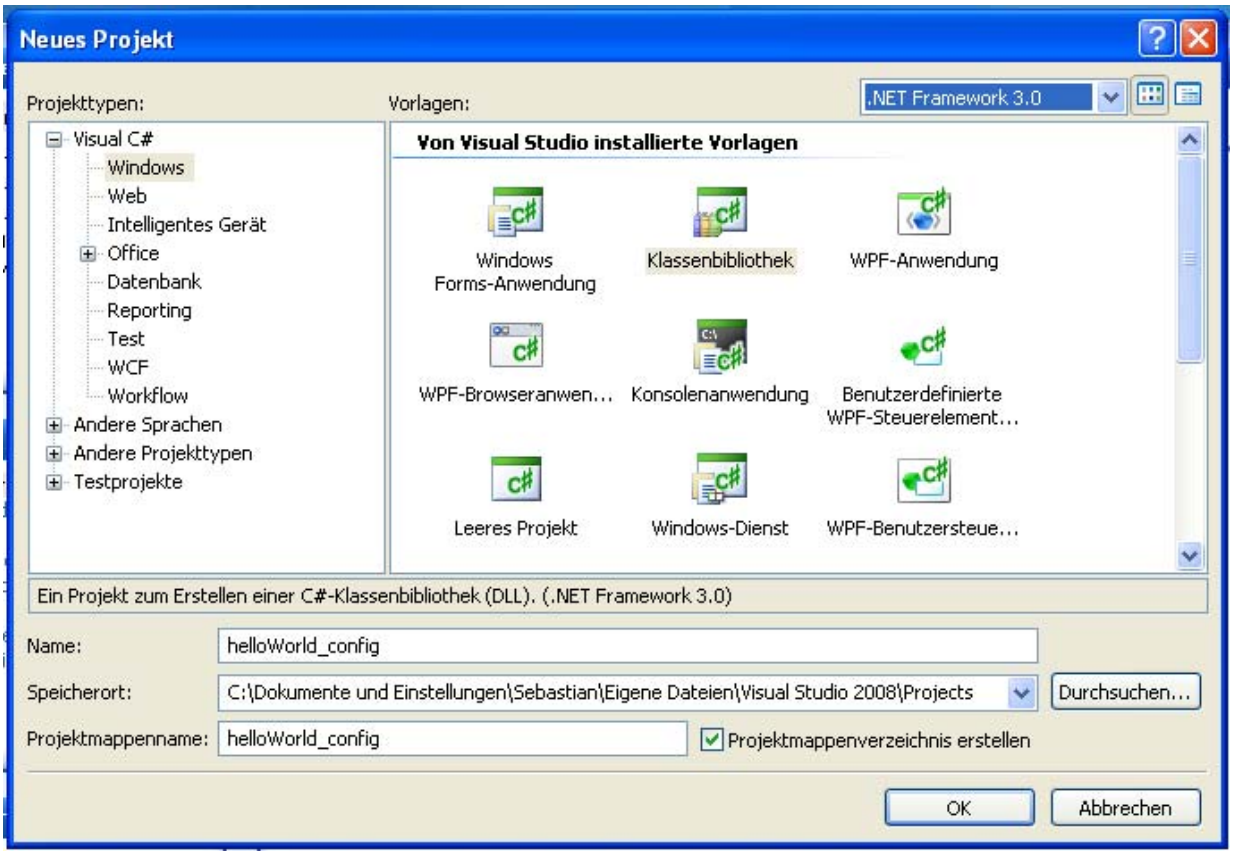
DaCom-Bibliotheken:
config_framework.dll (Verzeichnis: Experte\quad_config\config)

und

hs_client_framework.dll
(Verzeichnis: z. B. Experte\tools\QuadClient)

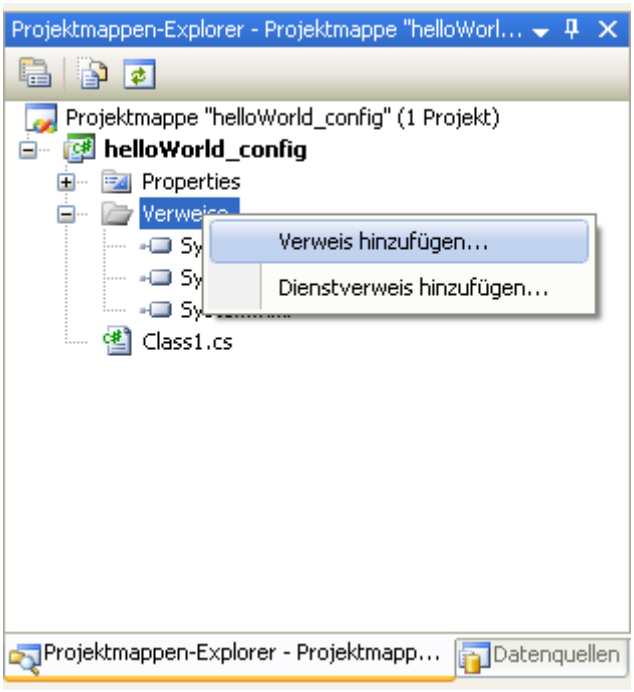
DaCom-Doku: *client_framework.chm* und *config_framework.chm*

HowTo – Erste Schritte in VS (Assembly-Erstellung)



1. Neue Klassenbibliothek anlegen. Für das QuadConfig-PlugIn und das QuadClient-Plugin ist jeweils ein eigenes Projekt notwendig.

2. Dem Projekt einen Verweis auf die config_framework.dll bzw. hs_client_framework.dll hinzufügen.



Anmerkung:
Die Datei config_framework.dll befindet sich im Hauptverzeichnis des QuadConfigs (EXP\quad_config\config\).

Die Datei hs_client_framework.dll befindet sich im Hauptverzeichnis des QuadClients.

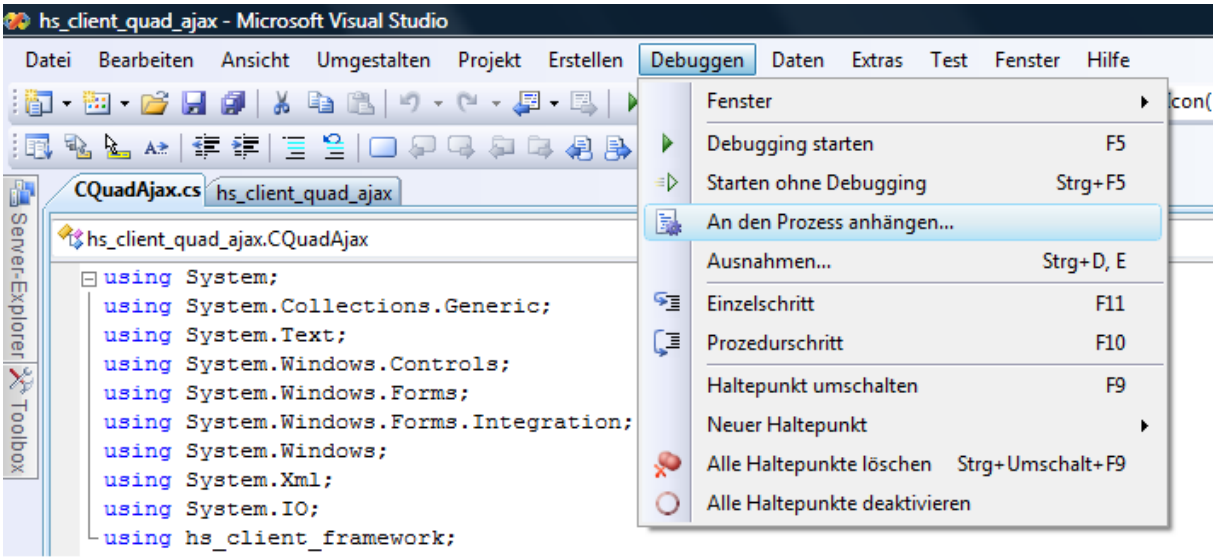
3. Projekteigenschaften festlegen
4. Klassen definieren
5. Start der Programmierung ;)

HowTo – Erste Schritte (HelloWorld)

1. Entpacken Sie die Datei helloworldlplugin.zip in einen beliebigen Ordner.
2. Öffnen Sie die Datei \hs_client_quad_helloworld\hs_client_quad_helloWorld.sln mit VS 2008.
3. Wählen Sie das Projekt hs_client_quad_helloworld im Projektmappen-Explorer aus.
4. Klappen Sie den Punkt „Verweise“ auf und entfernen Sie den fehlerhaften Verweis.
5. Fügen Sie den Verweis auf das hs_client_framework (z. B. '\Experte\tools\QuadClient') hinzu.
6. Kompilieren Sie das Projekt.
7. Wählen Sie das Projekt helloworld_config im Projektmappen-Explorer aus.
8. Klappen Sie den Punkt „Verweise“ auf und entfernen Sie den fehlerhaften Verweis.
9. Fügen Sie den Verweis auf das hs_config_framework ('\Experte\quodconfig\config') hinzu.
10. Kompilieren Sie das Projekt.
11. Kopieren Sie den Ordner plugins in den Ordner EXP/quad/ (Überschreiben: Ja). Anmerkung: Stelln Sie sicher, dass die Ausgabepfade der Projekte in das entsprechende Unterverzeichnis des PlugIns-Ordners zeign. Sollte dies nicht der Fall sein, müssen die Assemblys von Hand aus dem Debug-/Release-Order in das entsprechende Unterverzeichnis des PlugIns (siehe auch oben „Liste der Dateien).
12. Legen Sie im QuadConfig einen Quadranten vom Typ HelloWorld an, weisen Sie diesem ein KO zu und übertragen Sie das Projekt auf den HS.

HowTo – Fehlersuche

In der Konfigurationsdatei des QuadClients (config.xml) gibt es einen Parameter speziell für Entwickler. Hier wird ein Pfad angegeben, in dem das Assembly des PlugIns liegt. Der QuadClient verwendet dann nicht das Assembly vom HS/FS sondern das aus dem angegebenen Pfad.
Somit muss das Assembly nicht nach jeder Neucompilierung auf den HS/FS übertragen werden.
Der Parameter lautet: `<assembly path="..\quad_assembly\" />`
Mit dem VS ist es möglich an den Prozess des QuadClients anzuhängen und somit das PlugIn „live“ zu debuggen (siehe folgende Abbildung). Dazu muss im Auswahlmenu von „An den Prozess anhängen“ der Prozess „hs_client_net.exe“ ausgewählt werden. Damit der Debugger an den QuadClient-Prozess angehangen werden kann, sollt dieser sich in der Anmeldemaske befinden, da direkt nach der Anmeldung alle PlugIns gestartet werden.



Beispielquellcode

Folgende Quellcodes liegen dem SDK bei:

- HelloWorld (helloworldlplugin.zip)
  Das PlugIn zeigt den Wert eines K-Objektes an und kann über einen Schaltknopf zwischen 0 und 1 umschalten.

- Meldungsarchiv (*hs_client_quad_meldungsarchive.sln*)
  Anzeige des HS/FS-Meldungsarchives.

Anmerkungen

Es sind unbedingt die Verweise auf das hs_client_framework ('\Experte\tools\QuadClient') und das config_framework ('Experte\quad_config\config')

- Im 'Projektmappen-Explorer' (standardmäßig rechts oben zu finden) gibt es den Punkt Verweise
- Verweise aufklappen
- Fehlerhaften Verweis (wird dargestellt durch ein gelbes Warnsymbol) entfernen
- Verweis hinzufügen -> Durchsuchen -> config_framework.dll bzw. hs_client_framework.dll auswählen

Hinweise zu den Beispielquellcodes

PlugIn 0-4_Meldungsarchiv:

Ordnerstruktur:

\hs_client_quad_meldungsarchive	Projekt für das Client-PlugIn
\meldungsarchive_config	Projekt für das QuadConfig-PlugIn
\plugins	PlugInverzeichnis in das die kompilierten Assemblies kopiert werden

Änderungen im Framework

09.10.2008	Änderung der config_framework.dll: IPlugIn hat sich geändert: - SaveToArray(): Attribut _lst entfällt (Als Ersatz dient die Methode GetLinks). - GetLinks(): Liefert eine Liste mit allen Verweisen des PlugIns auf Projektdaten - Validate(): Methode wird nun von QuadConfig aufgerufen. Neuer Parameter vorhanden. Validate dient dazu, um Fehlermeldungen oder Warnungen in QuadConfig anzuzeigen. - Neue Klasse CPluginErrorItem

Text-Versionen

11.03.2010	<support_gipc> in version.xml ergänzt
01.12.2009	Abschnitt „HowTo – Fehlersuche“ ergänzt
22.12.2008	Erste Version für SDK-Entwickler
09.10.2008	Hinweise zum Musterprojekt Änderungen im Framework
08.10.2008	config_data.xml geändert help.chm hinzugefügt Musterprojekt
07.10.2008	1. Vorabversion

Fragen/Korrekturvorschläge

Anregungen zum Dokument richten Sie bitte an
Oliver Herrmann per eMail: oh@dacom-ha.de