



Projekt: Geogrid[®] - Plugin

Systembeschreibung

Dok. Nr.: TA2351-240000A/00

Datum: 01.07.2005


Ausgabe: A


Blattzahl: 89

ÄNDERUNGSÜBERSICHT

Ausgabe	Datum	Änderung / Änderungsnummer	Geänderte Seiten
A	01.07.2005	Erstausgabe	alle

VERTEILERLISTE

Firma	Name	Org.-Einheit	Funktion	Anzahl
		IRGS61		Projektserver 

 in elektronischer Form

INHALTSVERZEICHNIS

ÄNDERUNGSÜBERSICHT	2
VERTEILERLISTE	2
INHALTSVERZEICHNIS	3
1 ALLGEMEINES	5
1.1 Anwendungsbereich.....	5
1.2 Gegenstand	5
1.3 Dokumentänderung.....	5
1.4 Anzuwendende Dokumente	5
1.5 Abkürzungen und Begriffe.....	6
1.6 Vorbehalt.....	6
2 PLUGIN-SCHNITTSTELLE	7
2.1 Einleitung	7
2.2 Die Plugin-Schnittstellen-CD.....	7
2.3 Das Interface Control Document (ICD)	8
2.4 Das Software Development Kit (SDK)	9
2.5 Die Geogrid®-Fremd-System-Schnittstelle (FSS)	9
3 SYSTEMBESCHREIBUNG	10
3.1 Plugin.....	12
3.1.1 Die Verbindung zwischen Client und Server herstellen.....	13
3.1.2 DLL-Plugin.....	13
3.1.3 EXE-Plugin	15
4 PLUGINS ERSTELLEN	16
4.1 Hello World DLL-Plugin.....	16
4.1.1 MFC-Anwendungsassistent	16
4.1.2 Projekt-Einstellungen	18
4.1.3 Zu exportierende DLL Funktionen.....	23
4.1.4 Lizenz-Schlüssel	25
4.1.5 Ergänzungen in HelloWorld.cpp.....	25
4.1.6 Implementierung der Schnittstelle IGeoPlugin	26
4.1.7 Das Plugin testen	29
4.1.8 Zusammenfassung.....	30
4.2 Hello World EXE-Plugin	32
4.2.1 MFC-Anwendungsassistent	32
4.2.2 Projekt-Einstellungen	35
4.2.3 Lizenz-Schlüssel	39
4.2.4 Implementierung der Schnittstelle IGeoPlugin	40
4.2.5 Ergänzungen in HelloWorld.cpp.....	42
4.2.6 (Klassenassistent) Verbindung trennen – ExitInstance	47

4.2.7	Das Plugin testen	48
4.3	Hello World in einer .Net-Umgebung	49
4.3.1	Projekteinstellungen	51
4.4	HelloWorld-Plugin with Graphic	63
4.4.1	Fehlerbehandlung	64
4.4.2	Graphik erzeugen	65
4.4.2.1	Graphikeinstellungen	65
4.4.3	Dokument / Kartenfenster	66
4.4.3.1	Aktuelle Mittelpunktcoordinate des sichtbaren Bereichs	68
4.4.4	Zugriff auf Graphik-Daten	68
4.4.5	Hello World mit Graphic-Events	72
4.5	Hello World mit Menü	76
4.5.1	Ein Hauptmenü in die Geogrid®-Menüzeile einfügen	76
4.5.2	Popup Menüs	81
4.5.2.1	Popup Menüs erstellen und anzeigen	81
4.5.2.2	Erweitern von Geogrid® Popup-Menüs	82
4.5.2.3	Untermenüs	83
4.5.2.4	Toolbars	85
4.5.2.5	Was ist noch möglich?	86
4.5.2.6	Was geht nicht?	88
4.6	Modes	88

1 ALLGEMEINES

1.1 Anwendungsbereich

Dieses Dokument gilt im Rahmen des Projekts Geogrid® - Plugin.

1.2 Gegenstand

Dieses Dokument enthält die umfassende Beschreibung der Plugin-Schnittstelle. Es gibt einen Überblick über die Möglichkeiten, die sich einem Anwendungsentwickler bieten, um Geogrid® durch eigene Funktionalitäten zu erweitern. Anhand von Beispielen werden unterschiedliche Anwendungsgebiete aufgezeigt. Der Umgang mit dem Software Development Kit (SDK) wird anhand von Beispielen behandelt

1.3 Dokumentänderung

Änderungen und Ergänzungen, die sich aus Erkenntnissen während der Realisierung ergeben, werden über eine Neuausgabe allen Beteiligten zur Kenntnis gebracht. Auf diese Weise werden Änderungen und Erweiterungen dokumentiert. Die Verteilung des Nachtrags oder der Neuausgabe richtet sich nach dem Verteilerblatt.

1.4 Anzuwendende Dokumente

Sofern nicht anders erwähnt, gilt jeweils die zuletzt gültige Ausgabe der Referenzdokumente zum Zeitpunkt der Ausgabe dieses Dokuments.

- /1/ Geogrid® Projekthandbuch [PHB]
(PL2351-000000A/01)
- /2/ Geogrid® Abkürzungen und Begriffsdefinitionen
(TN2351-000000A/01)
- /3/ Geogrid® FSS
(IS2351-180000A/20)

1.5 Abkürzungen und Begriffe

Siehe /2/.

1.6 Vorbehalt

Hinsichtlich der detaillierten Ausführung und Auslegung der Bildschirmpräsentationen, Menü-Oberflächen und Dialogboxen behält sich die Fa. EADS Deutschland GmbH vor, technische Änderungen vorzunehmen.

2 PLUGIN-SCHNITTSTELLE

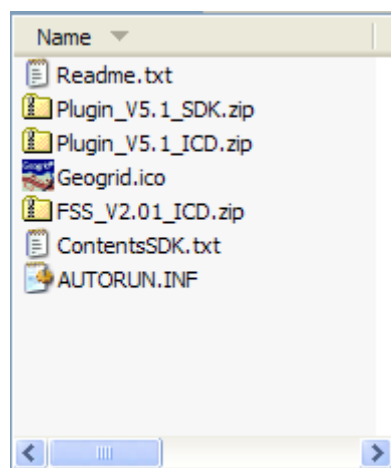
2.1 Einleitung

Die vorliegende Beschreibung gibt eine Einführung in die Plugin-Architektur und erläutert anhand von einfachen Beispielen, wie ein Plugin funktioniert. Natürlich kann dabei nicht auf alle Aspekte eingegangen werden. Die Beispiele sollen einem Entwickler genügend Einblicke geben, um sich mit Hilfe der Plugin-Schnittstellen-Beschreibung (Interface Control Document = ICD) und der Programme im Ordner *samples* des entpackten SDK, weiteres Wissen im Umgang mit der Plugin-Schnittstelle anzueignen.

Einen Überblick über alle verfügbaren Methoden erhalten Sie, wenn Sie im Plugin-ICD in der Start-Seite auf **Plugin History** klicken. Damit gelangen Sie zu einer Tabelle, in der alle Schnittstellen mit ihrer Versionsnummer aufgeführt sind. Von dort kommen Sie zu Tabellen, in denen die zugehörigen Methoden aufgelistet sind. Die Methoden sind in der Regel so benannt, dass sich ihre Funktion aus ihrem Namen erschließen lässt. Aus der Tabelle können Sie aber auch direkt zu den Beschreibungen gelangen. Während des Programmierens können Sie über den Index oder die Volltextsuche zur Beschreibung der gewünschten Methode springen. In dem bereits erwähnten Ordner *samples* finden Sie zu jeder Methode ein Anwendungsbeispiel. Zum Auffinden eines Beispiels verwenden Sie am Besten die Suchfunktion in Ihrer Entwicklungsumgebung.

2.2 Die Plugin-Schnittstellen-CD

Auf der CD „Geogrid-Schnittstelle (Interface)“ befinden sich in komprimierter Form, als so genannte ZIP-Archive, die nachfolgenden Dateien:



Beschreibung des CD-ROM Inhalts
Entwicklungswerkzeug zum Erstellen von Plugins
Dokumentation der Entwicklungsumgebung und Schnittstellen-Methoden
Beschreibung der Geogrid®-Fremdsystem Schnittstelle
Beschreibung der Dateistruktur des entpackten SDK

Abbildung 1: Inhalt der Plugin-Schnittstellen CD am Beispiel Geogrid-Schnittstelle (Interface) Version 5.1

2.3 Das Interface Control Document (ICD)

Die Datei **PlugIn_Vx.y_ICD.zip** (Vx.y steht für die aktuelle Version z.B. V5.1) enthält in komprimierter Form die Geogrid®-PlugIn Schnittstellenbeschreibung als HTML-Hilfe. Erstellen Sie einen Ordner für das ICD (z.B.:C:\Plugin-ICD) und extrahieren Sie den Inhalt dieser Datei dorthin. Nach dem Extrahieren enthält der Ordner „C:\Plugin-ICD“ die Verzeichnisse Docu und UML sowie die Datei **Geogrid_Plugin_ICD_Vxy.chm**. Durch einen Doppelklick auf [Geogrid_Plugin_ICD_V51.chm](#) öffnen Sie eine HTML-Hilfe, die alle Informationen aus dem Ordner Docu und dessen Unterordnern enthält. Hier finden Sie u.a.

- einen kurzen Überblick über die COM Technologie (introduction und COM), die allerdings nicht die Lektüre der einschlägigen Literatur ersetzt.
- Eine Liste der neusten Schnittstellenerweiterungen (What's new?)
- Beispiele, die zeigen wie ein Plugin erstellt wird (Examples)
- Einen Überblick über alle Schnittstellen (Plugin History)

Das ICD ist das wichtigste Hilfsmittel beim Erstellen eines Plugins, denn sie enthält die Beschreibung aller Schnittstellen-Methoden. Zur Suche nach bestimmten Methoden verwenden Sie am besten die Volltextsuche oder den Index auf der linken Seite des Hilfe-Fensters.

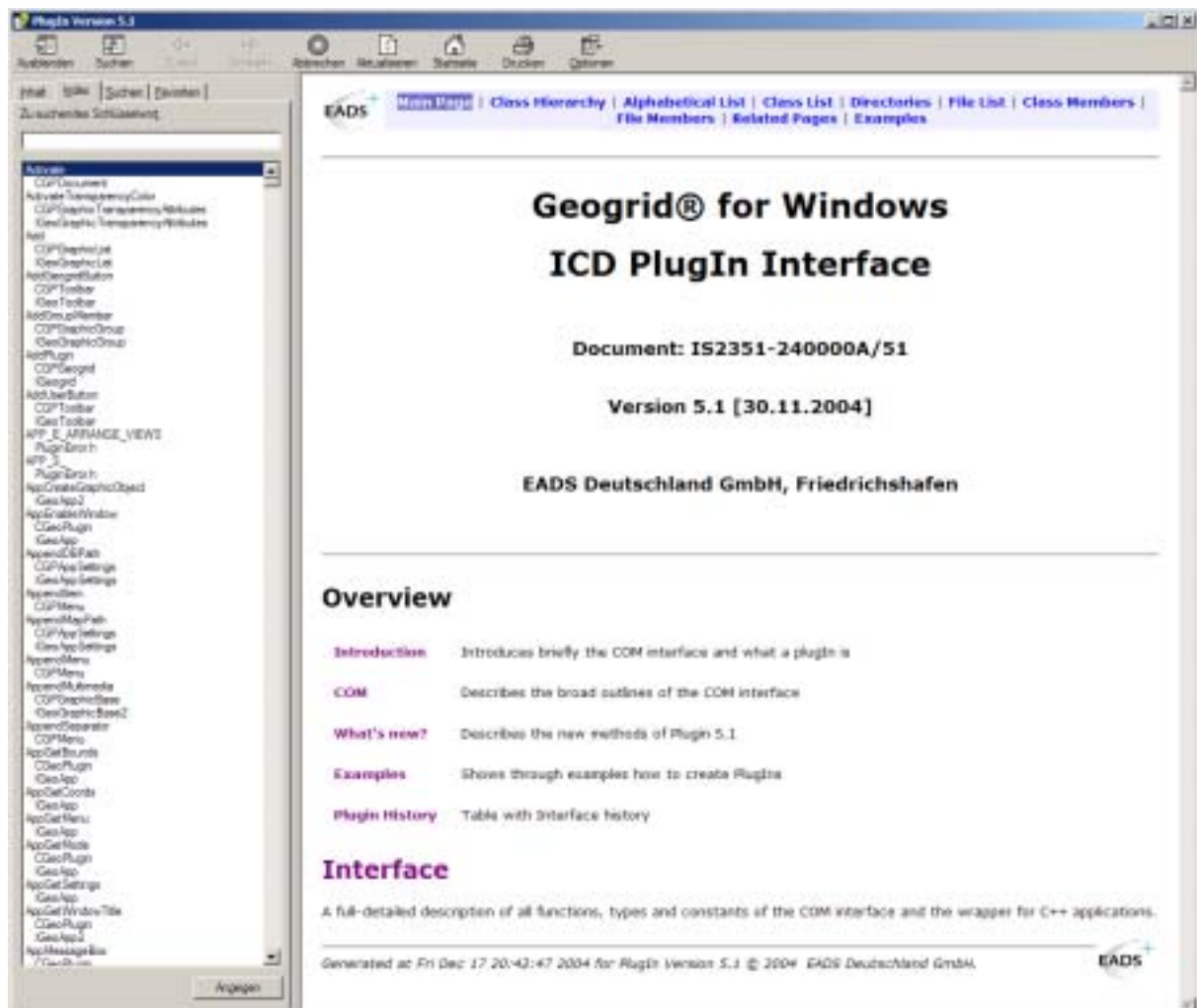


Abbildung 2: Startseite von Geogrid_Plugin_ICD_V51.chm

2.4 Das Software Development Kit (SDK)

Die Datei **PlugIn_Vx.y_SDK.zip** enthält in komprimierter Form die notwendigen Bibliotheken zum Erstellen eines Geogrid®-Plugins sowie Beispielprogramme. Erstellen Sie einen Ordner für das SDK (z.B.:C:\PluginSDK) und extrahieren Sie den Inhalt dieser Datei dorthin. Nach dem Extrahieren enthält der Ordner „C:\PluginSDK“ folgende Verzeichnisse

- Interface** Schnittstellendefinitionen als IDL-Dateien und den durch den Microsoft-MIDL Compiler erzeugten Bibliotheken.
- Samples** Beispiele geschrieben in Visual C++ inklusive Quellcode
- Tutorial** Beispiele, zu denen sich weiter unten in diesem Dokument eine Schritt für Schritt Anleitung findet
- Wrapper** Visual C++ Wrapper Class Library zur einfachen Implementierung von Geogrid®-Plugins

Wie die Dateien aus den Ordnern Interface und Wrapper verwendet werden, wird im Rahmen der Schritt für Schritt Anleitungen zur Erstellung eines Plugins ab Kapitel 4.1 erläutert.

2.5 Die Geogrid®-Fremd-System-Schnittstelle (FSS)

Die Datei **FSS_V2.01_ICD.zip** enthält in komprimierter Form die Beschreibung Geogrid®-Fremd-System-Schnittstelle als HTML Dokumentation. Sie benötigen einen Webbrowser um das ICD zu lesen. Erzeugen Sie einen Ordner (z.B.: C:\FSS-ICD) und extrahieren Sie den Inhalt dieser Datei dorthin. Die Startseite der Dokumentation wird mit "Default.html" aufgerufen.

Diese Schnittstelle ermöglicht die Kommunikation mit der Geogrid® Applikation.

Folgende Funktionen sind u.a. verfügbar:

- Anzeigen, Zoomen und Zentrieren von Karten
- Erzeugen, Ändern und Löschen von Graphiken und das Anzeigen auf der Karte
- Erzeugen, Ändern, Löschen und Wiederherstellen von Graphiken als Overlays
- Lesen von Karteninformationen
- Lesen von Höhendaten
- Anfordern von Koordinaten

Diese Funktionalitäten stellen einem fremden System geographische Informationen zur Verfügung und ermöglichen die interaktive Darstellung von Szenarien auf der Karte. Eine LAN/WAN Schnittstelle mit TCP/IP Protokoll ist ausreichend um ein fremdes System mit Geogrid® zu verbinden. Die FSS Client Server Architektur ermöglicht die Kommunikation über Socket und CORBA Schnittstellen.

Die Geogrid®-FSS Schnittstelle ist nicht Gegenstand dieser Beschreibung. Weitere Details entnehmen Sie bitte der Geogrid®-FSS Schnittstellenbeschreibung.

3 SYSTEMBESCHREIBUNG

Mit dem Plugin-SDK wird dem Anwendungs-Entwickler ein Werkzeug in die Hand gegeben, mit dem er eigene Applikationen in ein Geogrid®-Produkt integrieren kann oder in seiner Applikation auf Funktionalitäten von Geogrid®-Produkten zugreifen kann.

Die Plugin-Schnittstelle stellt Methoden bereit, um Funktionen der Geogrid®-Software benutzen zu können. Die Definition dieser Schnittstellen basiert auf dem Microsoft COM-Standard. Um Anwendungsentwicklern die Einarbeitung in COM zu ersparen, steht neben der Schnittstellen-Bibliothek auch ein MFC C++ Wrapper zur Verfügung, der als Klassenbibliothek verwendet werden kann. Diese Bibliothek kapselt die COM-Schnittstelle.

Es ist möglich Geogrid® um neue Funktionalitäten zu erweitern, z.B. indem ein neues Menü in das Standard-Menü eingefügt wird. Dies ist eine Anwendung für ein DLL-Plugin. Nachfolgend ist ein Beispiel für ein DLL-Plugin zu sehen. Dabei ist das Geogrid® Menü um das Menü „Skizzen Plugin“ ergänzt worden. Die dahinterliegende Funktionalität ermöglicht das Erstellen von Freihandzeichnungen.

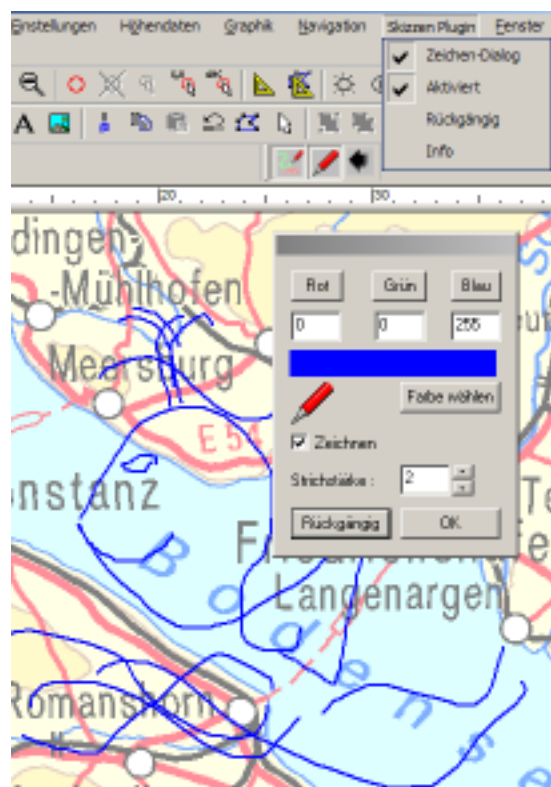


Abbildung 3: DLL-Plugin zum Einfügen von Freihandzeichnungen

Über die Plugin-Schnittstelle kann Geogrid® als Komponente in ein anderes System eingebunden werden. Geogrid® dient dann z.B. als Kartendarstellungsmodul, in dem Geo-Informationen graphisch dargestellt werden. Das Aussehen von Geogrid® kann dabei so verändert werden, das es nicht mehr als fremde Komponente erkennbar ist. Dies ist eine Anwendung für ein EXE-Plugin.

Ein Beispiel für die Verwendung eines EXE-Plugins ist die unten dargestellte GIS-Applikation. Hier wird Geogrid® zur Darstellung von Karten mit allen dazugehörigen Funktionalitäten (Kartenwechsel, Positionieren, Zoomen, Dimmen, etc.) sowie zum Zeichnen und Positionieren von Graphiken verwendet. Das Geogrid®-Menü ist komplett ausgeblendet. Trotzdem können über die Plugin-Schnittstelle Geogrid®-Funktionalitäten und -Dialoge aufgerufen werden.

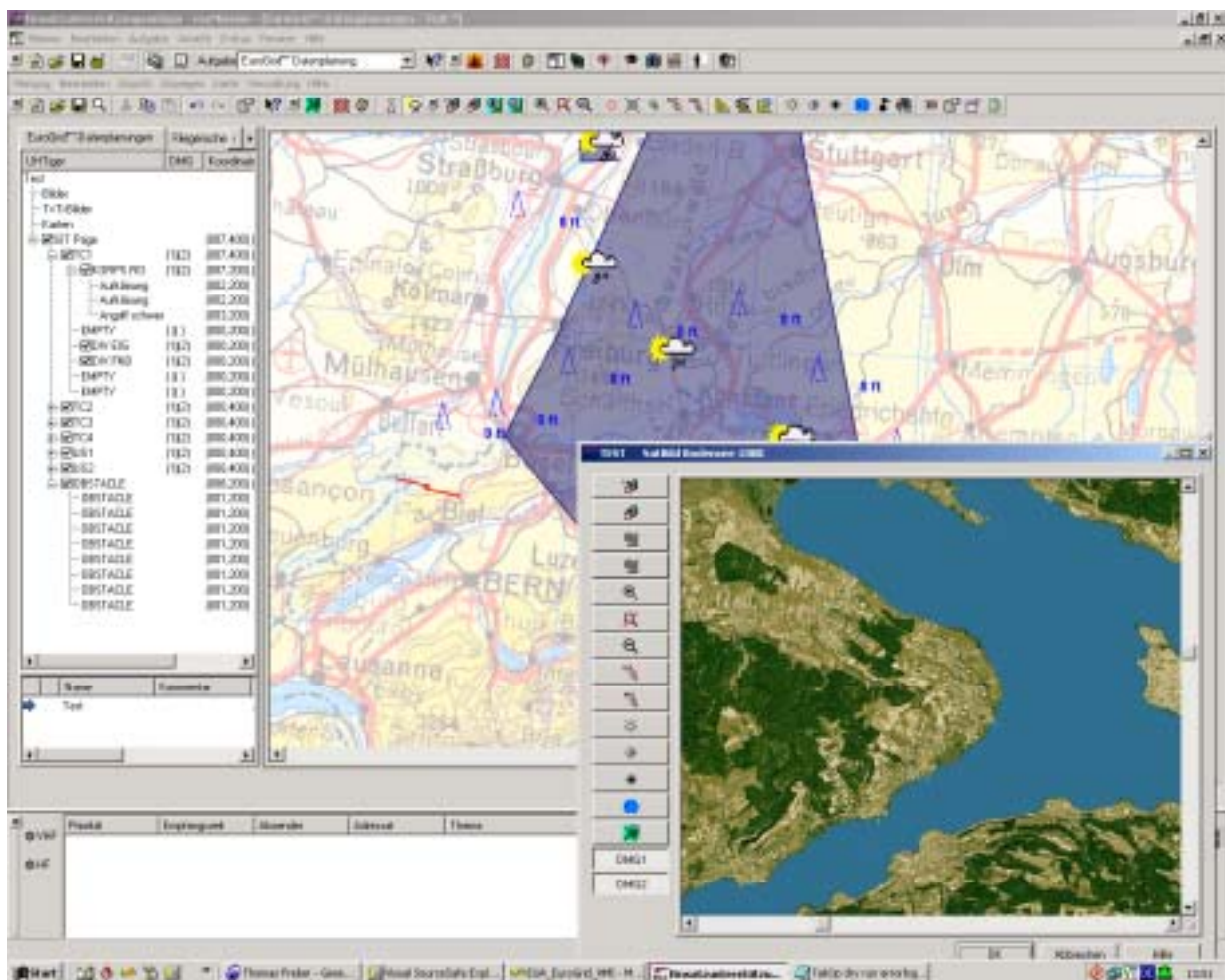


Abbildung 4: Anwendungs-Beispiel für ein EXE-Plugin mit eigener Benutzer-Oberfläche

3.1 Plugin

Ein Plugin ist eine Client-Komponente, welche über die Plugin-Schnittstelle auf Funktionen der Geogrid®-SW zugreift. Es können mehrere Plugins gleichzeitig mit Geogrid kommunizieren. Jedes Plugin muss die Schnittstelle IGeoPlugin implementieren. Sie dient zum Aufruf von Plugin-Methoden durch Geogrid®. Diese sind notwendig, um das Plugin über Aktionen in der Geogrid®-GUI zu informieren. Die zu implementierenden Methoden sind:

```
IGeoPlugin {
    STDMETHODIMP Initialize (IGeoApp *pGeoApp, GEOPLUGINHANDLE hPlugin);
    STDMETHODIMP Shutdown();

    STDMETHODIMP OnMenuEvent( IGeoMenuEvent *pMenuEvent,
                              GEOMENUEVENTRET *pRet);
    STDMETHODIMP OnMouseEvent( IGeoMouseEvent *pMouseEvent,
                              GEOMENUEVENTRET *pRet);
    STDMETHODIMP OnMapEvent ( IGeoMapEvent *pMapEvent,
                              GEOMENUEVENTRET *pRet);
    STDMETHODIMP OnWindowEvent(IGeoWindowEvent *pWindowEvent,
                              GEOMENUEVENTRET *pRet);
    STDMETHODIMP OnGraphicEvent ( IGeoGraphicEvent *pGraphicEvent,
                              GEOMENUEVENTRET *pRet);
    STDMETHODIMP OnSettingsEvent(IGeoSettingsEvent *pSettingsEvent,
                              GEOMENUEVENTRET *pRet);

    STDMETHODIMP GetPluginInfo (IGeoPluginInfo **ppPluginInfo);
    STDMETHODIMP GetLicenseKey (BSTR *pstrLicenseKey);
}
```

Die oben aufgeführten Methoden werden von dem mitgelieferten Wrapper implementiert. Zu jeder dieser Methoden existiert eine zugehörige virtuelle Methode. Die Methoden regeln die Kommunikation über die COM-Schnittstelle. Jede dieser Methoden ruft ihre zugehörige virtuelle Methode auf. Diese kann bei Bedarf vom Plugin-Entwickler überschrieben werden.

```
CGeoPlugin {
    // PlugIn interface methods to be implemented by the user
    virtual void Init() ;
    virtual void Destroy() ;

    virtual GEOMENUEVENTRET OnMenuEvent (IGeoMenuEvent *pMenuEvent);
    virtual GEOMENUEVENTRET OnMouseEvent (IGeoMouseEvent *pMouseEvent);
    virtual GEOMENUEVENTRET OnMapEvent (IGeoMapEvent *pMapEvent);
    virtual GEOMENUEVENTRET
        OnWindowEvent (IGeoWindowEvent *pWindowEvent);
    virtual GEOMENUEVENTRET
        OnGraphicEvent (IGeoGraphicEvent *pGraphicEvent);
    virtual GEOMENUEVENTRET
        OnSettingsEvent(IGeoSettingsEvent *pSettingsEvent);

    virtual CString GetLicenseKey();
    virtual BOOL GetPluginInfo ( CString &strPluginName,
                                CString &strPluginVendor,
                                CString &strPluginDescription,
                                long *lMajor, long *lMinor);
}
```

3.1.1 Die Verbindung zwischen Client und Server herstellen

Am Beginn steht der Aufruf der Methode *ColInitialize*. Sie initialisiert die COM-Bibliothek. Eine Applikation muss die COM-Bibliothek initialisieren bevor sie COM-Bibliotheksfunktionen aufrufen kann. Für DLL-Plugins ist kein gesonderter Aufruf dieser Methode notwendig, da die DLL in demselben Prozess wie Geogrid® läuft und dort bereits die Initialisierung durch einen Aufruf von *OleInitialize* stattgefunden hat. Diese Methode ruft ihrerseits die Methode *ColInitialize* auf. Bei einem EXE-Plugin, unter Verwendung der Wrapper-Bibliothek, findet die Initialisierung im Konstruktor der Wrapper Klasse *CGPGeogrid* statt.

Das EXE-Plugin benutzt die Methode *CLSIDFromProgID*, um an die CLSID der Geogrid®-Applikation zu gelangen. Die CLSID wird bei dem Aufruf der Methode *CoGetClassObject* benötigt

CoGetClassObject liefert den Zeiger zu einem Schnittstellenobjekt, das durch eine CLSID spezifiziert ist und mit *CreateInstance* von *IClassFactory* erzeugt wurden. *CoGetClassObject* lokalisiert den ausführbaren Code, der dafür erforderlich ist und lädt ihn wenn nötig dynamisch. Mit diesem Aufruf wird z.B. Geogrid® durch das EXE-Plugin gestartet, wenn Geogrid® noch nicht läuft. Diese Methode wird im Konstruktor der Wrapper-Klasse *CGPGeogrid* aufgerufen. Dieser wiederum wird in den mitgelieferten Beispielen in der *InitInstance()* – Methode des EXE-Plugins aufgerufen (siehe Kapitel 4.2.5).

Um diese einzelnen Schritte braucht sich ein Anwender des Wrappers keine Gedanken machen, da durch die Wrapper Klassen alle notwendigen Initialisierungen und das Erzeugen von COM-Objekten übernommen werden.

3.1.2 DLL-Plugin

Ein **DLL-Plugin** wird in dem Adressraum der Geogrid®-Software ausgeführt. Seine Lebensdauer ist auf die der Geogrid®-Software beschränkt. Da es direkt zur Geogrid®-Software dazugelinkt wird und in demselben Adressraum ausgeführt wird, erfolgt die Kommunikation über die Schnittstelle schneller, als bei einem EXE-Plugin. Bei einem EXE-Plugin wird ein Funktionsaufruf zunächst an ein Proxy-COM-Objekt geleitet. Dieses leitet den Aufruf an den Server weiter. Dort wird der Aufruf von einem Stub-Objekt entgegengenommen und schließlich an Geogrid® weitergeleitet. Die Antwort auf einen solchen Aufruf muss den gleichen Weg in umgekehrter Richtung zurücklegen. Damit Plugin-DLLs beim Start von Geogrid geladen werden können, müssen sie sich in dem Plugin-Verzeichnis, unterhalb des bin-Verzeichnisses der installierten Geogrid®-SW, befinden.

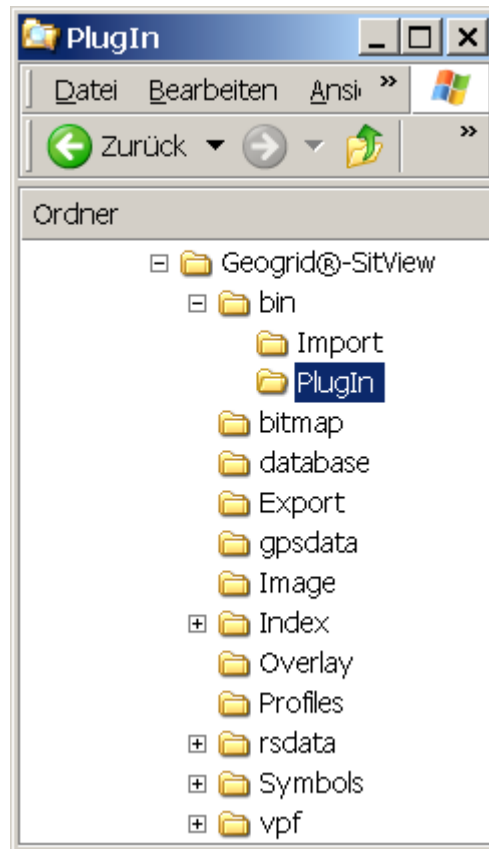


Abbildung 5: Speicherort der Plugin-DLL

Die Plugin-DLLs müssen alle in einem Verzeichnis stehen und können nicht verteilt auf dem Rechner liegen.

3.1.3 EXE-Plugin

Ein **EXE-Plugin** ist unabhängig von der Geogrid®-SW. Es kann zunächst ohne Geogrid® starten und erst wenn es erforderlich wird, Geogrid® aufrufen und die Plugin-Schnittstelle verwenden. Das Plugin und die Geogrid®-SW werden in zwei unterschiedlichen Prozessen ausgeführt. Das EXE-Plugin wird nicht beendet, wenn Geogrid® beendet wird. Da die Kommunikation über Prozessgrenzen hinweg erfolgt und dadurch der Kommunikationsaufwand größer ist, sind Schnittstellenaufrufe etwas zeitaufwendiger.

4 PLUGINS ERSTELLEN

4.1 Hello World DLL-Plugin

Im Folgenden werden die Schritte gezeigt, die notwendig sind, um ein minimales Plugin als DLL in einer Microsoft Visual C++ Umgebung zu erstellen. Minimal-Plugin bedeutet, es wird eine Verbindung zwischen dem Geogrid®-Programm und dem Plugin hergestellt und ein Dialog angezeigt, der signalisiert, dass eine Verbindung zustande gekommen ist. Zur Kapselung der COM-Schnittstelle wird ein MFC-Wrapper verwendet.

4.1.1 MFC-Anwendungsassistent

Im Menü *Datei* des Developer Studios klicken Sie auf *Neu*, um den MFC-Anwendungsassistenten auszuführen. Im Dialogfeld *Neu* klicken Sie auf das Register *Projekte* (1).

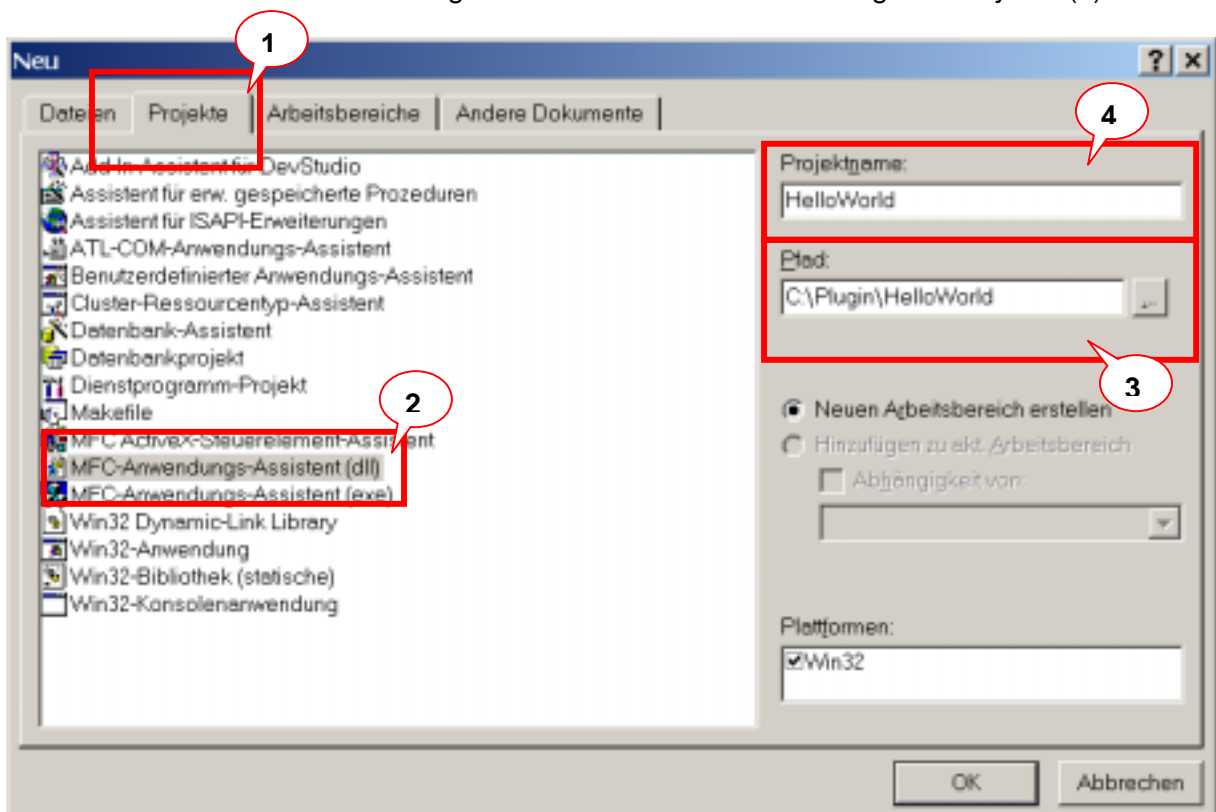


Abbildung 6: Projekt für Plugin-Dll anlegen

Wählen Sie die Option *MFC-Anwendungs-Assistent (dll)* (2) und geben Sie dann unter *Pfad* z.B. **C:\Plugin** ein (3). Unter *Projektname* tragen Sie **HelloWorld** ein (4). Anschließend klicken Sie auf *OK*.

In dem nachfolgend erscheinenden Dialog sind keine Eintragungen erforderlich. Klicken Sie hier auf *Fertigstellen*.

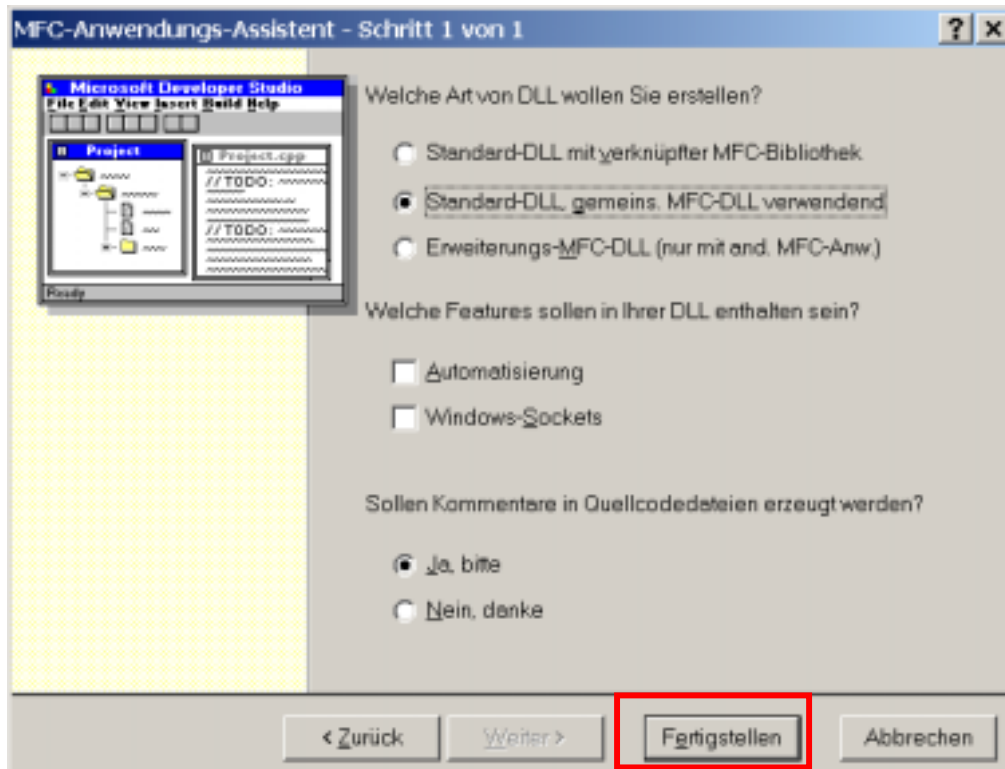


Abbildung 7: Anwendungsassistent

Bevor nun der Anwendungs-Assistent das neue Projekt erzeugt, wird der Dialog *Neue Projektinformationen* angezeigt:

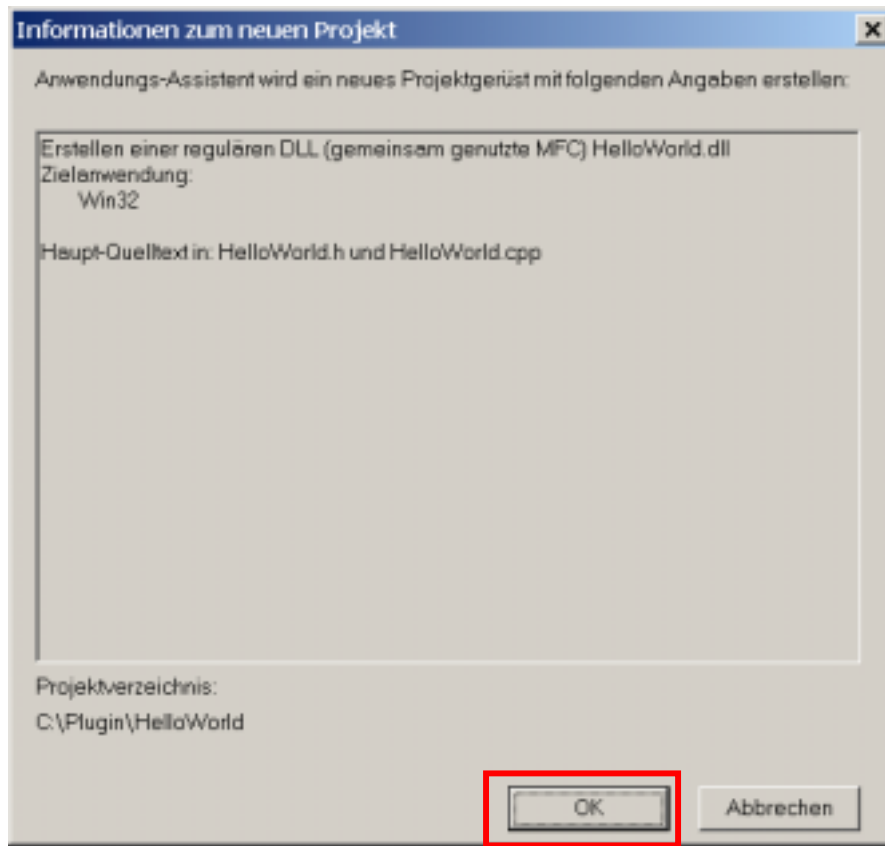


Abbildung 8: Informationen zum neuen Projekt

Durch Klicken auf die Schaltfläche *OK* legt der Anwendungsassistent die unter Pfad im ersten Dialog eingetragene Verzeichnisstruktur auf Ihrem Rechner an, sofern sie noch nicht vorhanden war.

4.1.2 Projekt-Einstellungen

Wenn Sie bisher noch nicht das Plugin-SDK auf Ihren Rechner extrahiert haben, ist jetzt der richtige Zeitpunkt. Extrahieren Sie die Datei **Plugin_Vx.y_SDK** auf Ihren Rechner; z.B. nach C:\PluginSDK. Dabei steht Vx.y für die Version der Plugin-Schnittstelle z.B. V5.1.

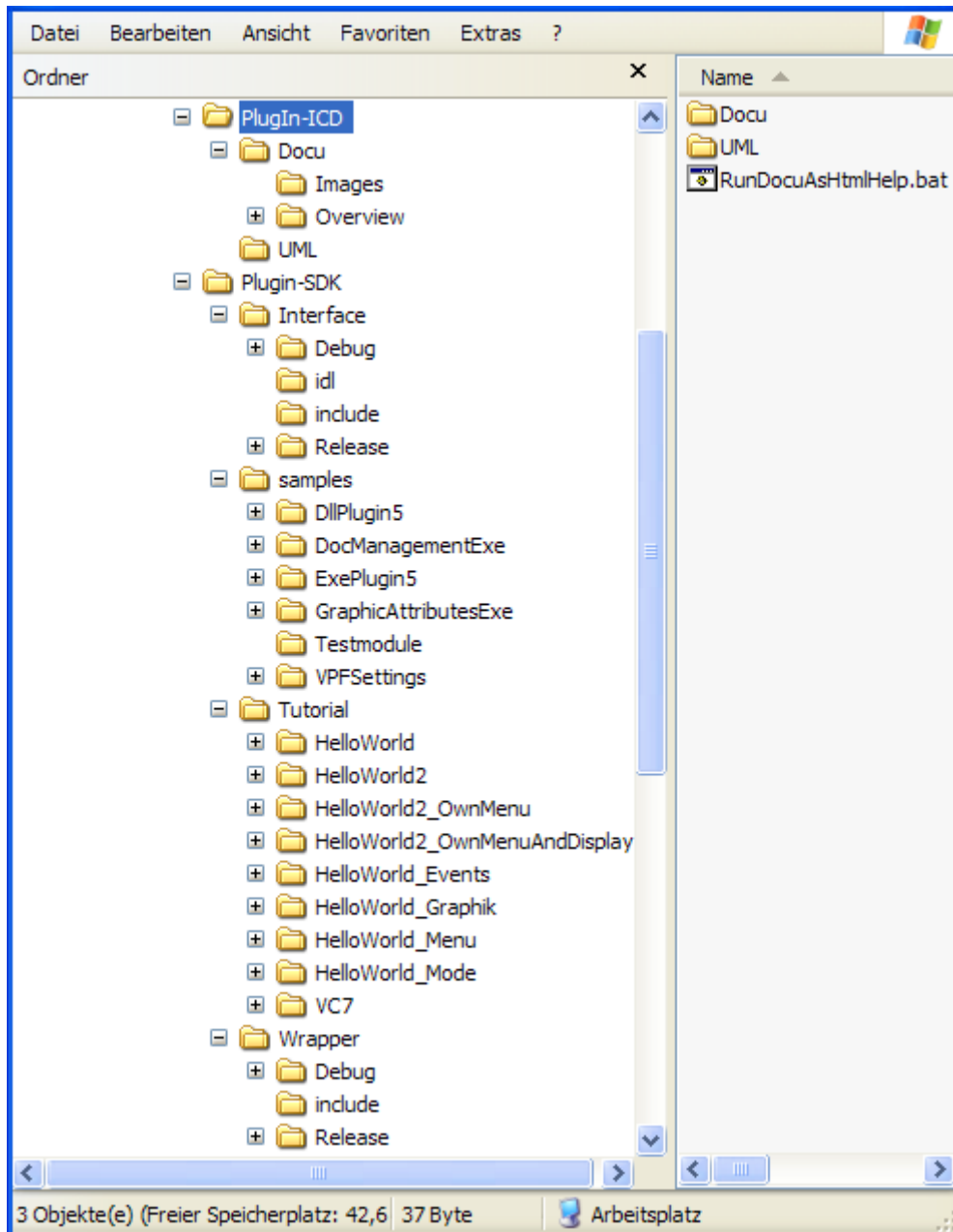


Abbildung 9: Struktur von Plugin-ICD und Plugin-SDK

Öffnen Sie über das Menü *Projekt/Einstellungen* den Dialog *Projekteinstellungen*. Wählen Sie links oben neben *Einstellungen für:* in der Auswahlliste *Alle Konfigurationen* (1). Anschließend klicken Sie auf das Register *C/C++* (2) und selektieren in der Auswahlliste *Kategorie* die Option *Präprozessor* (3). Unter *Zusätzliche Include-Verzeichnisse* tragen Sie die Pfade zu den Include-Dateien des Interface- und Wrapper-Ordners des Plugin-SDK ein (4).

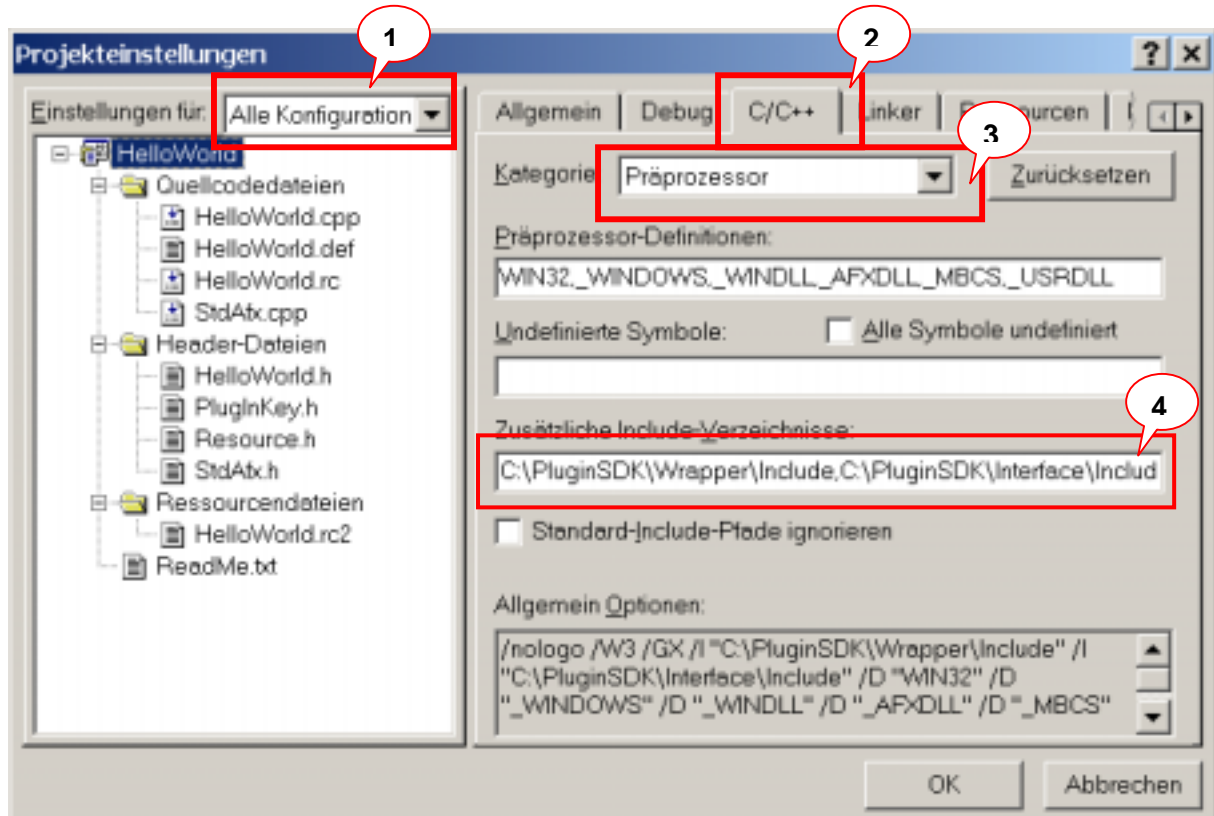


Abbildung 10: Einstellungen für Include-Pfade

Wählen Sie links oben in der Auswahlliste *Einstellungen für: Debug* aus **(1)**. Anschließend wechseln Sie auf das Register *Linker* **(2)** und stellen Sie in der Auswahlliste neben *Kategorie* die Option *Eingabe* ein **(3)**. Unter *Objekt-/Bibliothek-Module* tragen Sie die Bibliotheksnamen aus den Verzeichnissen **C:\PluginSDK\InterfaceDebug** und **C:\PluginSDK\Wrapper\Debug** ein **(4)**, z.B. WrapperD5.lib InterfaceD5.lib. Die Ziffern in den Librarynamen verweisen auf die Version des verwendeten Plugin-SDK. Unter *Zusätzliche Bibliothek-Verzeichnisse* tragen Sie die oben erwähnten Pfade zu den lib-Dateien des Interface- und Wrapper-Ordners ein **(5)**.

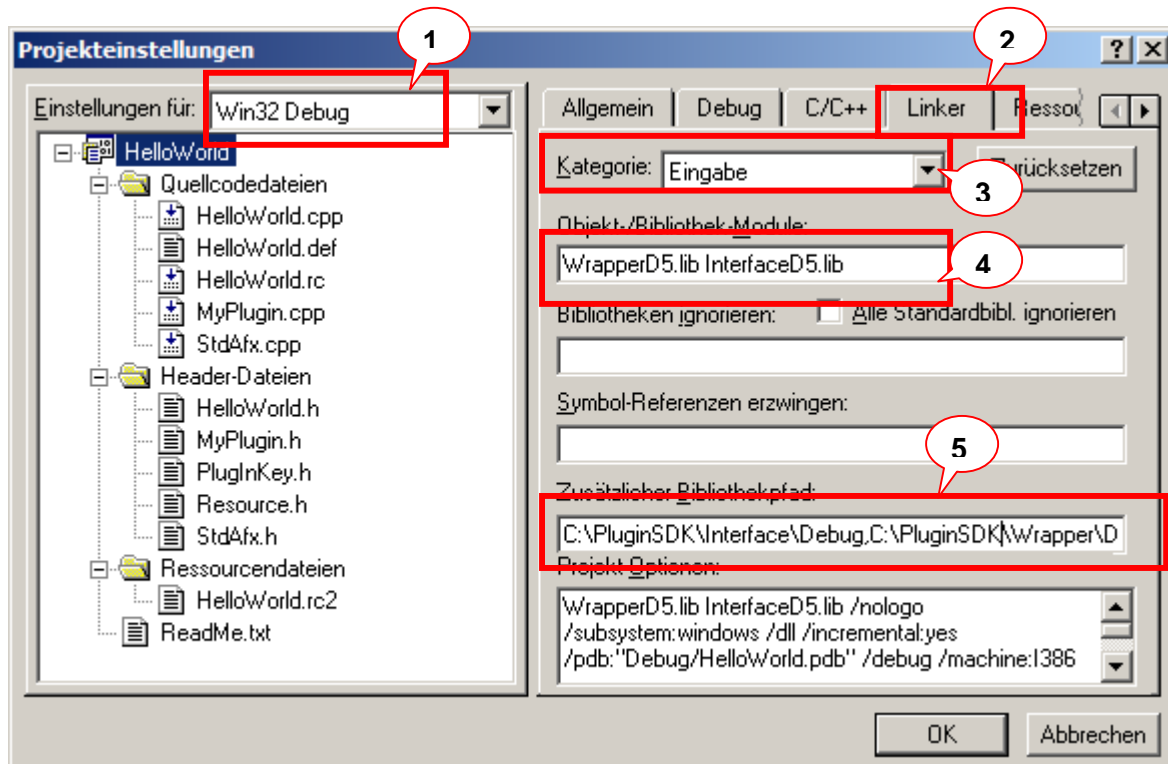


Abbildung 11: Einstellungen für Debug-Libraries

Wählen Sie links oben in der Auswahlliste *Einstellungen für: Release* aus **(1)**. Stellen Sie in der Auswahlliste neben *Kategorie* die Option *Eingabe* ein **(2)**. Unter *Objekt-/Bibliothek-Module* tragen Sie die Bibliotheksnamen aus den Verzeichnissen **C:\PluginSDK\Interface\Release** und **C:\PluginSDK\Wrapper\Release** ein **(3)**, z.B. WrapperR5.lib InterfaceR5.lib. Unter *Zusätzliche Bibliothek-Verzeichnisse* tragen Sie die oben erwähnten Pfade zu den lib-Dateien des Interface- und Wrapper-Ordners ein **(4)**.

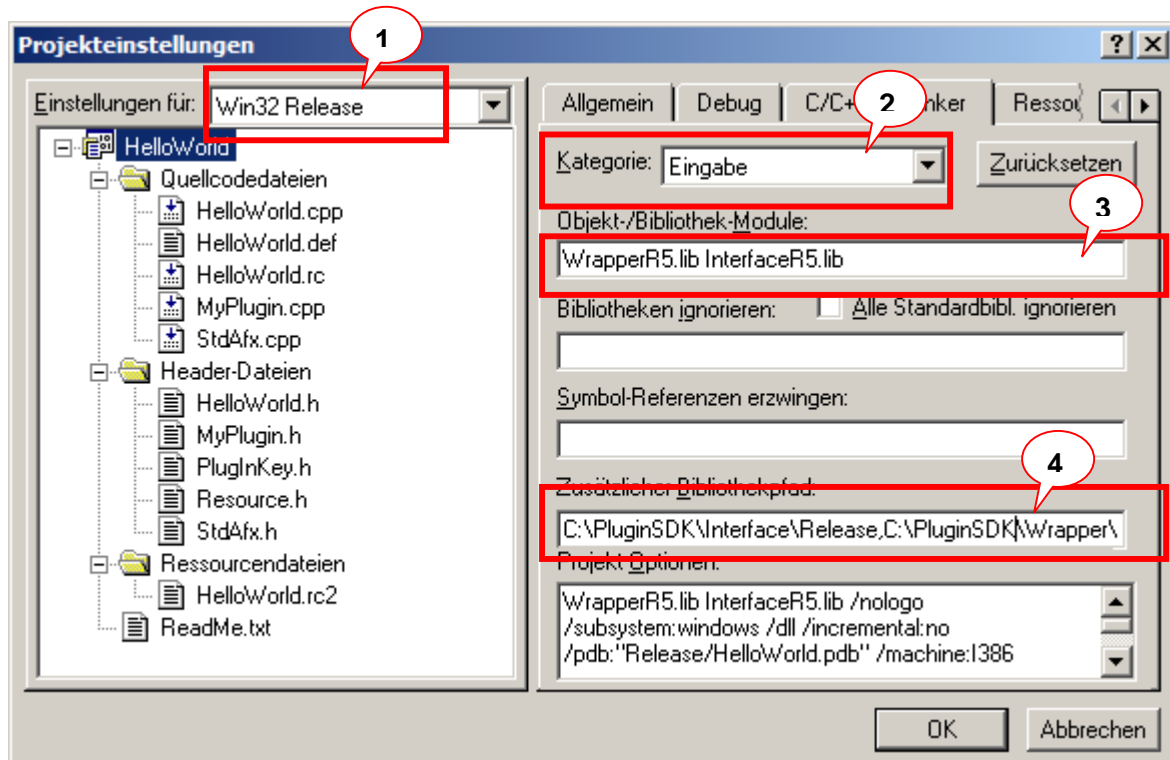


Abbildung 12: Einstellungen für Release-Libraries

4.1.3 Zu exportierende DLL Funktionen

Der Weg bis hierher war Ihnen sicher bereits bekannt. Nun kommen wir zu den für die Erstellung des Plugins wesentlichen Punkten.

Durch die DLL sind vier Funktionen zu implementieren und zu exportieren. D.h. die Implementierung wird dem Anwender durch Makro-Aufrufe abgenommen. Die Makros sind in der Datei PluginMacros.h im include-Verzeichnis der Schnittstellen enthalten.

1. STDAPI DllGetObject(

```
REFCLSID, //CLSID for the class object
REFIID, // ID der Schnittstelle
LPVOID * // Adresse des Schnittstellenzeigers, der zur ID passt.
);
```

DllGetObject wird aus der CoCreateInstance und der CoGetObject Funktion durch CoLoadLibrary aufgerufen, um einen Zugriff auf die Klassenfabrik des COM-Servers zu erhalten.

DllGetObject wird im Parameter *rclsid* die CLSID CLSID_GeoPlugin übergeben. Als zweiter Parameter wird **IID_IClassFactory** übergeben. Das bedeutet der Rückgabewert ist ein Schnittstellenzeiger auf eine Klassenfabrik. Die Implementierung der Klassenfabrik wird Ihnen ebenfalls durch ein Makro (IMPLEMENT_GEOPLUGIN_CLASS) abgenommen.

2. STDAPI DllCanUnloadNow();

Diese Funktion stellt fest, ob die DLL, in der sie implementiert ist, noch in Gebrauch ist. Wenn nicht, kann die DLL sicher aus dem Speicher entfernt werden.

Wenn eine DLL, die durch einen Aufruf von **CoGetObject** geladen wurde, die Funktion **DllCanUnloadNow** nicht exportiert, wird diese DLL nicht eher aus dem Speicher entfernt, bis die Applikation die Funktion CoUninitialize aufruft, um die COM Bibliotheken zu entfernen.

3. STDAPI DllRegisterServer(void);

DllRegisterServer enthält die notwendigen Aufrufe für die Einträge des Plugins in der Registry.

4. STDAPI DllUnregisterServer(void);

DllUnregisterServer entfernt die Registry-Einträge, die mit DllRegisterServer vorgenommen wurden.

Für den Export dieser Funktionen nehmen Sie in der Datei *HelloWorld.def* folgende Einträge vor:

```
DllGetObject PRIVATE
DllCanUnloadNow PRIVATE
DllRegisterServer PRIVATE
DllUnregisterServer PRIVATE
```

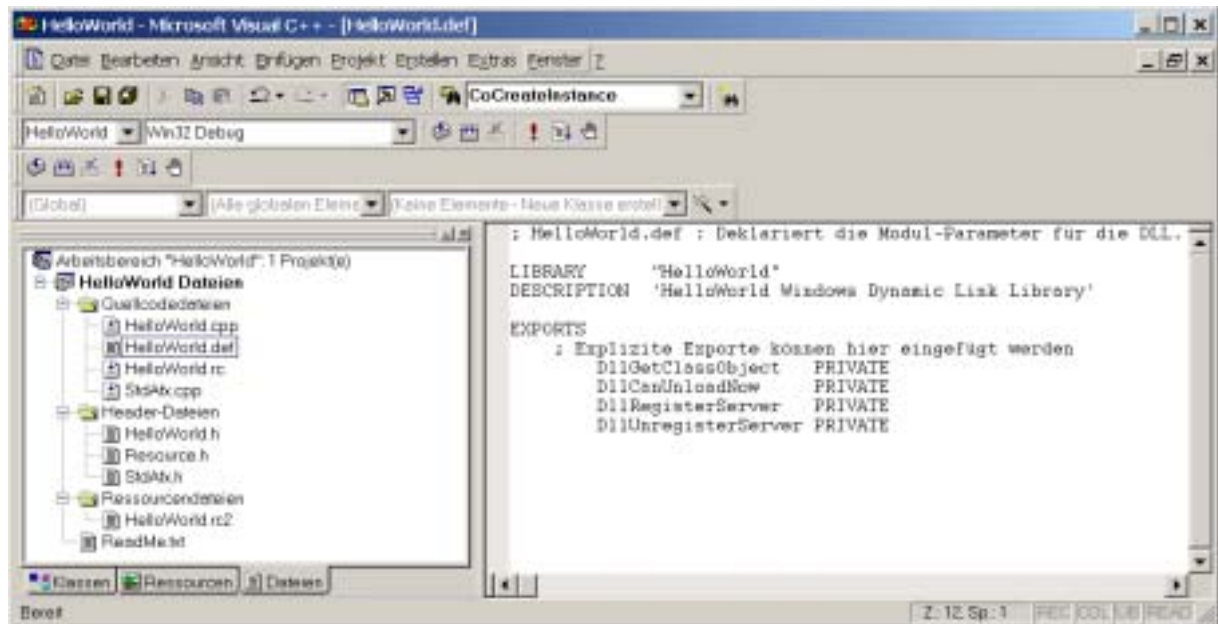


Abbildung 13: Def-Datei für Exportfunktionen

4.1.4 Lizenz-Schlüssel

Für die Nutzung des Plugins ist ein Lizenz-Schlüssel erforderlich. Sollten Sie noch nicht in Besitz eines solchen Schlüssels sein, können Sie ihn beim Hersteller der Geogrid®-SW und der Plugin-Schnittstelle beziehen.

Der Schlüssel ist in einer Include-Datei enthalten, die Sie Ihrem Projekt hinzufügen. In unserem Beispiel heißt Sie PlugInKey.h.

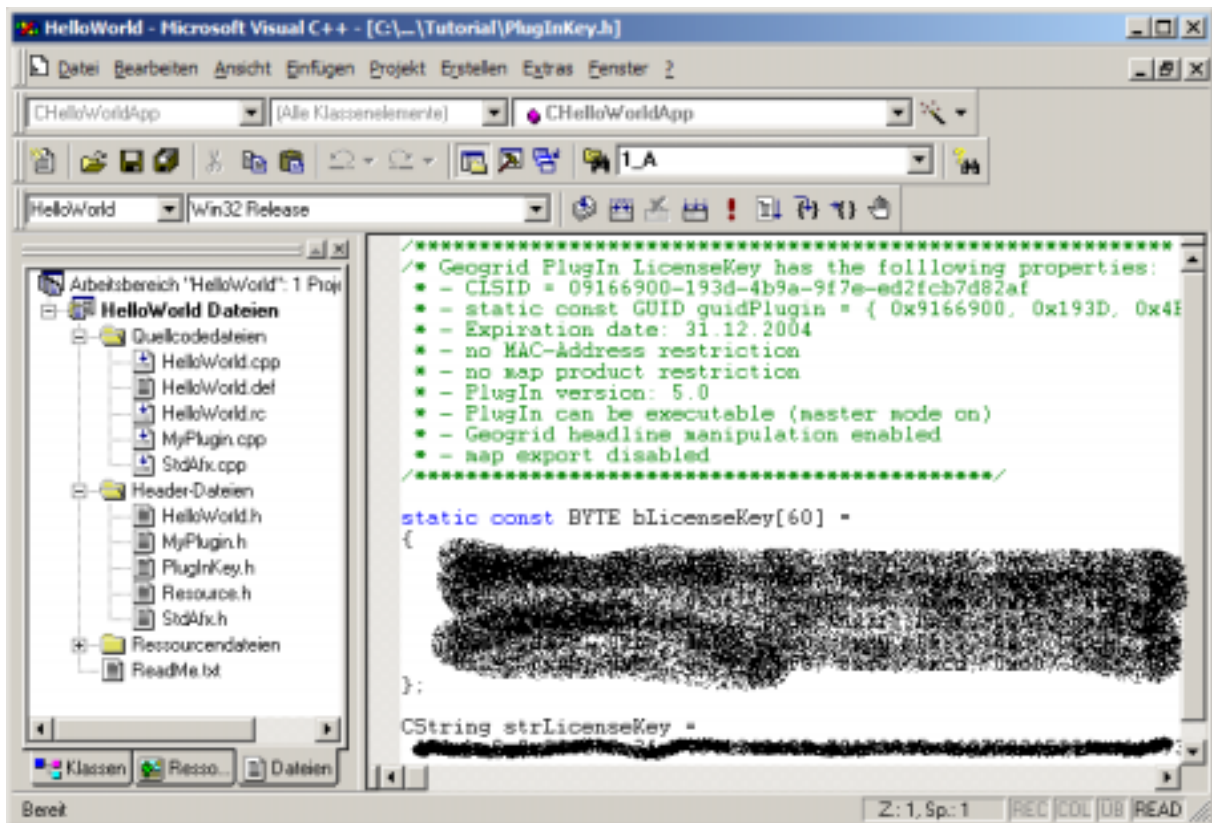


Abbildung 14: Lizenz-Datei

4.1.5 Ergänzungen in HelloWorld.cpp

Includieren Sie die Datei Wrapper.h. Sie enthält die Definitionen der Wrapperklassen und – Methoden. Fügen Sie die GUID aus der Datei PlugInKey.h hinzu.

Nun werden die bereits unter Punkt 3. erwähnten Makros eingefügt. Zunächst das Makro für die Registrierung der Plugin-DLL. Die Registry-Einträge werden von Geogrid® dazu verwendet um über den Plugin-Namen an dessen GUID zu gelangen. Dem Makro wird neben der CLSID des Plugins (CLSID_GeoPlugin) der Name der DLL übergeben. Die CLSID kann ebenfalls der Datei PlugInKey.h entnommen werden. Mit dem Makro *IMPLEMENT_PLUGIN_DLL_ENTRY_POINTS* werden die Exportfunktionen implementiert. *IMPLEMENT_PLUGIN_MODULE_ROUTINES* implementiert die Standard COM Funktionen:

```
ModuleAddRef(void)
ModuleRelease(void)
ModuleIsStopping(void)
ModuleIsIdle(void)
```

Beispiel:

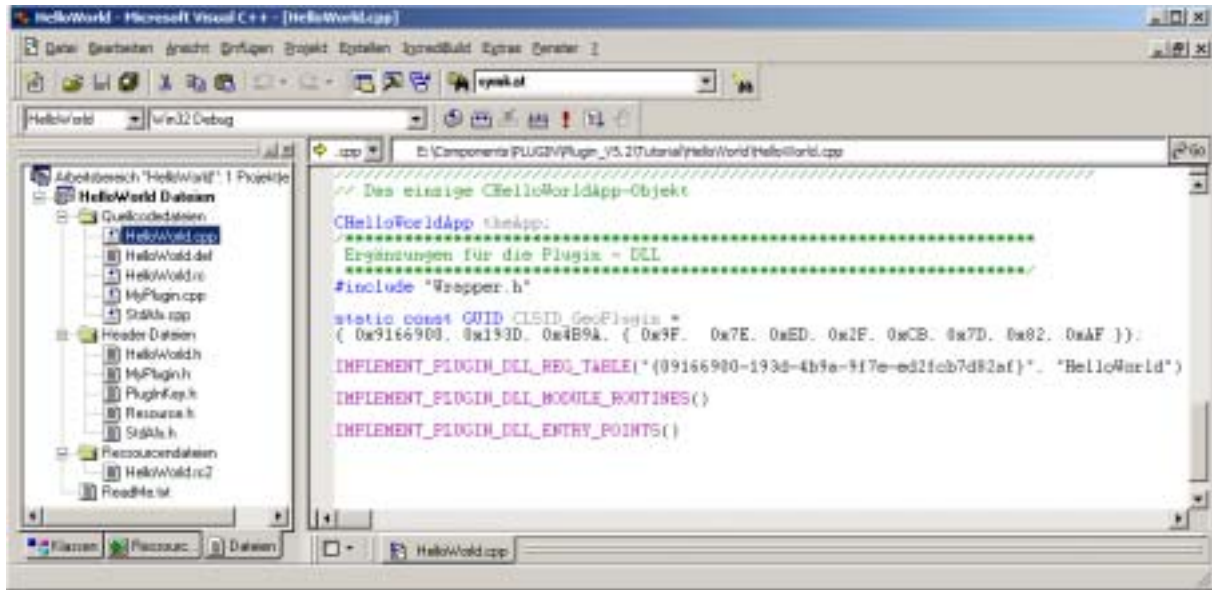


Abbildung 15: Einbinden der Class ID

4.1.6 Implementierung der Schnittstelle IGeoPlugin

CMyPlugin wird von der Klasse CGeoPlugin, welche die Schnittstelle IGeoPlugin implementiert, abgeleitet. Die Klasse CGeoPlugin enthält virtuelle Methoden, die überschrieben werden können. In diesem Beispiel beschränken wir uns auf Init(), Destroy() sowie GetLicenseKey().

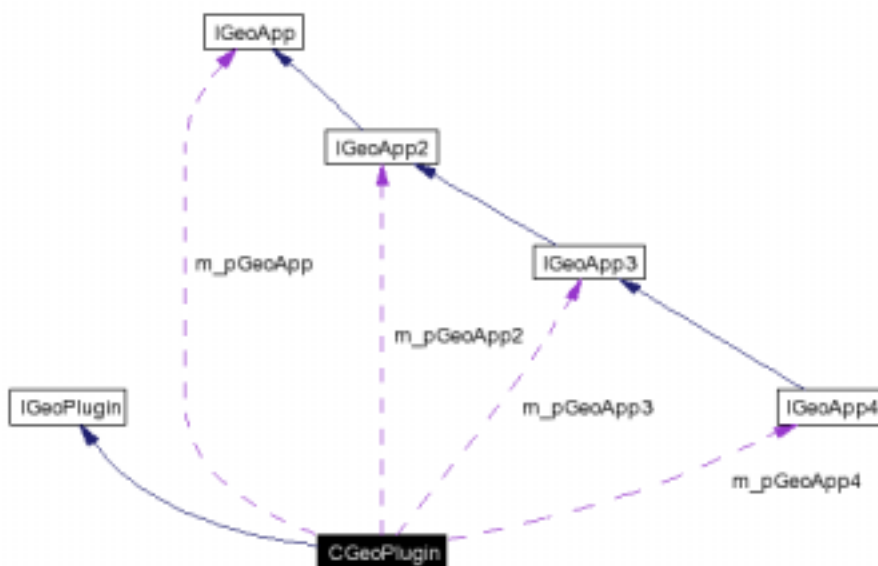


Abbildung 16 Ableitungshierarchie für CGeoPlugin

MyPlugin.h:

```
// specification of the PlugIn class
class CMyPlugin : public CGeoPlugin
{
public:
    void Init();
    void Destroy();

    CString GetLicenseKey();
};
```

MyPlugin.cpp:

```
// Implementation of the PlugIn class
#include "stdafx.h"

#include "Interface.h"
#include "Wrapper.h"

#include "MyPlugin.h"
#include "PlugInKey.h"

IMPLEMENT_GEOPLUGIN_CLASS(CMyPlugin)

void CMyPlugin::Init()
{
    AppMessageBox("Hello World!", "MyPlugin");
}

void CMyPlugin::Destroy()
{
    // Schreibe die aktuelle Zeit in das Geogrid INI file
    CTime curTime = CTime::GetCurrentTime();
    ProfileSetString ("MyPlugin", "ShutdownTime",
        curTime.Format ("%d.%m.%Y %H:%M:%S"));

    AfxMessageBox( "Good bye Plugin" );
}

CString CMyPlugin::GetLicenseKey()
{
    // Returns the license key for authorization of the plugin.
    // This method is called once before the plugin is
    // initialised.
    // 'strLicenseKey' is the license key defined in
    // 'Geogrid_PlugIn_Licens_Key.h'.
    // This include file has to be requested from EADS
    // Deutschland GmbH, Friedrichshafen.

    return strLicenseKey;
}
```

Die *Init()* Methode dient, wie der Name schon sagt, der Initialisierung des Plugins. Sie ist die erste Methode, die von Geogrid aufgerufen wird, wenn die Anmeldung des Plugins erfolgreich war. D.h. der durch *GetLicenseKey* an Geogrid® übergebene Lizenz Schlüssel ist gültig. Hier ist es z.B. sinnvoll Plugin-Menüs anzulegen oder Member-Variablen zu initialisieren.

Init enthält nur eine Methode. Sie ist in der Schnittstelle *IGeoApp* definiert und wird durch *CGeoPlugin* implementiert. Da *CMyPlugin* von *CGeoPlugin* abgeleitet ist können wir sie einfach aufrufen.

```
AppMessageBox("Hello World!", "MyPlugin");
```

Diese Methode zeigt eine einfache Message Box in der Mitte des Geogrid® Applikations-Rahmens an. Als Parameter werden zwei *CString* Werte übergeben. Der Erste enthält die Nachricht, die angezeigt werden soll, der Zweite den Titel der Dialogbox.

Geogrid® ruft *Destroy* auf, wenn es beendet wird. Damit können vom Plugin noch Aufräumarbeiten geleistet werden. In unserem Beispiel verwenden wir neben einem nochmaligen Aufruf der Methode *AppMessageBox* die Methode *ProfileSetString*. Sie wird dazu verwendet um Einträge im Ini-File der Geogrid® Applikation vorzunehmen. Die drei übergebenen Parameter haben den Typ *CString*. Sie definieren zuerst die Sektion im INI-File, dann den Schlüssel und schließlich den Wert:

```
[MyPlugin]  
ShutdownTime=31.08.2004 23:47:20
```

Wir haben sie benutzt, um die Zeit, wann Geogrid® zuletzt beendet wurde zu dokumentieren. Zum Ermitteln der Zeit ist die MFC Klasse *CTime* verwendet worden.

4.1.7 Das Plugin testen

Um das Plugin in Verbindung mit einer Geogrid® Applikation testen zu können, öffnen Sie über das Menü *Projekt/Einstellungen* den Dialog *Projekteinstellungen*. Wählen Sie links oben neben *Einstellungen für:* in der Auswahlliste *Alle Konfigurationen* (1). Anschließend klicken Sie auf das Register *Debug* (2) und selektieren in der Auswahlliste *Kategorie* die Option *Allgemein* (3). Unter *Ausführbare Programme für Debug-Sitzung:* tragen Sie den Pfad zu der exe-Datei der Geogrid® Applikation ein mit der Sie Ihr Plugin entwickeln möchten (4).

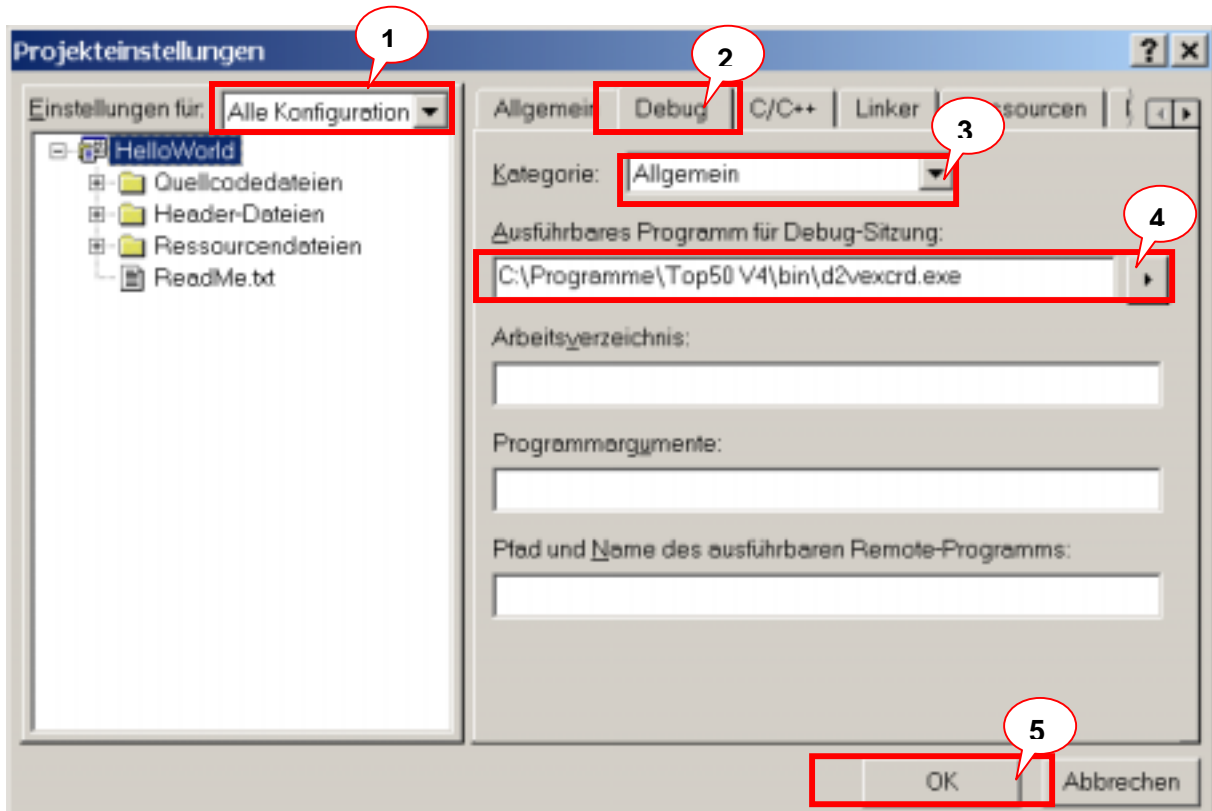


Abbildung 17: Einstellungen für eine Debug-Sitzung mit einem Geogrid® Viewer

Beenden Sie diesen Dialog mit **OK** (5). Anschließend öffnen Sie die Ini-Datei Ihrer Geogrid® Applikation. Sie befindet sich an der gleichen Stelle wie die exe-Datei, im bin-Ordner Ihres Installationsverzeichnisses. Dort tragen Sie unter der Init-Sektion den Pfad zu Ihrer Plugin-DLL ein, z. B.:

```
[INIT]  
PluginPath = C:\Plugin\HelloWorld\Debug
```

Wenn Sie jetzt eine Debug-Sitzung ausführen, wird automatisch Geogrid® gestartet und die Plugin DLL geladen. Dadurch muss die erzeugte DLL nicht nach jedem neuen Erstellen in das Plugin-Verzeichnis unterhalb des bin-Verzeichnisses der installierten Geogrid®-SW kopiert werden (siehe Kap. 3.1.2). Dieses Vorgehen darf nur für die Entwicklung eines Plugins verwendet werden. Im Einsatz muss sich die Plugin-DLL in dem oben beschriebenen Verzeichnis befinden (siehe Abbildung 5: Speicherort der Plugin-DLL in Kapitel 3.1.2)!

4.1.8 Zusammenfassung

In Abbildung 18 sind die Abläufe beim Start von Geogrid® mit einer PluginDLL als Sequenzdiagramm dargestellt. Beim Start sucht Geogrid® im Plugin-Verzeichnis nach DLLs. Die vorhandenen DLLs werden geladen und in die Registrierungsdatenbank (Registry) eingetragen. Über *CoCreateInstance* wird die Methode *DllGetClassObject* (siehe Kapitel 4.1.3) aufgerufen. Diese stellt eine Verbindung zu der Klassenfabrik her, die eine Instanz von *CGeoPlugin* erzeugt. Ein Zeiger auf diese Instanz wird an Geogrid® zurückgeliefert. Über diesen Zeiger wird der *LicenseKey* abgefragt und bei gültiger Lizenz die Initialisierungsmethode des Plugins aufgerufen.

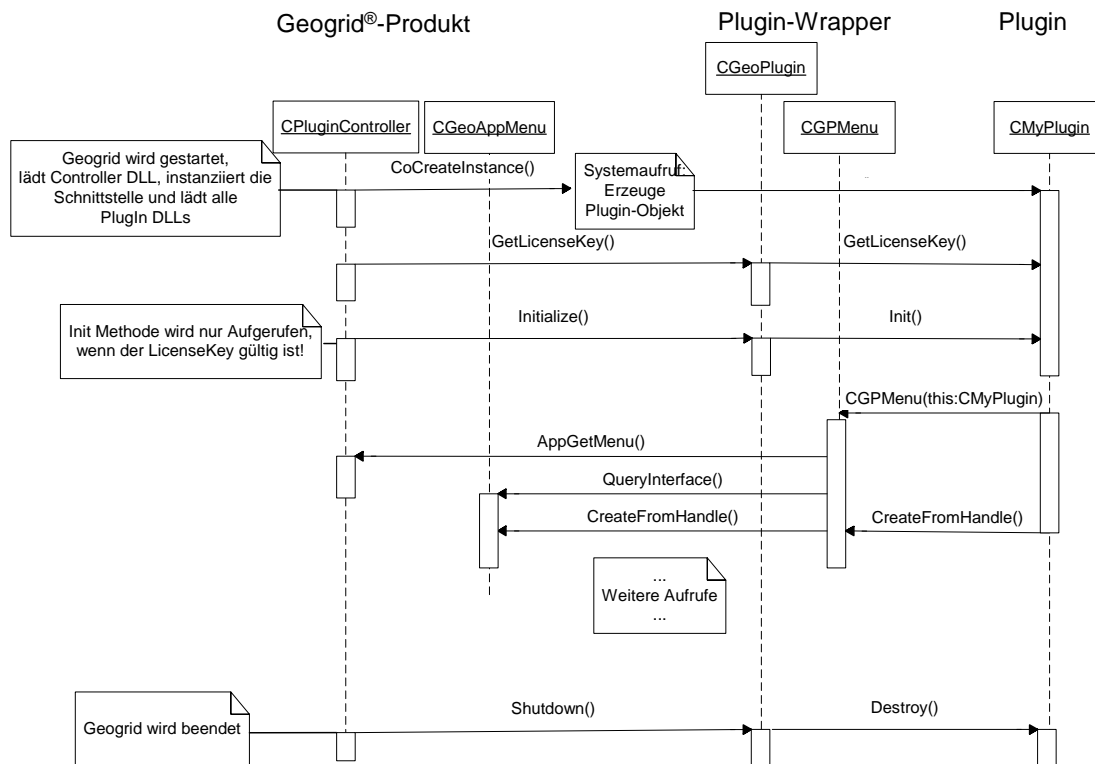


Abbildung 18: Sequenzdiagramm zur Kommunikation zwischen Plugin-DLL und Geogrid®

Abbildung 19, stellt die Ableitungs-Hierarchie für die Klasse CMyPlugin dar.

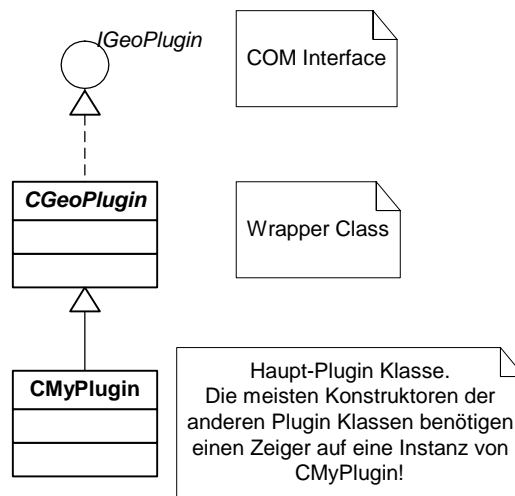


Abbildung 19: Klassendiagramm zu CGeoPlugin

In Abbildung 20 wird die Kommunikation zwischen der Plugin-DLL und Geogrid® in einem Blockdiagramm dargestellt. Die Schnittstellen werden durch eine Linie mit einem Kreis (—○) symbolisiert. Die allen COM-Objekten gemeinsame Schnittstelle IUnknown wird üblicherweise oberhalb der COM-Komponenten eingezeichnet. Die Komponenten sind als Rechtecke mit abgerundeten Ecken wiedergegeben. Sowohl über die Plugin-DLL (hier als Client.dll bezeichnet) als auch über die Geogrid® Applikation kann ein Datenaustausch erfolgen.

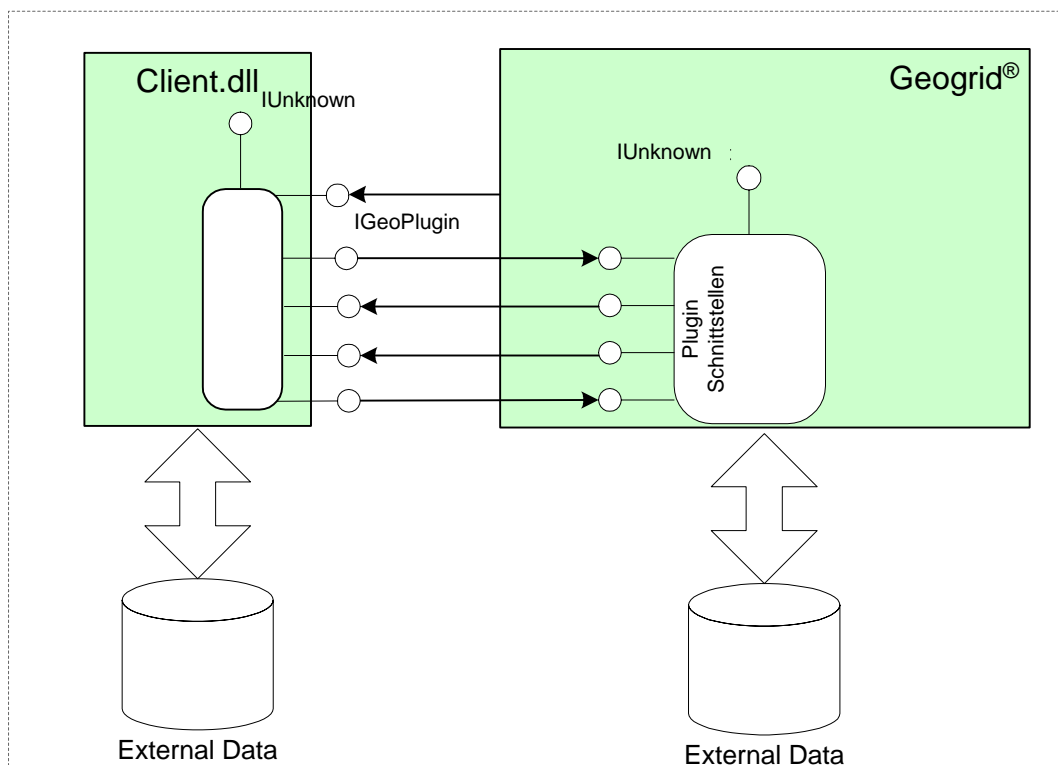


Abbildung 20: Kommunikation Zwischen DLL-Plugin und Geogrid®

4.2 Hello World EXE-Plugin

Im folgenden werden die Schritte gezeigt, die notwendig sind, um ein Minimal-Plugin als EXE in einer Microsoft Visual C++ Umgebung zu erstellen.

4.2.1 MFC-Anwendungsassistent

Im Menü *Datei* des Developer Studios klicken Sie auf *Neu*, um den MFC-Anwendungsassistenten auszuführen. Im Dialogfeld *Neu* klicken Sie auf das Register *Projekte* (1).

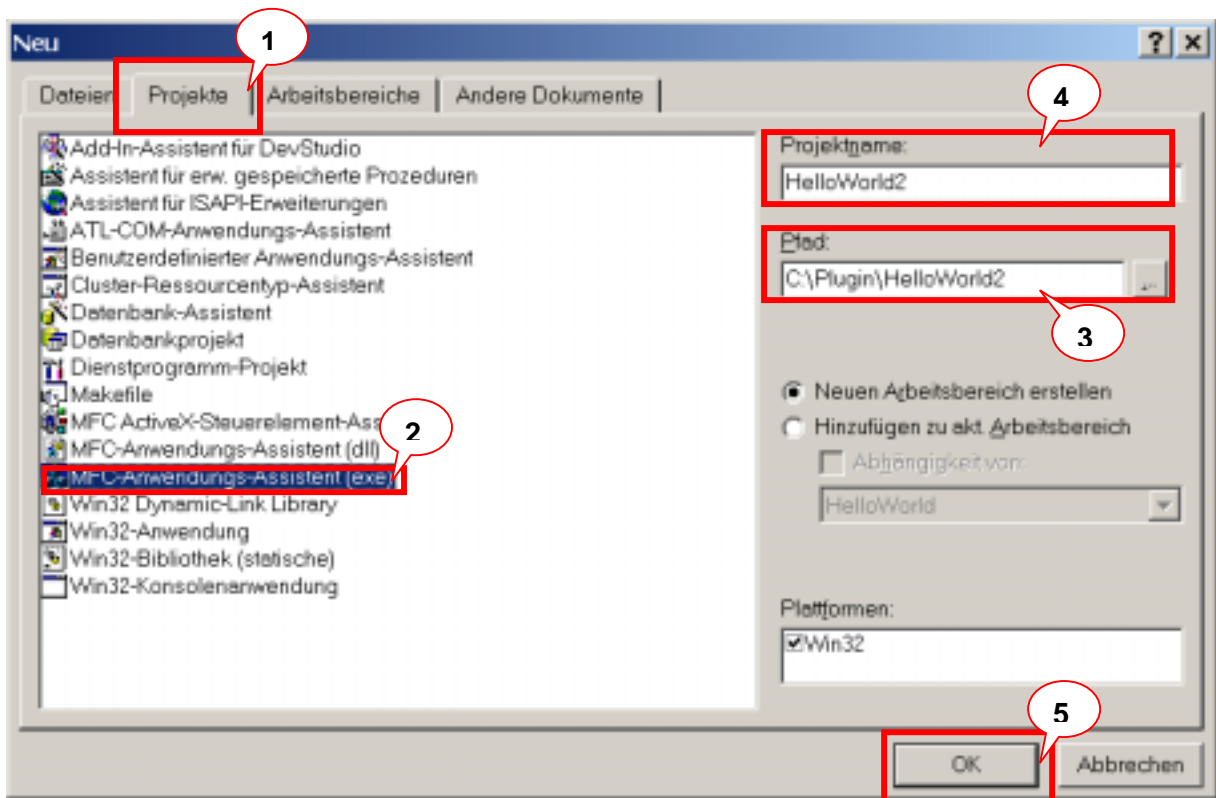


Abbildung 21: Exe-Plugin Projekt anlegen

Wählen Sie die Option *MFC-Anwendungs-Assistent (exe)* (2) und geben Sie dann unter *Pfad* **C:\Plugin** ein (3). Unter *Projektname* tragen Sie HelloWorld2 ein (4). Anschließend klicken Sie auf **OK** (5).

In dem nachfolgend erscheinenden Dialog wählen Sie die Option *Einzelnes Dokument(SDI)* aus. Klicken Sie hier auf *Fertigstellen*.

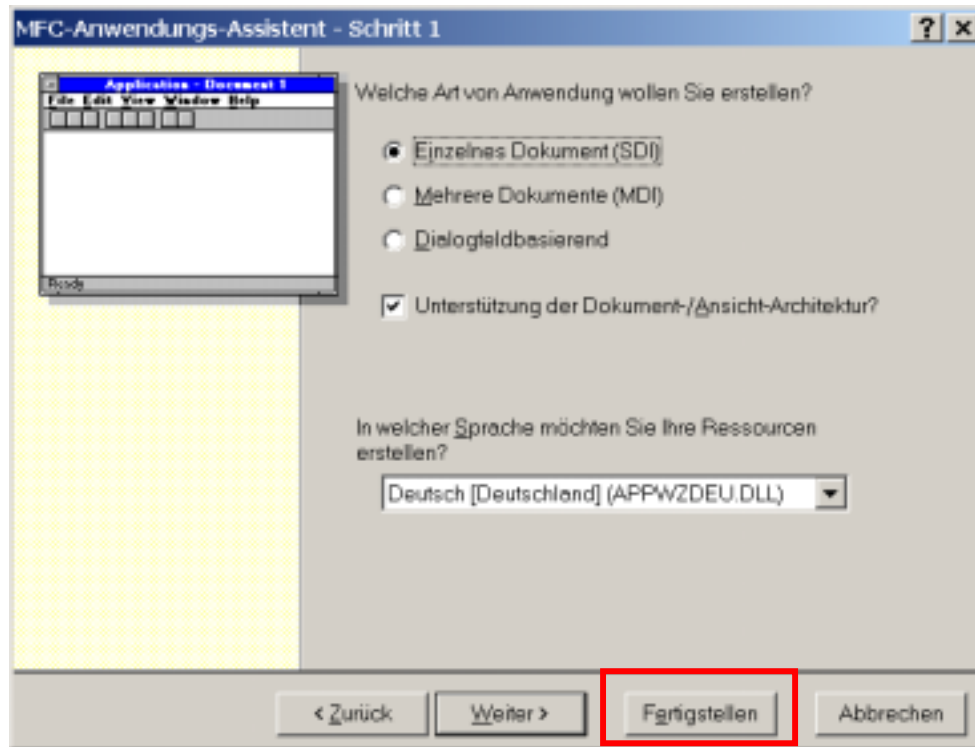


Abbildung 22: Anwendungsassistent

Bevor nun der Anwendungs-Assistent das neue Projekt erzeugt, wird der Dialog *Neue Projektinformationen* angezeigt:

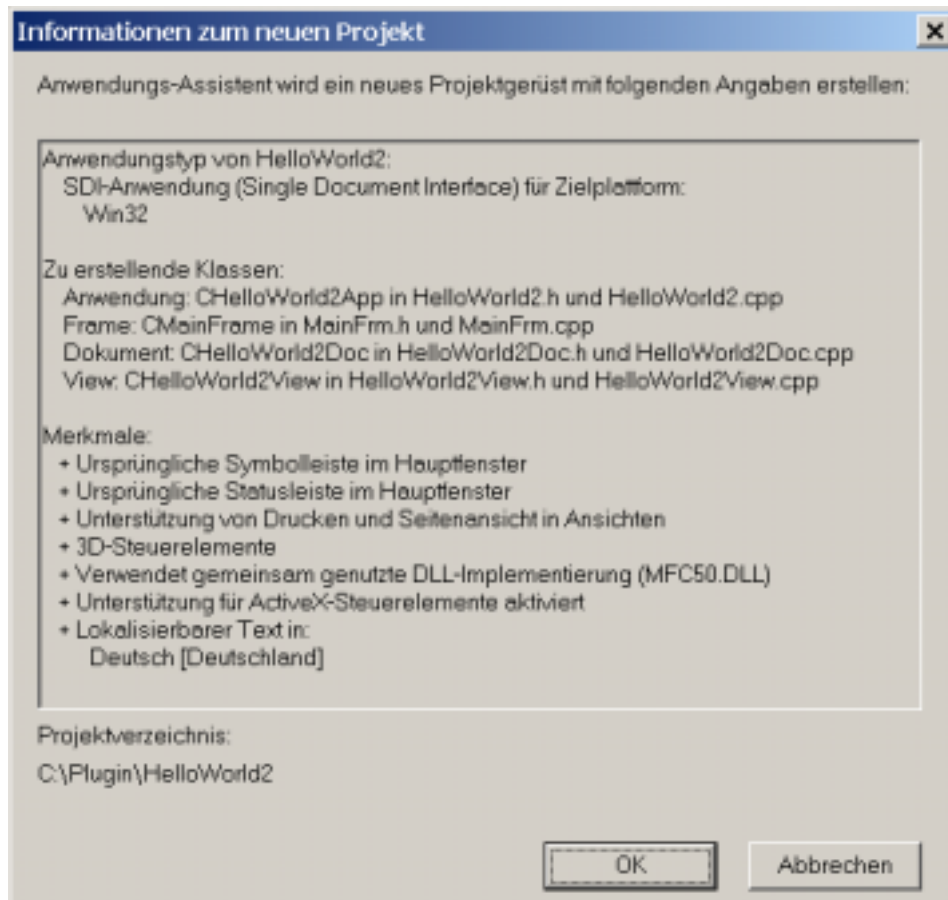


Abbildung 23: Informationen zum neuen Projekt

Durch Klicken auf die Schaltfläche *OK* legt der Anwendungsassistent die unter Pfad im ersten Dialog eingetragene Verzeichnisstruktur auf Ihrem Rechner an, sofern sie noch nicht vorhanden war.

4.2.2 Projekt-Einstellungen

Wenn Sie bisher noch nicht das Plugin-SDK auf Ihren Rechner extrahiert haben, ist jetzt der richtige Zeitpunkt. Extrahieren Sie die Datei **PlugIn_Vx.y_SDK** auf Ihren Rechner; z.B. nach C:\PluginSDK.

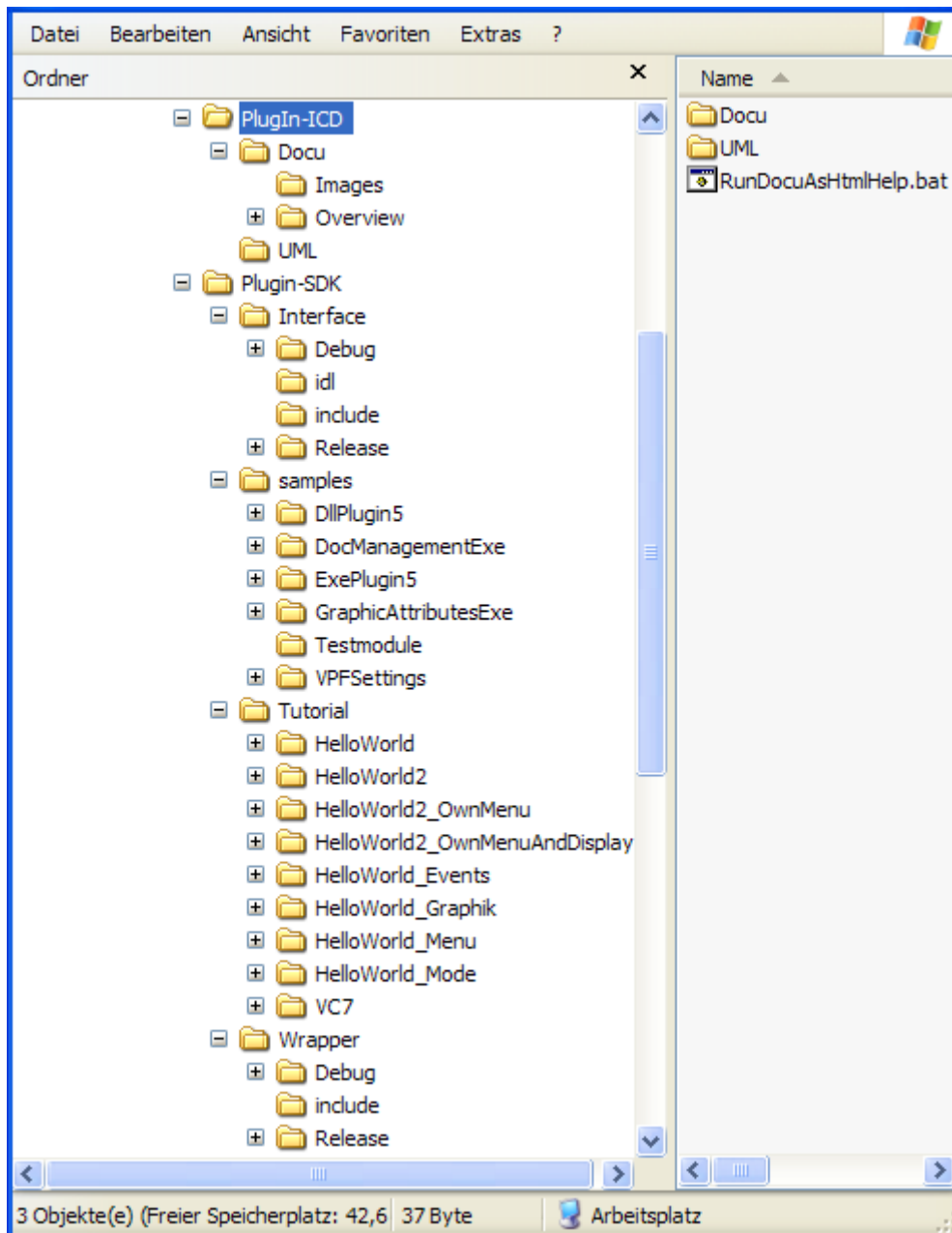


Abbildung 24: Struktur von Plugin-ICD und Plugin-SDK

Öffnen Sie über das Menü *Projekt/Einstellungen* den Dialog *Projekteinstellungen*. Wählen Sie links oben neben *Einstellungen für:* in der Auswahlliste *Alle Konfigurationen* (1). Anschließend klicken Sie auf das Register *C/C++* (2) und selektieren in der Auswahlliste *Kategorie* die Option *Präprozessor* (3). Unter *Zusätzliche Include-Verzeichnisse* tragen Sie die Pfade zu den Include-Dateien des Interface- und Wrapper-Ordners ein (4).

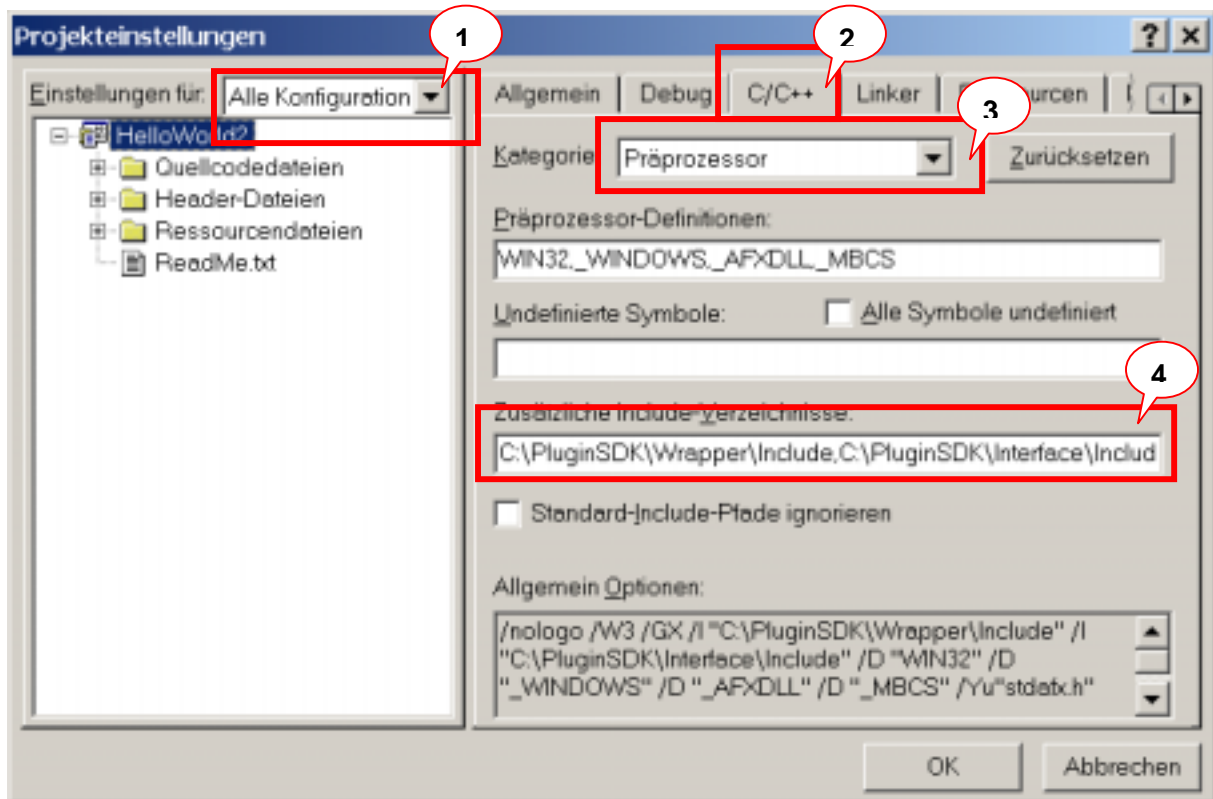


Abbildung 25: Einstellungen für Include-Pfade

Wählen Sie links oben in der Auswahlliste *Einstellungen für: Debug* aus (1). Anschließend wechseln Sie auf das Register *Linker* (2) und stellen Sie in der Auswahlliste neben *Kategorie* die Option *Eingabe* ein (3). Unter *Objekt-/Bibliothek-Module* tragen Sie die Bibliotheksnamen aus den Verzeichnissen **C:\PluginSDK\Interface\Debug** und **C:\PluginSDK\Wrapper\Debug** ein, z.B. WrapperD5.lib InterfaceD5.lib (4). Zusätzlich tragen Sie die Bibliothek rpctr4.lib ein. Unter *Zusätzliche Bibliothek-Verzeichnisse* tragen Sie die oben erwähnten Pfade zu den lib-Dateien des Interface- und Wrapper-Ordners ein (5).

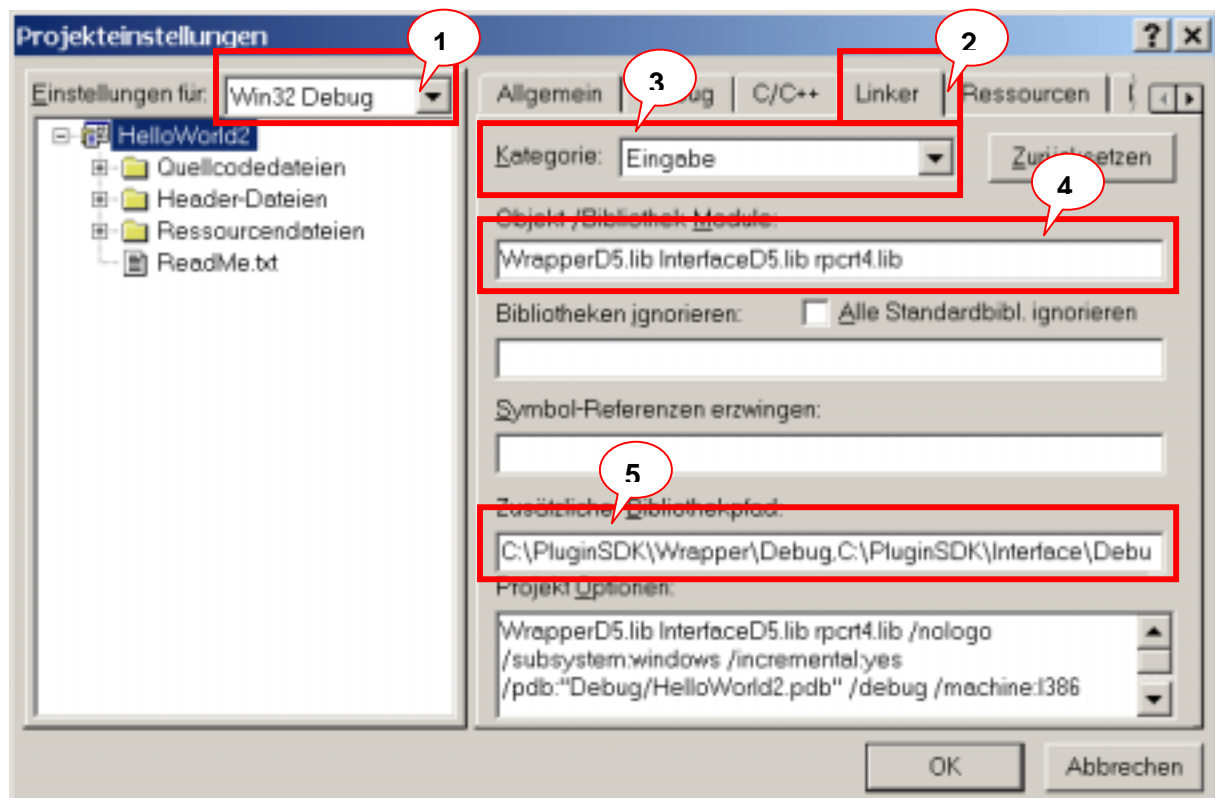


Abbildung 26: Einstellungen für Debug-Libraries

Wählen Sie in der Auswahlliste links oben *Einstellungen für: Release* aus **(1)**. Stellen Sie in der Auswahlliste neben *Kategorie* die Option *Eingabe* ein **(2)**. Unter Objekt-/Bibliothek-Module tragen Sie die Bibliotheksnamen aus den Verzeichnissen **C:\PluginSDK\Interface\Release** und **C:\PluginSDK\Wrapper\Release** ein, z.B. WrapperR5.lib InterfaceR5.lib **(3)**. Zusätzlich tragen Sie die Bibliothek rpctr4.lib ein. Unter *Zusätzliche Bibliothek-Verzeichnisse* tragen Sie die oben erwähnten Pfade zu den lib-Dateien des Interface- und Wrapper-Ordners ein **(4)**.

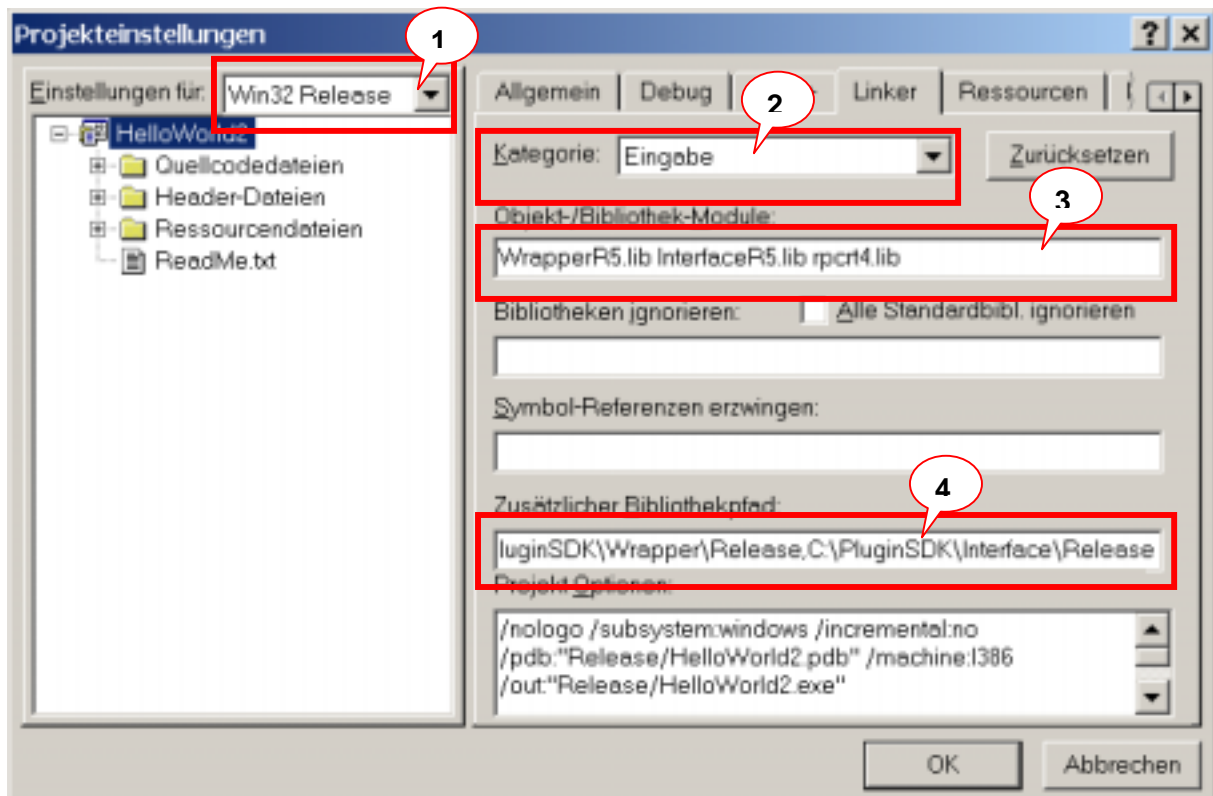
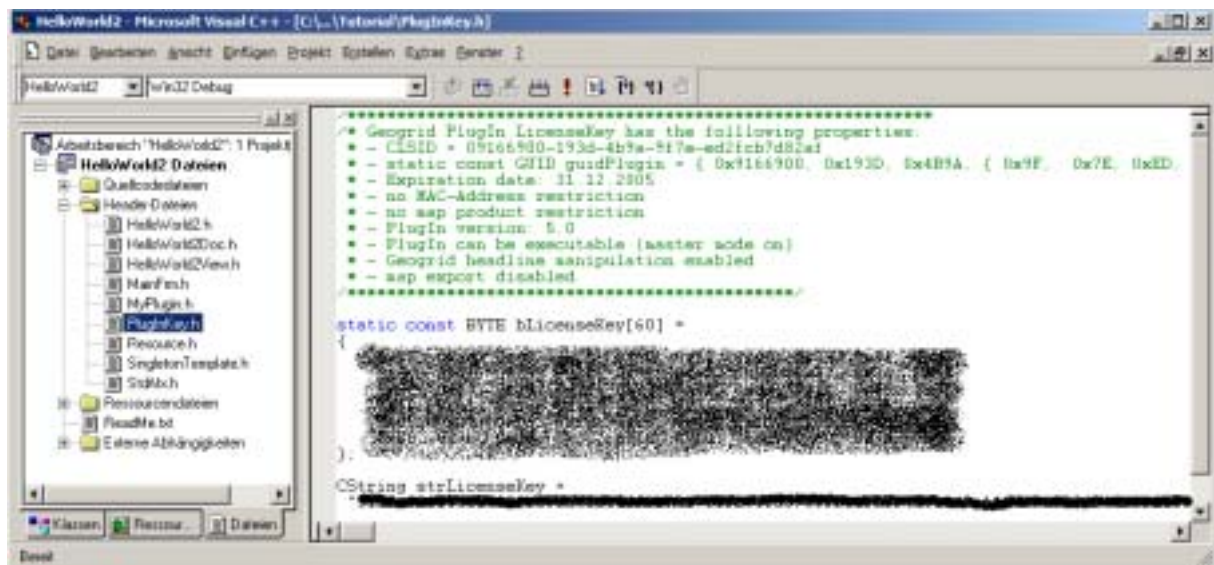


Abbildung 27: Einstellungen für Release-Libraries

4.2.3 Lizenz-Schlüssel

Für die Nutzung des Plugins ist ein Lizenz-Schlüssel erforderlich. Sollten Sie noch nicht in Besitz eines solchen Schlüssels sein, können Sie ihn beim Hersteller der Geogrid® SW und der Plugin-Schnittstelle beziehen.

Der Schlüssel ist in einer Include-Datei enthalten, die Sie Ihrem Projekt hinzufügen. In unserem Beispiel heißt Sie PlugInKey.h.



```
*****  
* Geogrid Plugin LicenseKey has the following properties:  
* - GUID = 05166900-1938-4b9a-917a-ed21cb7d82a1  
* - static const GUID guidPlugin = { 0x9166900, 0x1938, 0x4b9a, { 0x91, 0x7e, 0xed,  
* - Expiration date: 31 12 2005  
* - no MAC-address restriction  
* - no asp product restriction  
* - Plugin version: 5.0  
* - Plugin can be executable (master code on)  
* - Geogrid headline manipulation enabled  
* - asp export disabled  
*****  
  
static const BYTE bLicenseKey[60] =  
{  
[REDACTED]  
};  
  
CString strLicenseKey =
```

Abbildung 28: Lizenz-Datei

4.2.4 Implementierung der Schnittstelle IGeoPlugin

Dieser Punkt unterscheidet sich nicht von dem bei einem DLL-Plugin. Zur Vollständigkeit führen wir ihn hier noch einmal auf.

CMyPlugin wird von der Klasse CGeoPlugin, die die Schnittstelle IGeoPlugin implementiert, abgeleitet. Die Klasse CGeoPlugin enthält virtuelle Methoden, die wir überschreiben können. In diesem Beispiel beschränken wir uns auf Init(), Destroy() sowie GetLicenseKey().

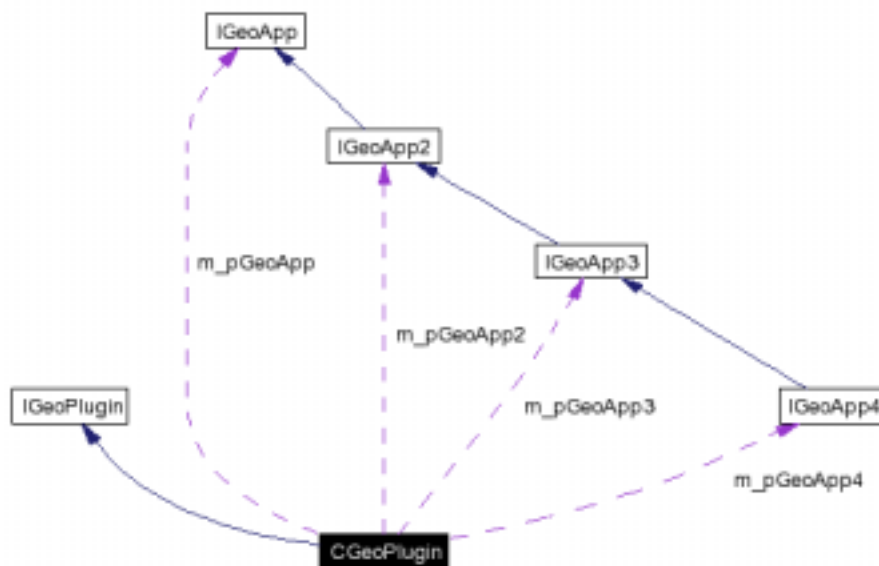


Abbildung 29 Ableitungshierarchie für CGeoPlugin

MyPlugin.h:

```
// specification of the PlugIn class
class CMyPlugin : public CGeoPlugin
{
public:
    void Init();
    void Destroy();

    CString GetLicenseKey();
};
```

MyPlugin.cpp:

```
// Implementation of the PlugIn class
#include "stdafx.h"

#include "Interface.h"
#include "Wrapper.h"

#include "MyPlugin.h"
#include "PlugInKey.h"

// Wird in einem EXE - Plugin nicht benötigt
// IMPLEMENT_GEOPLUGIN_CLASS(CMyPlugin)
```

```
void CMyPlugin::Init()
{
    AppMessageBox("Hello World!", "MyPlugin");
}

void CMyPlugin::Destroy()
{
    // Schreibe die aktuelle Zeit in das Geogrid INI file
    CTime curTime = CTime::GetCurrentTime();
    ProfileSetString ("MyPlugin", "ShutdownTime",
        curTime.Format ("%d.%m.%Y %H:%M:%S"));

    AfxMessageBox( "Good bye Plugin" );
}

CString CMyPlugin::GetLicenseKey()
{
    // Returns the license key for authorization of the plugin.
    // This method is called once before the plugin is
    // initialised.
    // 'strLicenseKey' is the license key defined in
    // 'Geogrid_Plugin_Licens_Key.h'.
    // This include file has to be requested from Dornier GmbH,
    // Friedrichshafen.

    return strLicenseKey;
}
```

IMPLEMENT_GEOPLUGIN_CLASS(CMyPlugin) wird im EXE-Plugin nicht benötigt und kann gelöscht oder auskommentiert werden. Bleibt diese Zeile erhalten, dann kann die Datei CMyPlugin.cpp sowohl in einem DLL-Plugin als auch in einem EXE-Plugin verwendet werden. Fehlt diese Zeile in einem DLL-Plugin, dann führt das zu folgendem Link-Fehler:

Kompilierung läuft...

MyPlugin.cpp

Linker-Vorgang läuft...

Bibliothek Debug/HelloWorld.lib und Objekt Debug/HelloWorld.exp wird erstellt
HelloWorld.obj : error LNK2001: Nichtaufgeloestes externes Symbol "public: virtual long __stdcall
*CGeoPluginClass::CreateInstance(struct IUnknown *,struct _GUID const &,void * *)"*
(?CreateInstance@CGeoPluginClass@@@UAGJPAUIUnknown@@@ABU_GUID@@@PAPAX@Z)
Debug/HelloWorld.dll : fatal error LNK1120: 1 unaufgeloeste externe Verweise
Fehler beim Ausführen von link.exe.

HelloWorld.dll - 2 Fehler, 0 Warnung(en)

Die *Init()* Methode dient, wie der Name schon sagt, der Initialisierung des Plugins. Sie ist die erste Methode, die von Geogrid aufgerufen wird, wenn die Anmeldung des Plugins erfolgreich war. D.h. der durch *GetLicenseKey* an Geogrid® übergebene Lizenz Schlüssel ist gültig.

In unserem Beispiel enthält *Init* nur die Methode

```
AppMessageBox("Hello World!", "MyPlugin");
```

Sie ist in der Schnittstelle *IGeoApp* definiert und wird durch *CGeoPlugin* implementiert. Da *CMyPlugin* von *CGeoPlugin* abgeleitet ist können wir sie einfach aufrufen.

Diese Methode zeigt eine einfache MessageBox in der Mitte des Geogrid® Applikations-Rahmens an. Als Parameter werden zwei CString Werte übergeben. Der Erste enthält die Nachricht, die angezeigt werden soll, der Zweite enthält den Titel der Dialogbox.

Geogrid® ruft *Destroy* auf, wenn es beendet wird. Damit können vom Plugin noch Aufräumarbeiten geleistet werden. In unserem Beispiel verwenden wir neben einem nochmaligen Aufruf der Methode *AppMessageBox* die Methode *ProfileSetString*. Sie wird dazu verwendet um Einträge in der Ini-File der Geogrid® Applikation vorzunehmen. Die drei übergebenen Parameter haben den Typ CString. Sie definieren zuerst die Sektion in der INI-File, dann den Schlüssel und schließlich den Wert:

```
[MyPlugin]  
ShutdownTime=31.08.2004 23:47:20
```

Wir haben sie benutzt, um die Zeit, wann Geogrid® zuletzt beendet wurde zu dokumentieren. Zum Ermitteln der Zeit ist die MFC Klasse CTime verwendet worden.

4.2.5 Ergänzungen in HelloWorld.cpp

Includieren Sie die Datei Wrapper.h. Sie enthält die Definitionen der Wrapperklassen und – Methoden. Fügen Sie die GUID aus der Datei PlugInKey.h hinzu.

Nun werden das bereits unter Punkt 3. erwähnte Makro *IMPLEMENT_PLUGIN_MODULE_ROUTINES* eingefügt, welches einige Standard COM Funktionen implementiert. .

Beispiel:

```
#include "Wrapper.h"  
#include "MyPlugin.h"  
  
//CLSID = 09166900-193d-4b9a-9f7e-ed2fcb7d82af  
static const GUID CLSID_GeoPlugin =  
{ 0x9166900, 0x193D, 0x4B9A, { 0x9F, 0x7E, 0xED, 0x2F, 0xCB, 0x7D,  
0x82, 0xAF } }  
  
IMPLEMENT_PLUGIN_MODULE_ROUTINES()
```

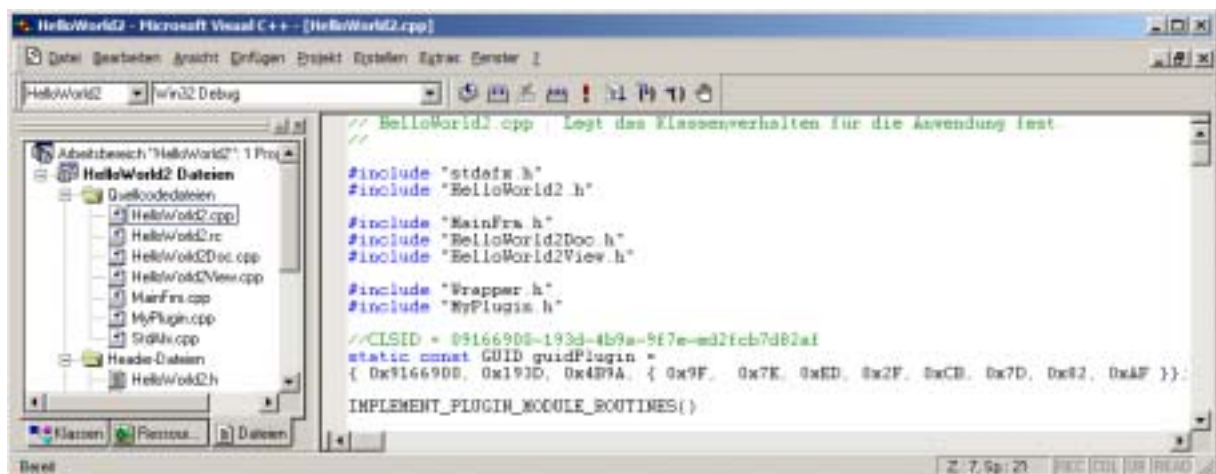


Abbildung 30: Plugin Class-Id

EXE Plugins sind für die Erzeugung einer Instanz der Plugin Klasse selbst verantwortlich, deswegen müssen sie sich nicht registrieren. Wir verwenden die Wrapper Klasse *CGPGeogrid*, die die IGeogrid Schnittstelle implementiert, um den Kontakt zur Geogrid® Applikation herzustellen.

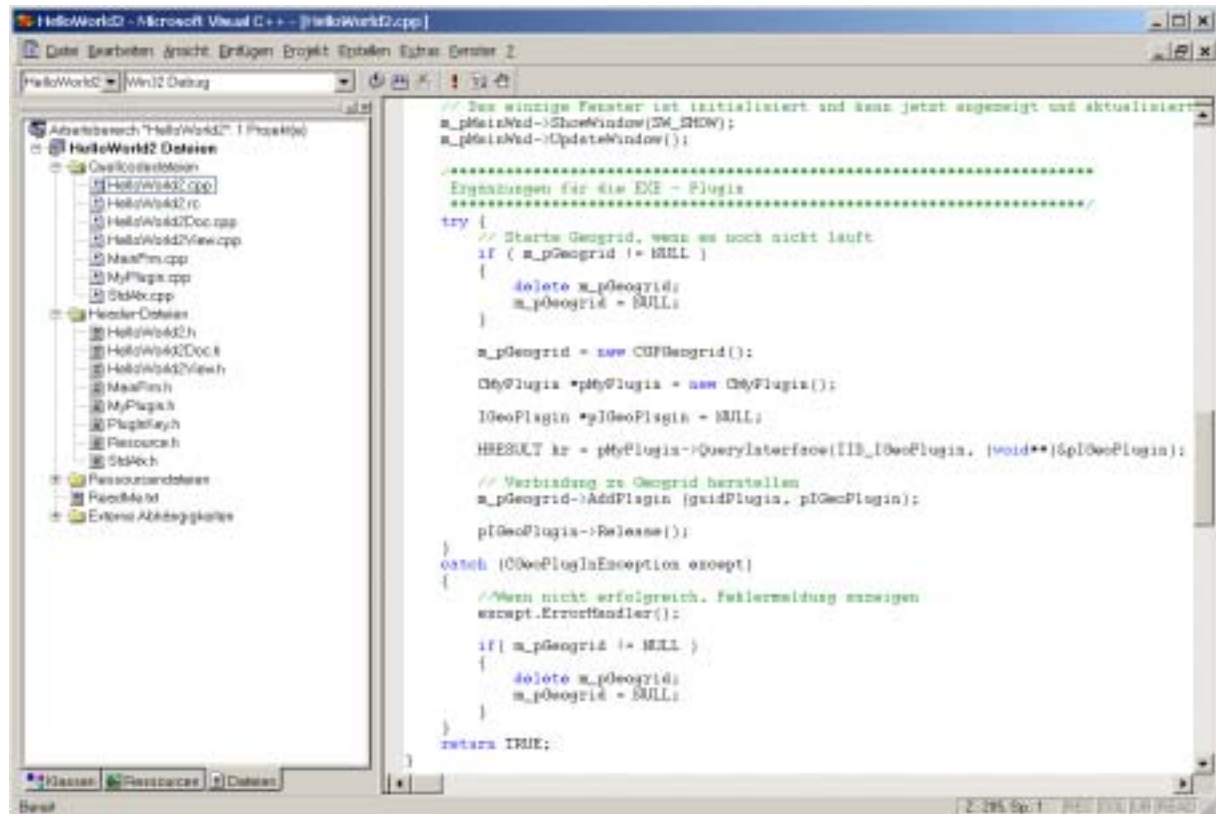


Abbildung 31: Ergänzungen in InitInstance

Implementieren Sie die "InitInstance" Methode der Haupt-Klasse von "CHelloWorld2App": Diese Methode wird beim Start des EXE Plugin aufgerufen. Das Erzeugen einer "CGPGeogrid" Instanz initialisiert das Plugin Interface (wichtig für den COM Mechanismus) und startet die Geogrid® Applikation, wenn sie noch nicht läuft. Mit dem Aufruf von "AddPlugin" wird das Plugin "MyPlugin" in die Geogrid® Applikation eingehangen. Beachten Sie, dass beim Fehlschlag der Initialisierung Exceptions geworfen werden (Verwenden Sie daher try/catch Blöcke). Bei erfolgreicher Initialisierung wird die Methode "Initialize" der Plugin Instanz "MyPlugin" durch die Geogrid® Applikation aufgerufen.

```

BOOL CHelloWorld2App::InitInstance()
{
    ...

    // Das einzige Fenster ist initialisiert und kann jetzt angezeigt und
    // aktualisiert werden.
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    /*****
    Ergänzungen für das EXE - Plugin
    *****/
    try {
        // Starte Geogrid, wenn es noch nicht läuft
    
```

```
    if ( m_pGeogrid != NULL )
    {
        delete m_pGeogrid;
        m_pGeogrid = NULL;
    }

    m_pGeogrid = new CGPGeogrid();

    m_pMyPlugin = new CMyPlugin();
    m_pMyPlugin->AddRef();

    IGeoPlugin *pIGeoPlugin = NULL;

    HRESULT hr = m_pMyPlugin->QueryInterface(IID_IGeoPlugin,
                                             (void**)&pIGeoPlugin);

    // Verbindung zu Geogrid herstellen, Plugin einhängen
    m_pGeogrid->AddPlugin (CLSID_GeoPlugin, pIGeoPlugin);

    pIGeoPlugin->Release();
}
catch (CGeoPlugInException except)
{
    //Wenn nicht erfolgreich, Fehlermeldung anzeigen
    except.ErrorHandler();

    if( m_pGeogrid != NULL )
    {
        delete m_pGeogrid;
        m_pGeogrid = NULL;
    }
}
return TRUE;
}
```

Wenn das EXE-Plugin eine Verbindung zu Geogrid® aufbaut, wird zunächst der Konstruktor der Klasse *CGPGeogrid* aufgerufen. Darin wird die COM-Bibliothek initialisiert. Anschließend erfolgt die Registrierung der Plugin-Schnittstellen. Wenn Geogrid® noch nicht läuft wird dieses nun durch den Aufruf der COM-Funktion *CoGetClassObject* gestartet. Beim Start wird das *CPluginController*-Objekt eine Instanz von *CGeogrid* erzeugt. Dabei handelt es sich um ein COM-Objekt, welches ebenfalls in der Registrierungsdatenbank eingetragen wird. Dieses COM-Objekt wartet nun darauf, dass ein EXE-Plugin Verbindung aufnimmt.

Auf der Plugin-Seite wird nun zunächst ein *CMyPlugin*-Objekt erzeugt und anschließend dessen Referenzzähler um eins erhöht (*AddRef()*). Mit *QueryInterface* wird ein Schnittstellenzeiger vom Typ *IGeoPlugin* angefordert und der Methode *AddPlugin* des *CGPGeogrid*-Objekts als Parameter übergeben. Von hier wird der Aufruf an *CGeogrid* weitergeleitet. Schließlich gelangt er zum *CPluginController*. Über den Schnittstellenzeiger wird der *LicenseKey* abgefragt und bei gültiger Lizenz die Initialisierungsmethode des Plugins aufgerufen.

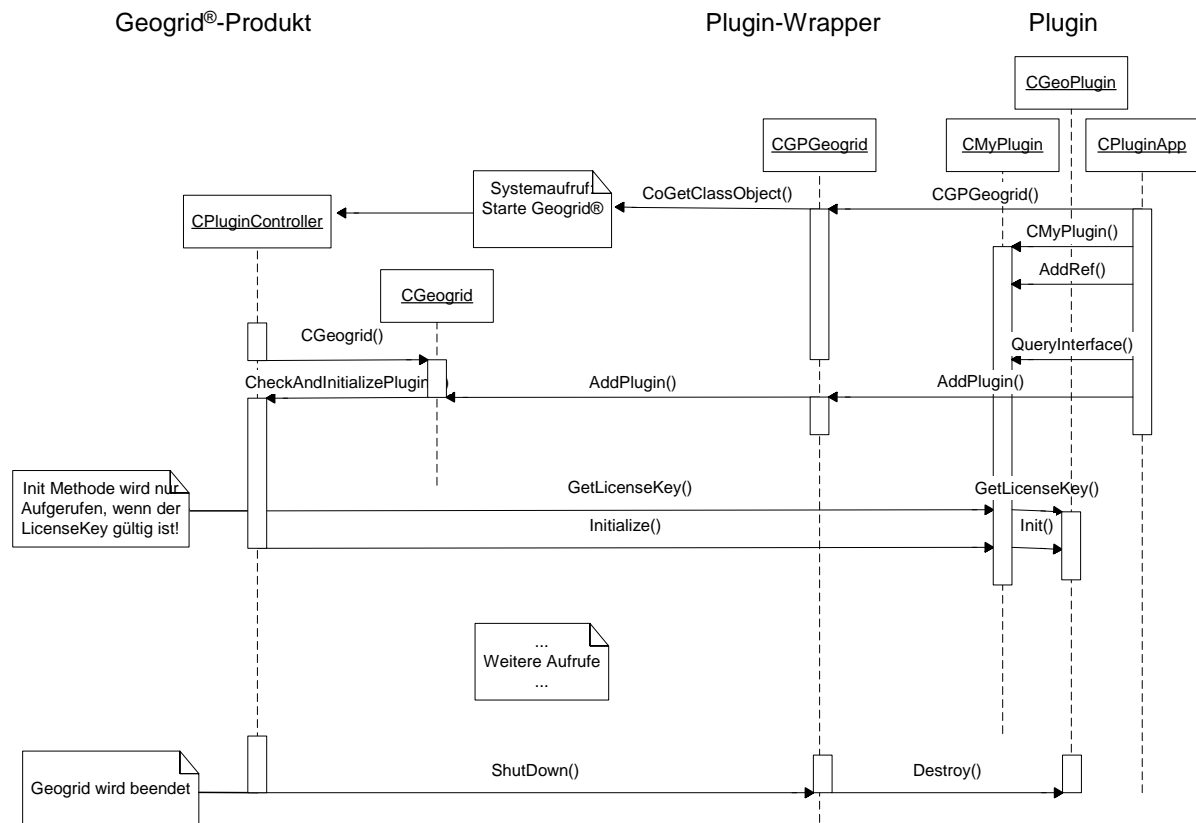


Abbildung 32: Sequenzdiagramm zur Kommunikation zwischen EXE-Plugin-und Geogrid®

In Abbildung 26 wird die Kommunikation zwischen der EXE-DLL und Geogrid® in einem Blockdiagramm dargestellt. Die Schnittstellen werden durch eine Linie mit einem Kreis (—○) symbolisiert. Die allen COM-Objekten gemeinsame Schnittstelle IUnknown wird üblicherweise oberhalb der COM-Komponenten eingezeichnet. Die Komponenten sind als Rechtecke mit abgerundeten Ecken wiedergegeben. Sowohl über das EXE-Plugin (hier als Client.exe bezeichnet) als auch über die Geogrid® Applikation kann ein Datenaustausch erfolgen. Im Gegensatz zur Plugin-DLL erfolgt zum Einen die Kommunikation über die Prozessgrenze hinweg. Zum Anderen muss Geogrid® für die Kommunikation mit dem EXE-Plugin die Schnittstelle IGeogrid implementieren, über die sich das EXE-Plugin bei Geogrid® anmelden kann.

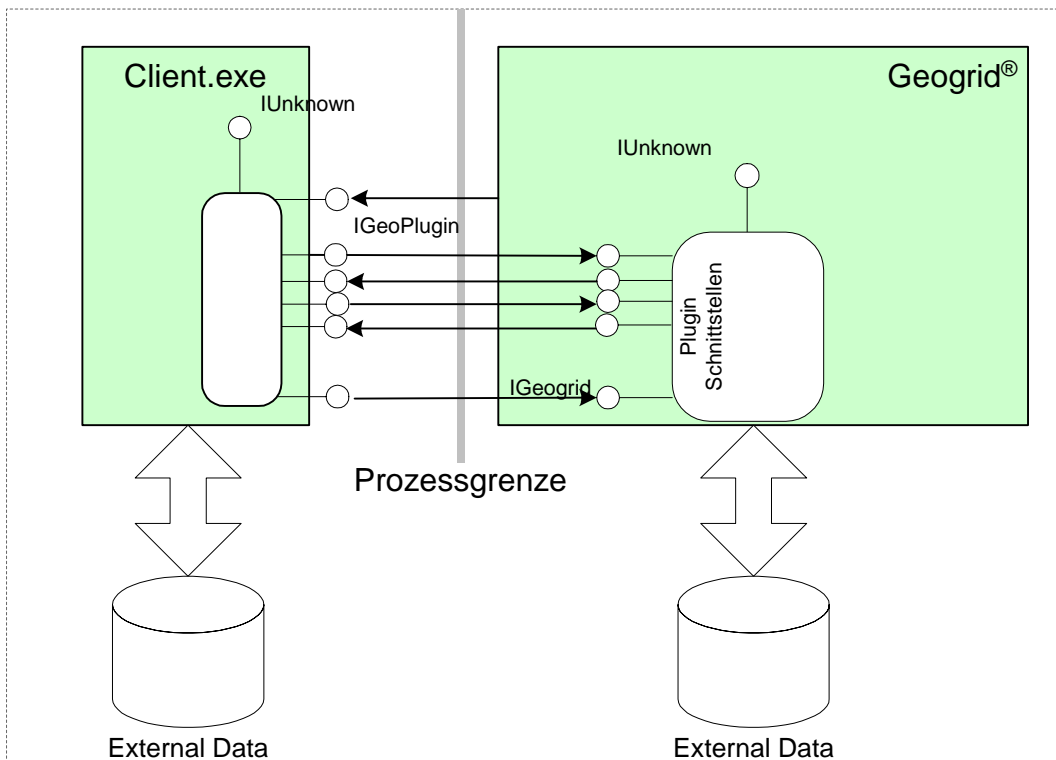


Abbildung 33: Kommunikation Zwischen EXE-Plugin und Geogrid®

4.2.6 (Klassenassistent) Verbindung trennen – ExitInstance

Im vorhergehenden Kapitel wurde erläutert, wie die Verbindung zwischen der Plugin-Applikation und Geogrid® zustande kommt. Wird das Plugin beendet, so sollte auch die Verbindung zu Geogrid® wieder gelöst werden. Dazu wird die Methode `ExitInstance` für die Plugin-Applikation implementiert. Diese Methode wird von dem Applikationsrahmen aufgerufen, wenn das Plugin beendet wird. Zur Implementierung dieser Methode verwenden wir den Klassenassistenten.

Im Menü *Ansicht* des Developer Studios klicken Sie auf *Klassen-Assisten...*, um den MFC-MFC-Klassen-Assistenten auszuführen. Im Dialogfeld *MFC-Klassen-Assistent* klicken Sie auf das Register *Nachrichtenzuordnung* (1). Unter *Klassenname* stellen Sie *CHelloWorld2App* ein (2). In der Auswahlliste *Nachrichten* selektieren Sie *ExitInstance* (3) und klicken anschließend auf *Funktion hinzufügen* am rechten Rand der Dialogbox (4).

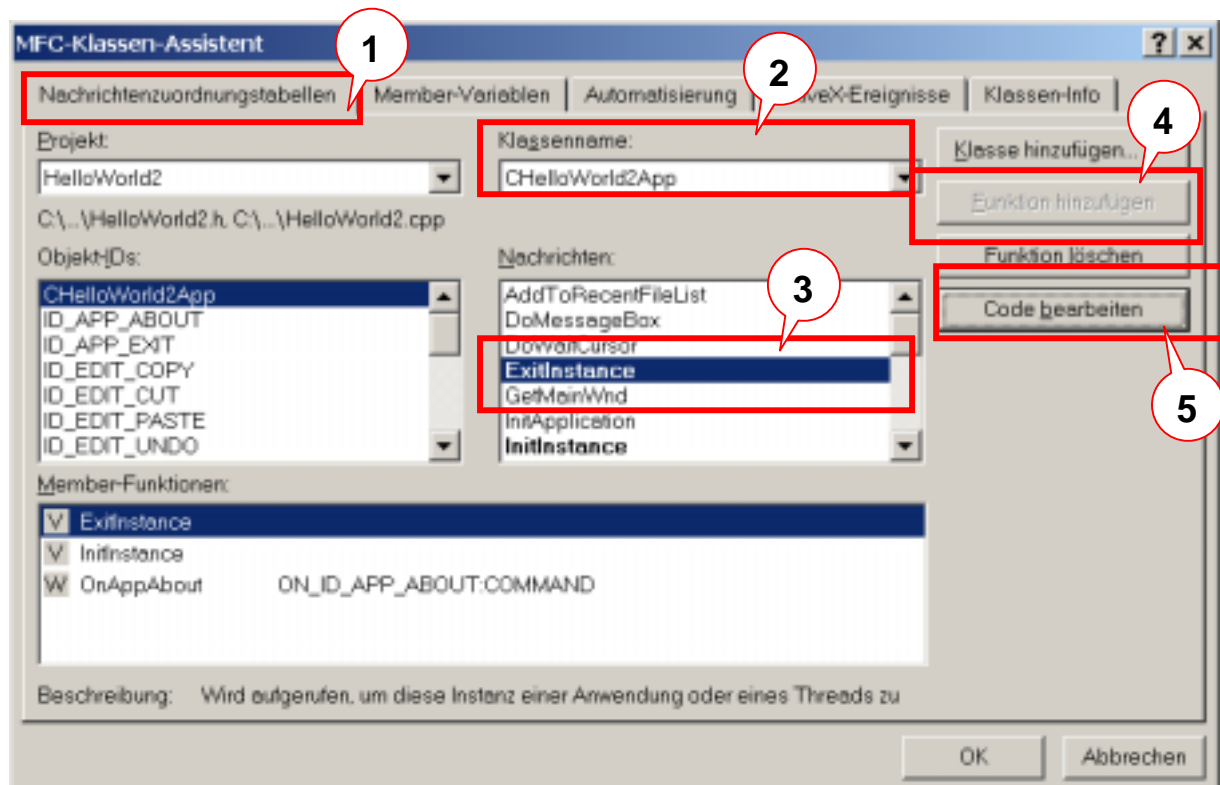


Abbildung 34: Klassen-Assistent, Hinzufügen der Methode `ExitInstance`

Implementierung der Methode `ExitInstance`:

Klicken Sie nun auf *Code bearbeiten* (5) um die Methode zu implementieren. Fügen Sie in das vom Klassen-Assistent erzeugte Gerüst den unten fettgedruckten Code.

```
int CHelloWorld2App::ExitInstance()  
{  
    // TODO: Speziellen Code hier einfügen und/oder Basisklasse aufrufen  
    if (m_pGeogrid != NULL)  
    {  
        try {  
            // entferne PlugIn aus Geogrid-Applikation  
            m_pGeogrid->RemovePlugin (CLSID_GeoPlugin);  
        }  
    }  
}
```

```
        catch (CGeoPlugInException except)
        {
            // bei Misserfolg Fehlermeldung anzeigen
            except.ErrorHandler();
        }

        delete m_pGeogrid;
        m_pGeogrid = NULL;
    }

    return CWinApp::ExitInstance();
}
```

4.2.7 Das Plugin testen

Erstellen Sie eine Version Ihres Plugins und starten Sie sie. Geogrid® wird automatisch gestartet, sofern es noch nicht läuft. In diesem Zusammenhang ist wichtig, dass Sie Ihre Geogrid® Applikation nach der Installation mindestens einmal manuell gestartet haben, damit Geogrid® sich registrieren kann.

4.3 Hello World in einer .Net-Umgebung

Klicken Sie im Menü *Datei auf Neu* und anschließend auf *Projekt*. Im Dialog *Neues Projekt* selektieren Sie unter *Projekttypen* im Baum *Visual C++-Projekte*. Im Fenster *Vorlagen* selektieren Sie *MFC-DLL* (1). Unter *Name* tragen Sie **HelloWorld** ein (2), *Speicherort* sei **C:\Plugin\VC7** (3).

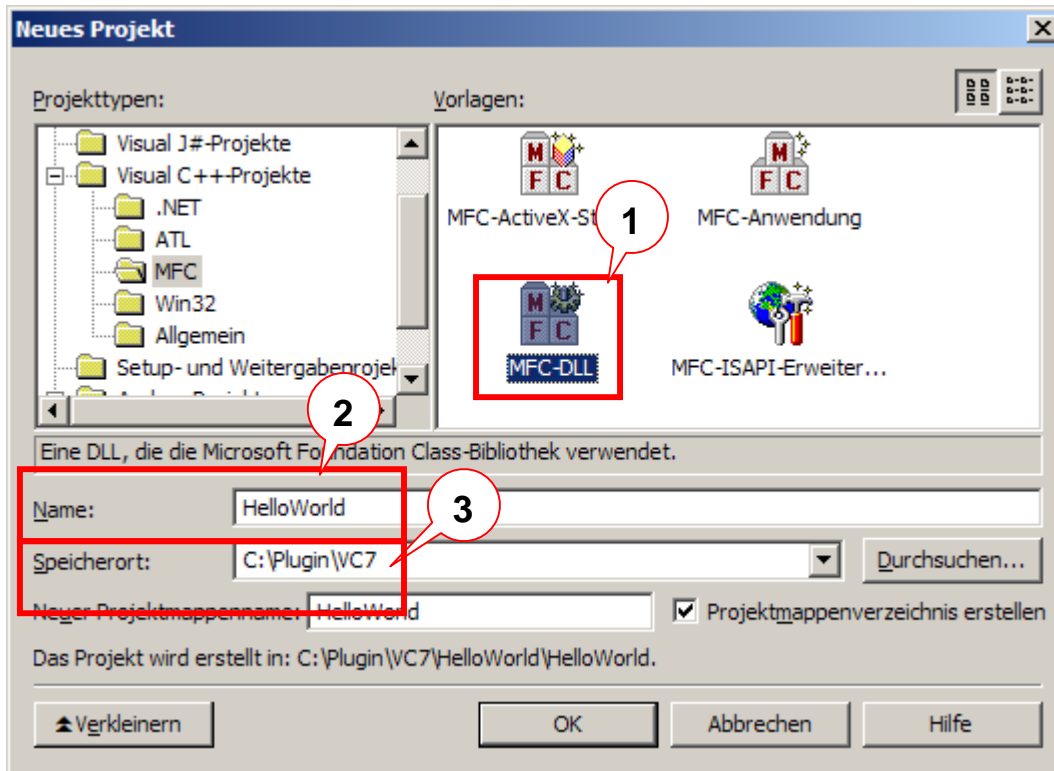


Abbildung 35: Neues DLL-Projekt in .NET anlegen

Klicken Sie auf OK. Anschließend erscheint der Dialog *MFC-DLL-Assistent-HelloWorld*.

Hier klicken Sie einfach auf **Fertig stellen** (1).

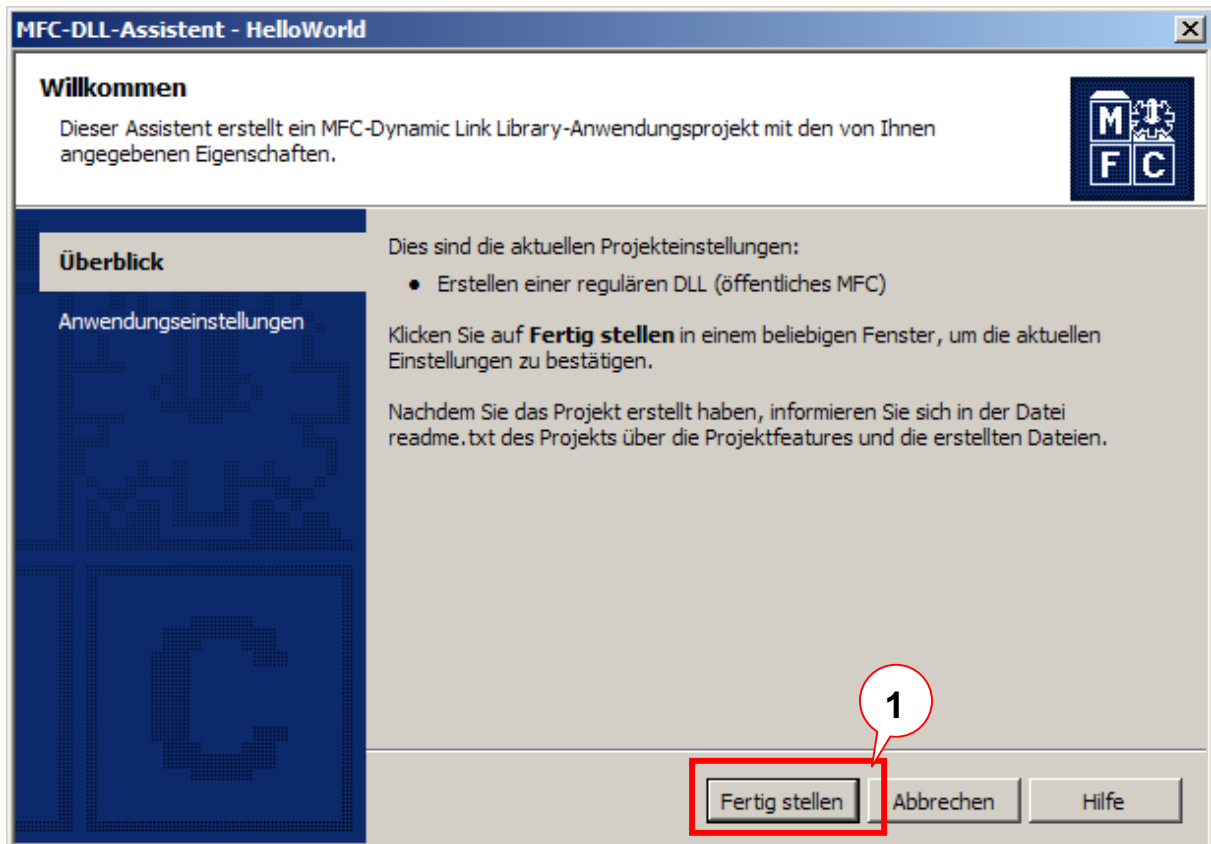


Abbildung 36: MFC-DLL-Assistent-HelloWorld

4.3.1 Projekteinstellungen

Im Menü *Projekt* klicken Sie auf *Eigenschaften von HelloWorld* und öffnen damit den Dialog

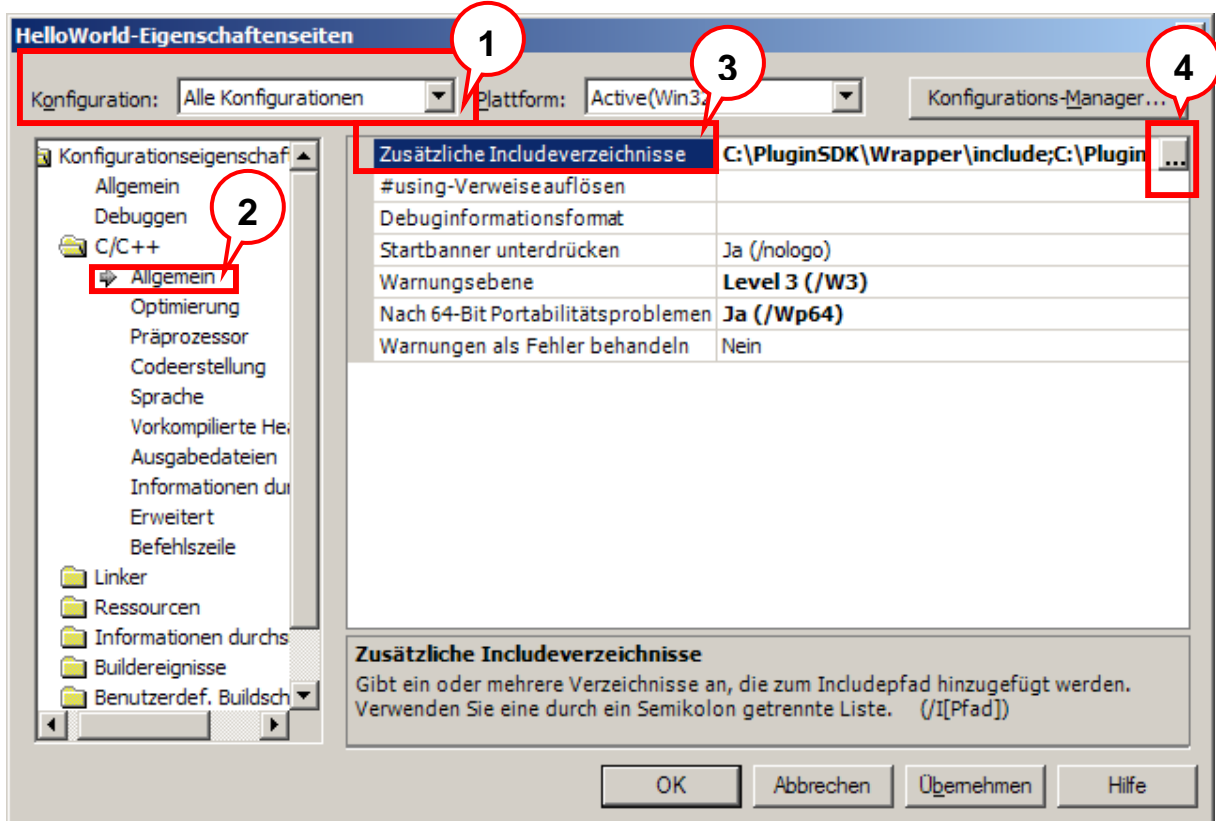


Abbildung 37: Zusätzliche Includeverzeichnisse

Stellen Sie in der Auswahlliste Konfiguration (oben links) *Alle Konfigurationen* ein (1). In dem Baum auf der linken Seite selektieren Sie *Allgemein* in dem Ordner *C/C++* (2). Markieren Sie in dem rechten Feld *Zusätzliche Includeverzeichnisse* (3). Durch einen Klick auf den Button mit den drei Punkten am rechten Rand dieses Feldes (4) öffnen Sie den Dialog *Zusätzliche Includeverzeichnisse*:

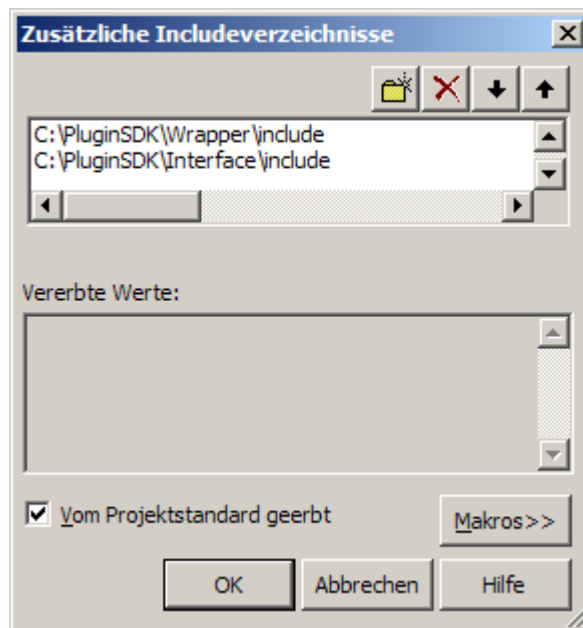


Abbildung 38: Editor für zusätzliche Plugin-Pfade

Fügen Sie hier die Pfade zu den Include-Dateien ein.

In dem Baum auf der linken Seite selektieren Sie *Präprozessor* (1). In dem Ordner *C/C++* markieren Sie in dem rechten Feld *Präprozessordefinitionen* (2). Durch einen Klick auf den Button mit den drei Punkten am rechten Rand dieses Feldes (3) öffnen Sie den Dialog *Präprozessordefinitionen*. Fügen Sie hier den Eintrag `_WIN32WINNT>=0x0400` hinzu (4). Funktionen hängen von einer bestimmten Windows-Version ab und werden bedingt übersetzt. Damit diese Funktionen auch auf einem neueren Betriebssystem übersetzt werden können, muss das oben erwähnte Macro angegeben werden.

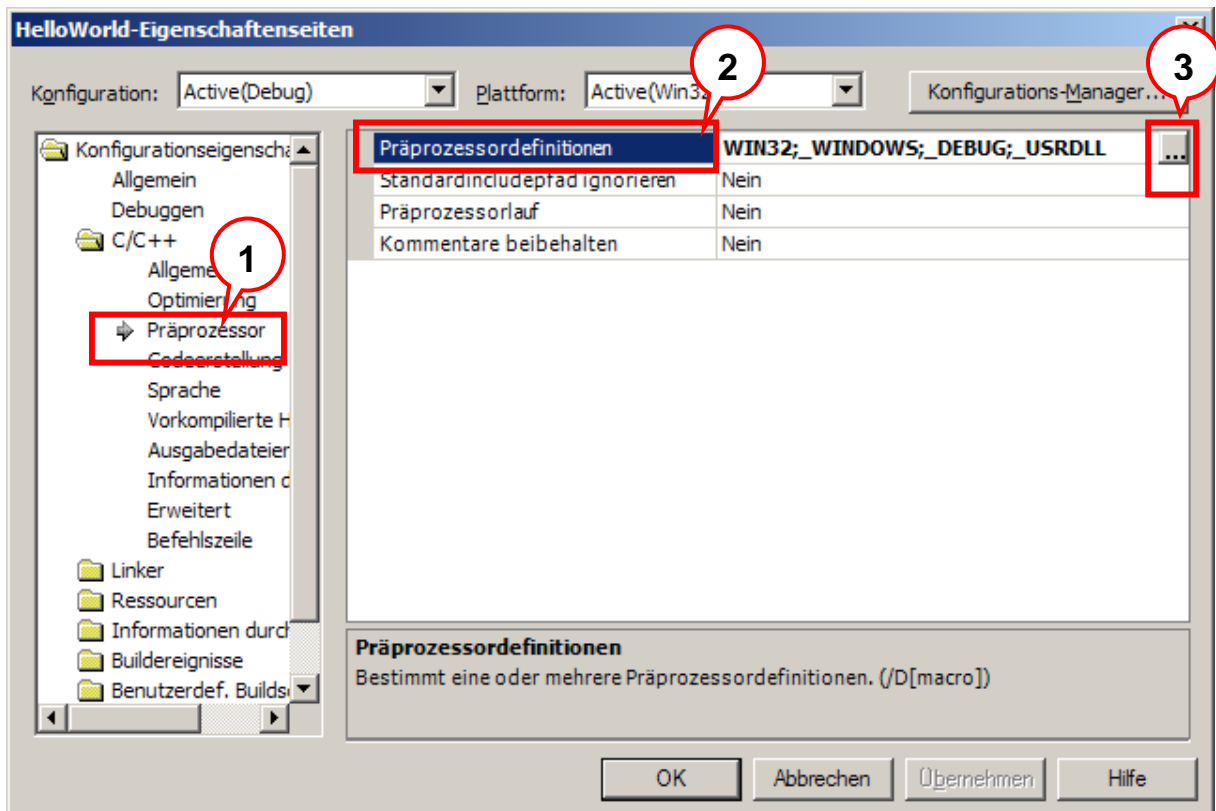


Abbildung 39: Präprozessordefinitionen hinzufügen

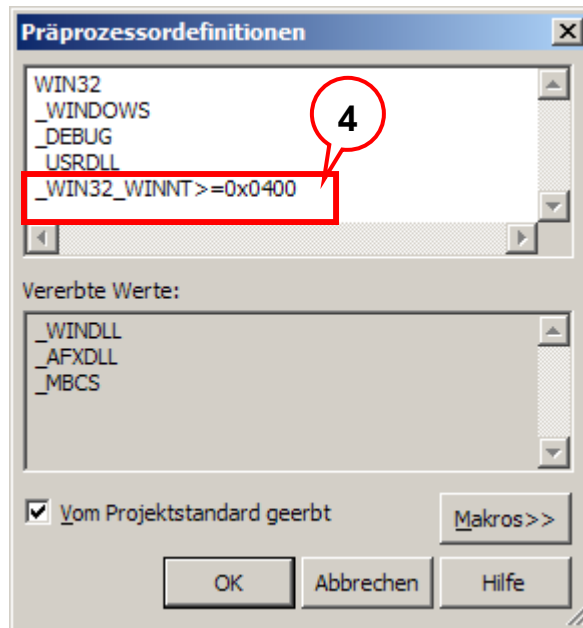


Abbildung 40: Editor für Präprozessordefinitionen

In dem Baum auf der linken Seite selektieren Sie *Sprache* in dem Ordner *C/C++* (1). Markieren Sie in dem rechten Feld *wchar_t als built-in Typ behandeln* (2) ändern Sie die Einstellung von *Ja* (*/Zc:wchar_t*) auf *Nein* (3) da sonst folgender Link Fehler auftritt:

```
HelloWorld error LNK2001: Nichtaufgelöstes externes Symbol "public: virtual long __stdcall  
CGeoPlugin::GetLicenseKey(wchar_t * *)"  
(?GetLicenseKey@CGeoPlugin@@UAGJPAPA_W@Z)
```

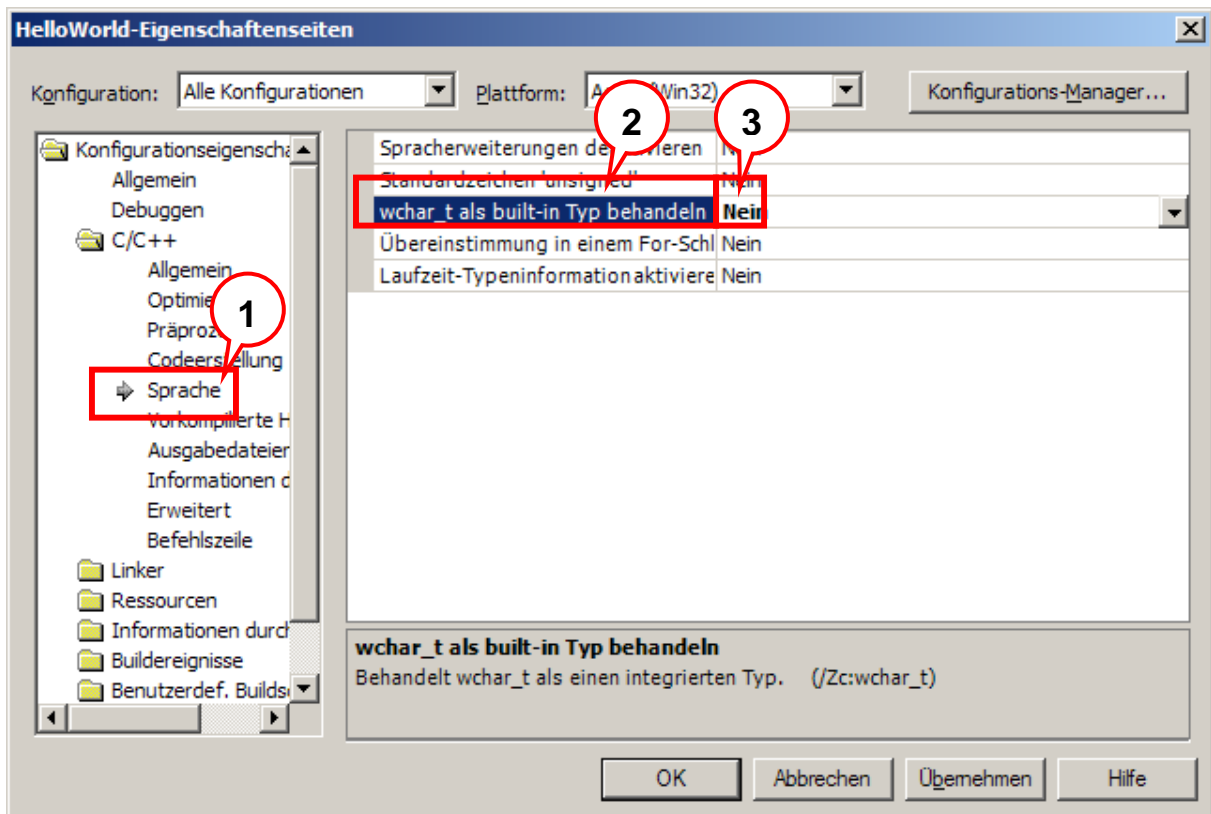


Abbildung 41: Einstellungen für *wchar_t als built in Typ* ändern

Stellen Sie unter Konfigurationen: *Active(Debug)* ein (1). Selektieren Sie im Ordner Linker den Eintrag *Allgemein* (2). Fügen Sie im rechten Teil der Dialogbox unter *Zusätzliche Bibliotheksverzeichnisse* die Pfade zu den Bibliotheken *Wrapper5_VC7D.lib* und *Interface5_VC7D.lib* ein (3) - (5). Achten Sie darauf, dass Sie die Bibliotheken aus dem Ordner VC7 verwenden, da diese mit der entsprechenden Entwicklungsumgebung erstellt wurden:

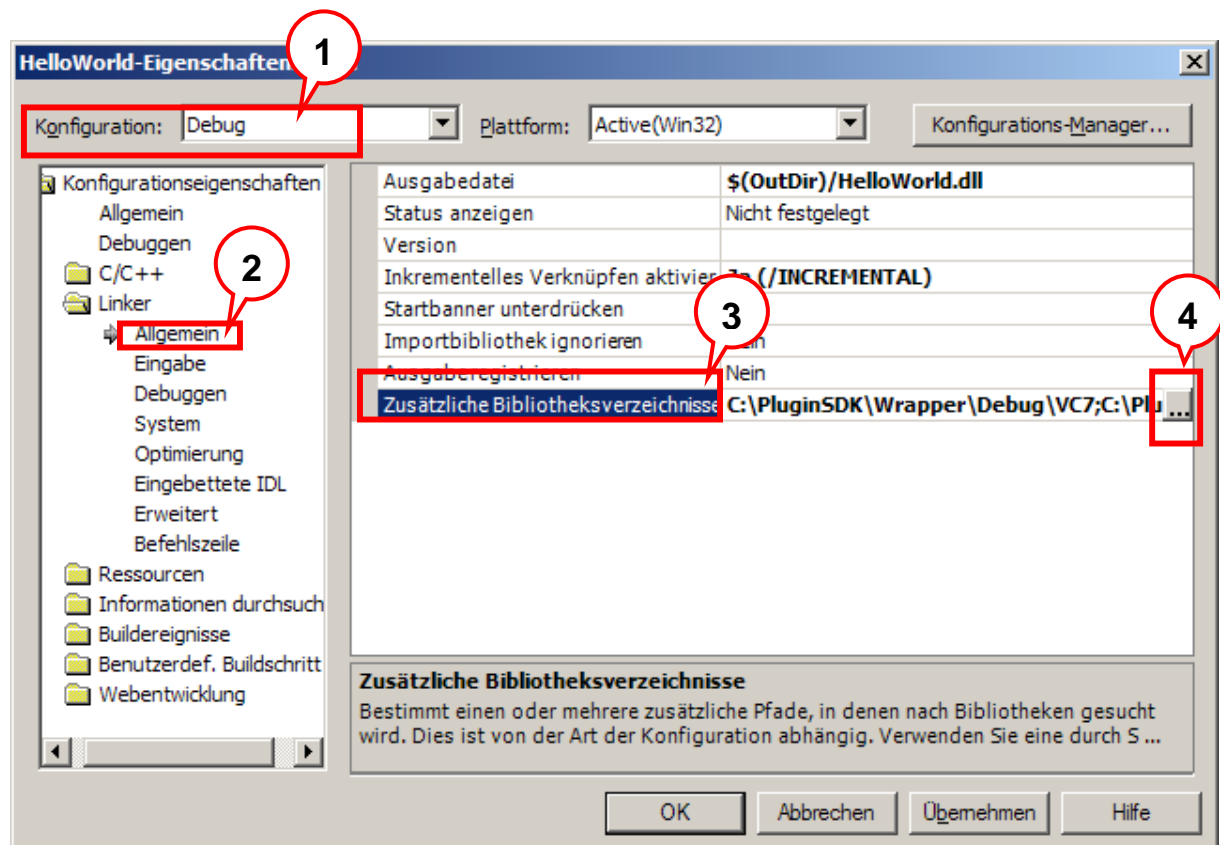


Abbildung 42: Zusätzliche Bibliotheksverzeichnisse hinzufügen

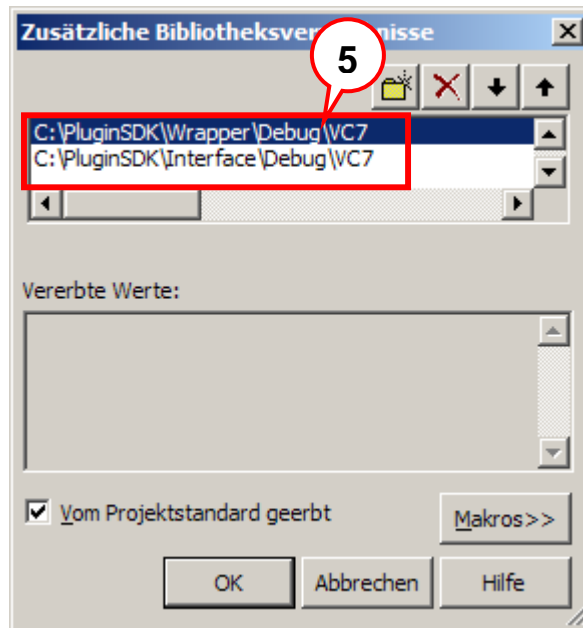


Abbildung 43: Editor für zusätzliche Bibliothekspfade

In dem Ordner *Linker* (1) klicken Sie auf *Eingabe* (2) und fügen unter *Zusätzliche Abhängigkeiten* (3) die Bibliotheken *Interface5_VC7D.lib* und *Wrapper5_VC7D.lib* hinzu (4),(5). Unter *Bibliothek ignorieren* fügen Sie *mfc42.lib;mfc42.lib;msvcrt.lib* ein (6).

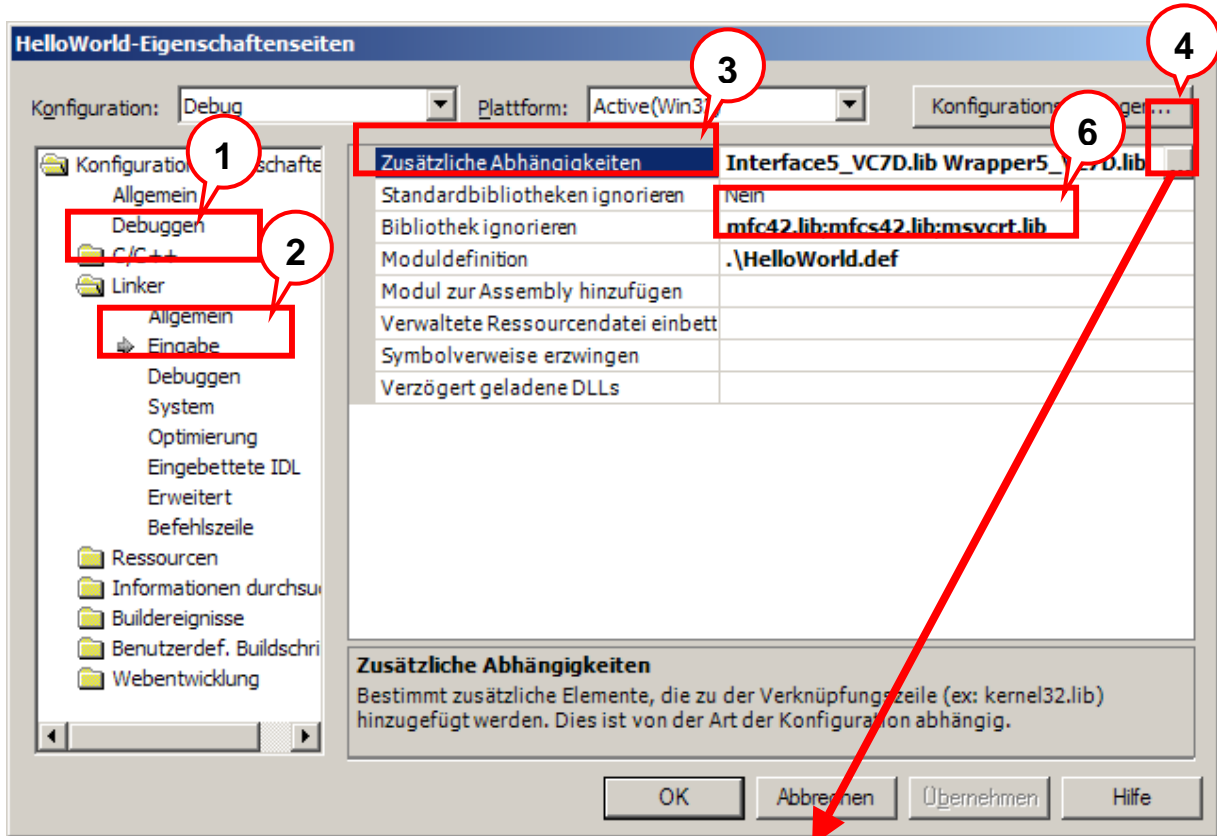


Abbildung 44: Eingaben für den Linker

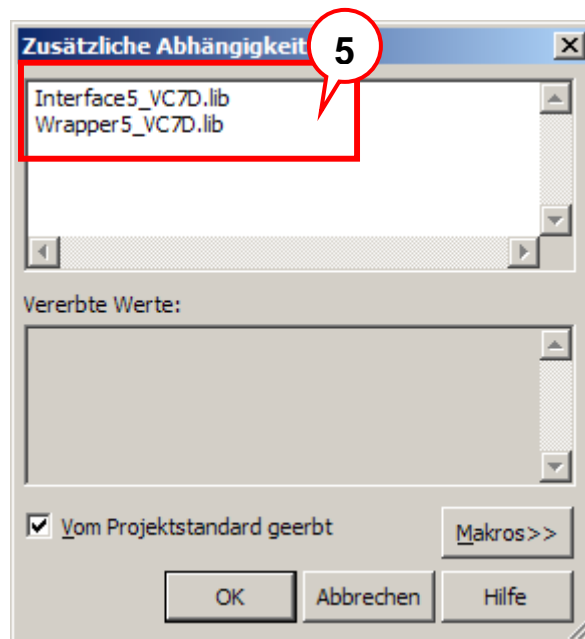


Abbildung 45: Eingaben für zusätzliche Abhängigkeiten

Stellen Sie unter *Konfiguration: Release* ein (1). Selektieren Sie im Ordner *Linker* (2) den Eintrag *Allgemein* (3). Fügen Sie im rechten Teil der Dialogbox unter *Zusätzliche Bibliotheksverzeichnisse* die Pfade zu den Bibliotheken *Wrapper5_VC7.lib* und *Interface5_VC7.lib* ein. Achten Sie darauf, dass Sie die Bibliotheken aus dem Ordner *VC7* verwenden, da diese mit der entsprechenden Entwicklungsumgebung erstellt wurden:

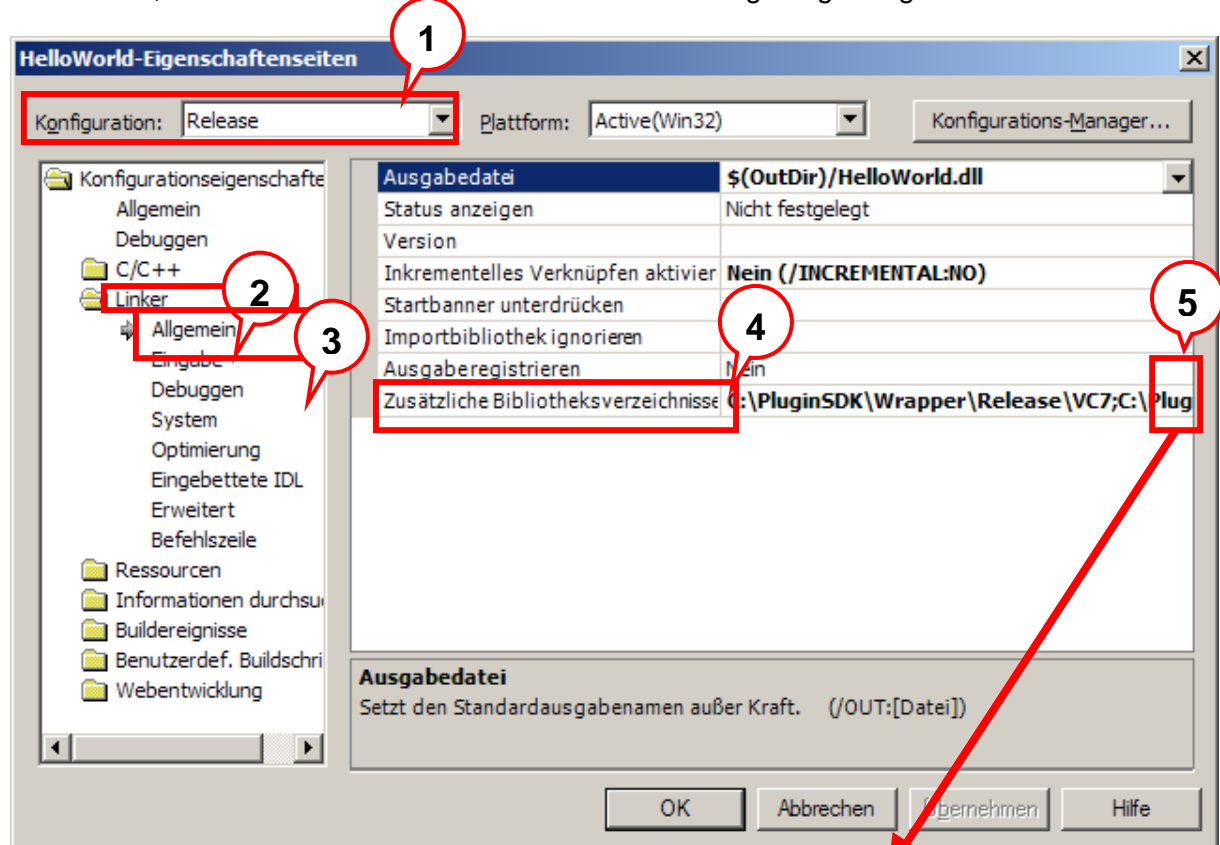


Abbildung 46: Zusätzliche Bibliotheksverzeichnisse für den Linker

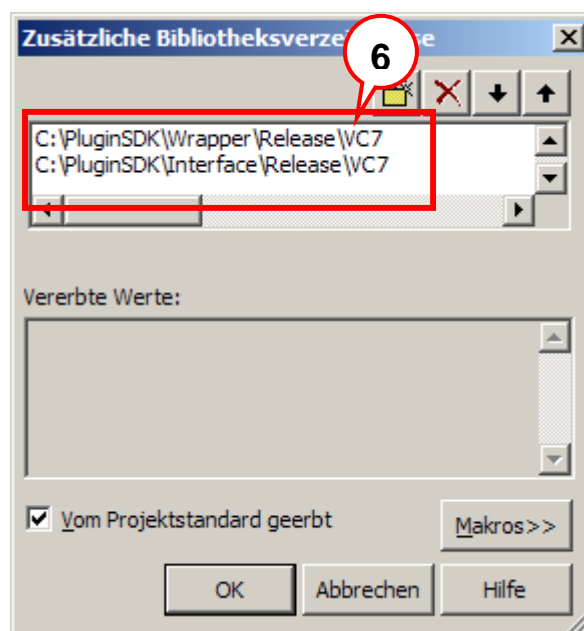


Abbildung 47: Editor für Zusätzliche Bibliotheksverzeichnisse

In dem Ordner *Linker* (1) klicken Sie auf *Eingabe* (2) und fügen unter *Zusätzliche Abhängigkeiten* (3) die Bibliotheken *Interface5_VC7.lib* und *Wrapper5_VC7.lib* hinzu (4),(5).

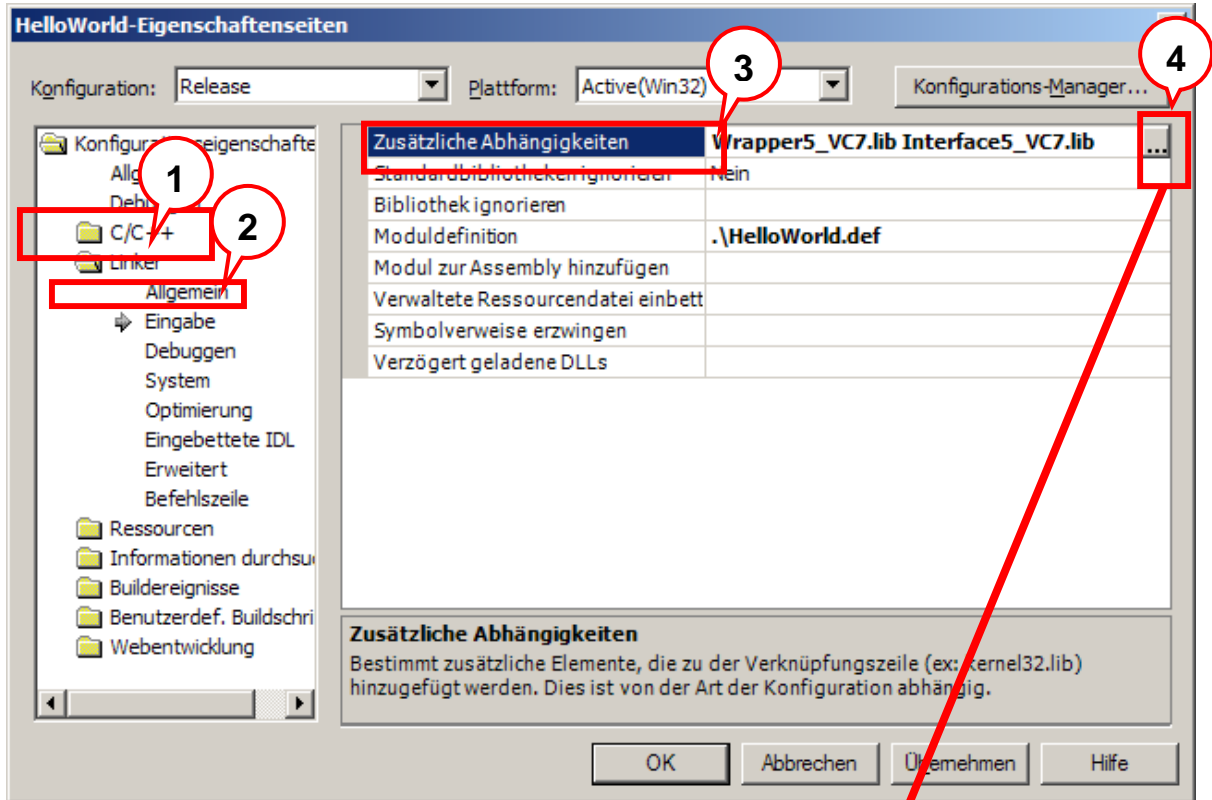


Abbildung 48: Zusätzliche Abhängigkeiten

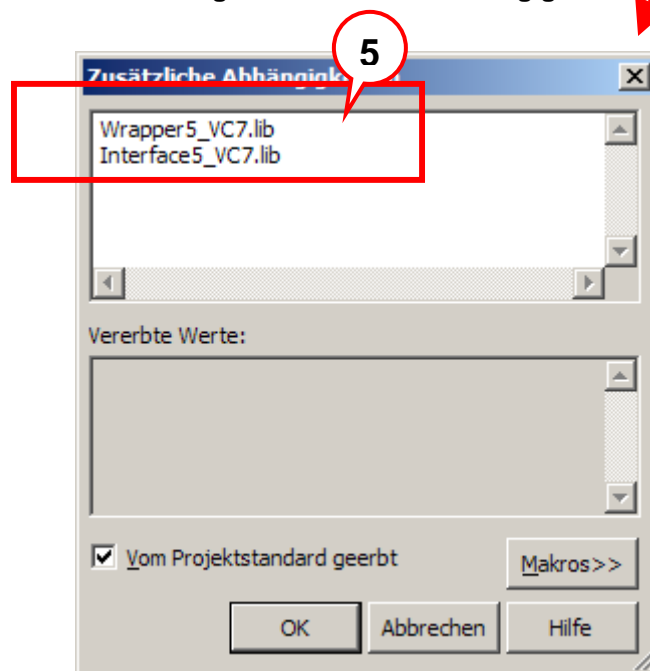


Abbildung 49: Editor für Zusätzliche Abhängigkeiten

Fügen Sie am Ende der Datei *HelloWorld.CPP* nachfolgenden Code ein: Dabei müssen Sie die hier verwendete CLSID bzw. GUID durch die aus Ihrer Lizenz-Datei ersetzen.

```
/*
*****
Ergänzungen für die Plugin - DLL
*****
#include "Wrapper.h"

static const GUID CLSID_GeoPlugin =
{ 0x9166900, 0x193D, 0x4B9A, { 0x9F, 0x7E, 0xED, 0x2F, 0xCB, 0x7D,
0x82, 0xAF } };

IMPLEMENT_PLUGIN_DLL_REG_TABLE( "{09166900-193d-4b9a-9f7e-ed2fcb7d82af}",
"HelloWorld")

IMPLEMENT_PLUGIN_MODULE_ROUTINES()

IMPLEMENT_PLUGIN_DLL_ENTRY_POINTS()
```

Fügen Sie die Datei mit Ihrem Lizenz-Schlüssel zu Ihrem Projekt hinzu. In unserem Beispiel ist es die Datei *PlugInKey.h*.

Nun fügen Sie die beiden Dateien *MyPlugin.h* und *MyPlugin.cpp* zu Ihrem Projekt hinzu.

MyPlugin.h:

```
// specification of the PlugIn class
class CMyPlugin : public CGeoPlugin
{
public:
void Init();
void Destroy();

CString GetLicenseKey();
};
```

MyPlugin.cpp:

```
// Implementation of the PlugIn class
#include "stdafx.h"

#include "Interface.h"
#include "Wrapper.h"

#include "MyPlugin.h"
#include "PlugInKey.h"

IMPLEMENT_GEOPLUGIN_CLASS(CMyPlugin)

void CMyPlugin::Init()
{
AppMessageBox("Hello World!", "MyPlugin");
}

void CMyPlugin::Destroy()
{
// Schreibe die aktuelle Zeit in das Geogrid INI file
CTime curTime = CTime::GetCurrentTime();
ProfileSetString ("MyPlugin", "ShutdownTime",
curTime.Format ("%d.%m.%Y %H:%M:%S"));
}
```

```
        AfxMessageBox( "Good bye Plugin" );
    }

CString CMyPlugin::GetLicenseKey()
{
    // Returns the license key for authorization of the plugin.
    // This method is called once before the plugin is initialised.
    // 'strLicenseKey' is the license key defined in
    // 'Geogrid_Plugin_Licens_Key.h'.
    // This include file has to be requested from EADS Deutschland GmbH,
    // Friedrichshafen.

    return strLicenseKey;
}
```

4.4 HelloWorld-Plugin with Graphic

In dem vorangegangenen Beispiel haben wir gesehen, wie die Verbindung zwischen einer Geogrid®-Applikation und einem Plugin hergestellt wird. Dabei wurde auf die Unterschiede eingegangen, die bei der Erstellung eines DLL-Plugins oder eines EXE-Plugins zu beachten sind. In diesem und allen weiteren Beispielen konzentrieren wir uns auf die Anwendung der zur Verfügung stehenden Methoden. Da die Implementierung unabhängig vom verwendeten Plugin-Typ ist, beschränken wir uns nun auf DLL-Plugins.

Wir wollen das „HelloWorld!“-Plugin so erweitern, dass der Text nicht mehr in einem Dialog, sondern in Form einer Textgraphik ausgegeben wird. Zu diesem Zweck definieren wir eine neue Methode *DrawTextGraphic*, die wir in unserer *Init*-Methode aufrufen. Nachfolgend ist zunächst der erweiterte Quellcode aufgelistet. In den anschließenden Kapiteln werden die einzelnen Methoden näher erläutert.

```
// specification of the PlugIn class
class CMyPlugin : public CGeoPlugin
{
public:

    /*****
    * Methods called by Geogrid(R)      *
    *****/
    void Init();
    void Destroy();

    CString GetLicenseKey();

    /*****
    * Actual Plugin methods          *
    *****/
    bool DrawTextGraphic();
    bool WriteToFile(CGPGraphic cgpGfx);
};

bool CMyPlugin::DrawTextGraphic()
{
    try
    {
        // Create Text graphic object
        CGPGraphicText cgpText(this);

        /*****
        * Adjust settings for text graphic
        *****/

        // Give the graphic object a name
        cgpText.SetObjName ("Text graphic");

        // Adjust the colour for the text
        cgpText.SetColor (255, 0, 128);

        // What is to be written
        cgpText.SetText ("Hello World!");

        // Select a Font
        cgpText.SetFont (CGF_TIMES);

        // How big is the text to be displayed in pixels
        cgpText.SetFontSize (60);
    }
}
```

```
// Shall the text be written horizontally (0) or with a different
// orientation
cgpText.SetOri (30);

// Text with white background
cgpText.SetBrush (GGB_SOLID);

// Get current document
CGPDocument cgpDoc;
cgpDoc.CreateFromActive (this);

// Get current position
GEODECCOORD gdcPos;
gdcPos = cgpDoc.GetPosition ();

// Set graphic position to the center of the visible map area
cgpText.SetCoord (1, &gdcPos);

// Finally draw the graphic on the map
cgpText.Attach (&cgpDoc);

return true;
}
catch (CGeoPlugInException except)
{
    except.ErrorHandler();
    return false;
}
}
```

4.4.1 Fehlerbehandlung

Methodenaufrufe, die zu einem Fehler führen, lösen eine Ausnahme (Exception) aus. Das bedeutet für den Plugin-Entwickler, dass alle Methodenaufrufe mit try und catch-Blöcken geklammert sein sollten.

```
try {
    // Your code, including plugin methods
}
catch (CGeoPlugInException except)
{
    except.ErrorHandler();
    return false;
}
```

Die Methode ErrorHandler() wertet die Fehlermeldung aus und zeigt einen Meldedialog mit der Fehlermeldung an.

Ohne eine entsprechende Ausnahmebehandlung, erscheint folgende Meldung:

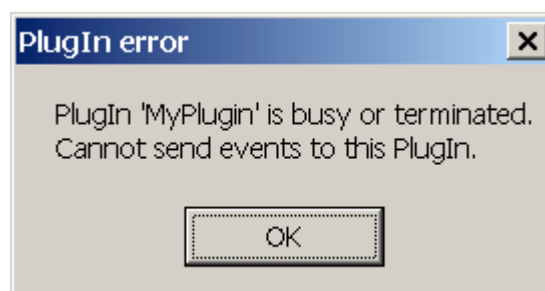


Abbildung 50: Fehlermeldung bei plugin-Exceptions

und die Verbindung zwischen Geogrid® und dem Plugin wird gelöst. Bei der Verwendung einer Plugin-DLL bedeutet das, dass Geogrid® neu gestartet werden muss, damit wieder eine Verbindung hergestellt wird. Im Falle eines EXE-Plugins muss sich das Plugin erneut bei Geogrid® anmelden.

4.4.2 Graphik erzeugen

Zuerst definieren wir den Graphiktyp *CGPGraphicText*. Dazu wird der Konstruktor mit einem Zeiger auf ein *CGeoPlugin*-Objekt aufgerufen. Da unsere Pluginklasse *CMyPlugin* von *CGeoPlugin* abgeleitet ist, können wir an dieser Stelle den this-Zeiger übergeben. In den allermeisten Fällen werden Objekte in einem Plugin auf diese Weise erzeugt. *CGeoPlugin* hat einen Zeiger auf die *IGeoApp* Schnittstelle, die unter anderem eine Methode zum Erzeugen von Graphikobjekten besitzt.

Im Prinzip ist damit bereits die Erzeugung einer Text-Graphik abgeschlossen. Standardmäßig hat eine Text-Graphik folgende Eigenschaften:

Schriftart:	Arial
Größe in Pixel:	20
Ausrichtung:	Horizontal
Hintergrund:	Weiß
Text:	„“ → Leerstring
Farbe:	Rot

Da der Text aus einem Leerstring besteht wird man mit den Standardeinstellungen auf der Karte nichts sehen. Also muss zumindest ein Text angegeben werden.

Ein zweiter wichtiger Punkt, der für alle Graphiken gilt, ist: damit eine Graphik auf der Karte gezeichnet werden kann, benötigt sie Koordinaten. Einpunkt-Graphiken, deren Position durch genau eine Koordinate definiert wird, benötigen eine Koordinate. Linien benötigen mindestens zwei Koordinaten, Flächen mindestens drei. Bei Einpunktgraphiken wird diese mit ihrem Mittelpunkt auf die Karte positioniert. Eine Ausnahme bilden taktische Zeichen. Hier wird die Stelle des Symbols, zur Positionierung verwendet, die im Symbolvorrat definiert wurde.

4.4.2.1 Graphikeinstellungen

Da wir nicht auf die Grundeinstellungen zurückgreifen wollen, nehmen wir noch einige Änderungen vor

1. Objektname

Mit `cgpText.SetObjName("Text graphic");` geben wir dem Graphikobjekt den Namen "Text graphic". Der Objektname kann auf Geogrid® -Seite im Tooltip zur Graphik angezeigt werden. In Lageprodukten ist er außerdem in der Baumansicht zu sehen.

2. Farbe

Die Farbe lässt sich mit `cgpText.SetColor(255, 0, 128);` einstellen. Die drei Parameter sind vom Typ Integer und entsprechen den Anteilen für Rot, Grün, und

Blau, aus denen sich die Graphikfarbe zusammensetzt. Es können Werte zwischen 0 und 255 angegeben werden.

3. Text

Der eigentliche Text wird mit `cgpText.SetText ("Hello World!");` gesetzt. Dieser Methode wird ein CString übergeben.

4. Schriftart

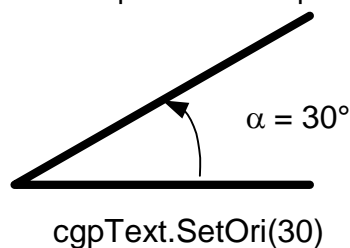
Für unseren Text können wir eine Schriftart auswählen. `cgpText.SetFont (CGF_TIMES);` Die einstellbaren Schriftarten finden Sie in der Hilfe, wenn Sie nach *GEOGRAPHICFONT* suchen.

5. Schriftgröße

Mit `cgpText.SetFontSize (60);` wird die Größe des Textes in Pixel angegeben.

6. Ausrichtung

Die Standardausrichtung der Graphik ist horizontal. Sie kann verändert werden. Der Wert, in `cgpText.SetOri (30);` entspricht dem Drehwinkel gegen den Uhrzeigersinn in Grad. Gedreht wird um den Mittelpunkt der Graphik.



Der Drehwinkel wird ausgehend von der Horizontalen gegen den Uhrzeigersinn gemessen. D.h. bei einem Winkel von 0° befindet sich die Textgraphik in der Waagerechten. Mit dem Aufruf `cgpText.SetOri(90)` steht die Textgraphik senkrecht und die Schrift verläuft von unten nach oben. Die Drehachse ist der Mittelpunkt der Graphik.

7. Hintergrund

Für Text-Graphiken kann eingestellt werden, ob die Schrift direkt auf die Karte gezeichnet oder mit einem Hintergrund versehen werden soll.

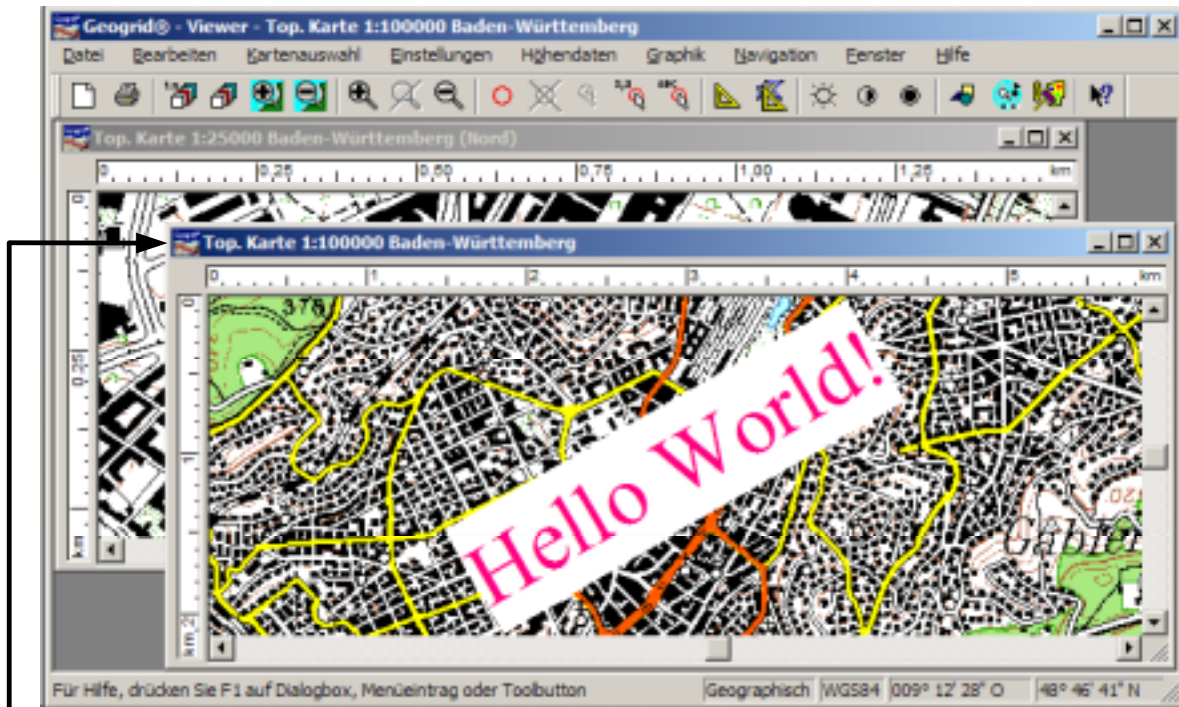
`cgpText.SetBrush (GGB_SOLID);` versieht den Text mit einem Hintergrund und hebt ihn so beim Zeichnen von der Karte ab.

`cgpText.SetBrush (GGB_HOLLOW);` beim Zeichnen wird der Text direkt auf die Karte gebracht.

In unserem Beispiel haben wir die Einstellung *GGB_SOLID* gewählt.

4.4.3 Dokument / Kartenfenster

Während Geogrid® die Anzeige mehrerer Fenster unterstützt, kann der Anwender immer nur mit einem Fenster zur gleichen Zeit arbeiten. Dieses Fenster wird als aktives Fenster bezeichnet. Das aktive Fenster befindet sich vor allen anderen Fenstern. Es lässt sich auch durch die Farbe der Titelzeile von den anderen Fenstern unterscheiden. Alle anderen Fenster sind inaktiv in Bezug auf die Anwender-Eingaben.



```
CGPDocument cgpDoc;  
//Get access to the currently active document  
cgpDoc.CreateFromActive(this);
```

Abbildung 51: Kartenfenster

Momentan erlaubt die Geogrid® Architektur je nach verwendeter Version eine eins-zu-eins oder eine eins-zu-zwei Beziehung zwischen Dokument und Ansicht. D.h. im ersten Fall, jedes Dokument besitzt genau eine Ansicht, nämlich das Kartenfenster. Im zweiten Fall besitzt jedes Dokument genau zwei Ansichten: Das Kartenfenster und den Baum.

Vergleichen wir Geogrid® mit Word, kann man folgende Analogien feststellen:

Word-Dokumente werden in Geogrid® durch die Overlays repräsentiert. Beim Öffnen einer Karte wird automatisch ein leeres Overlay angelegt. Was in Word dem eingegebenen Text entspricht sind in Geogrid® die gezeichneten Graphiken. Der weiße Dokument-Hintergrund entspricht der Karte.

Wir müssen der Graphik mitteilen, in welches Dokument sie gezeichnet werden soll. Das erfordert den Zugriff auf ein Dokument. Dazu wird zunächst ein Objekt vom Typ *CGPDocument* angelegt und anschließend die Verbindung zum aktiven Dokument in Geogrid® hergestellt.

```
// Get current document  
CGPDocument cgpDoc;  
cgpDoc.CreateFromActive (this);
```

Mit dem durch den Standardkonstruktor erzeugten Objekt können wir noch nichts anfangen. Hier werden lediglich Membervariablen initialisiert. Erst durch den Aufruf der Methode *CreateFromActive* wird das Dokumentobjekt nutzbar. Als Parameter erhält diese Methode wieder den Zeiger auf ein *CGeoPlugin* Objekt, über den wir Zugriff auf das aktive Geogrid®-Dokument erhalten.

4.4.3.1 Aktuelle Mittelpunktkoordinate des sichtbaren Bereichs

Damit die Graphik in unserem Beispiel immer im sichtbaren Bereich der Karte gezeichnet wird, ermitteln wir zunächst die Mittelpunktkoordinate für dieses Gebiet.

```
// Get current position
GEODECCOORD gdcPos;
cgpDoc.GetPosition (&gdcPos);
```

Koordinaten werden immer als Geographische Koordinaten (dezimal) mit dem Geodätischen Datum WGS84 definiert!

Hinter dem Datentyp GEODECCOORD verbirgt sich folgende Struktur:

```
typedef struct GEODECCOORD
{
    double x;          //!< x-Value of decimal coordinate
    double y;          //!< y-Value of decimal coordinate
    double height;     //!< Height at specified coordinate
} GEODECCOORD;
```

Sie wird uns im Zusammenhang mit Koordinaten immer wieder begegnen. Der Methode *cgpDoc.GetPosition* wird als Parameter ein Zeiger auf eine *GEODECCOORD* Struktur übergeben, in die die aktuelle Position geschrieben wird. Alternativ können Sie die Methode auch ohne Parameter aufrufen, und den Rückgabewert einer *GEODECCOORD* Struktur zuweisen:

```
// Get current position
GEODECCOORD gdcPos;
gdcPos = cgpDoc.GetPosition ();
```

Diese Koordinate weisen wir nun unserer Graphik zu

```
cgpText.SetCoord (1, &gdcPos);
```

SetCoord ist eine Methode die für alle Graphiktypen verwendet wird, also auch für Linien und Flächen, die durch mehr als eine Koordinate definiert werden. Als Parameter werden ihr die Anzahl der Koordinaten und ein Zeiger auf eine *GEODECCOORD* Struktur übergeben, die diese Koordinaten enthält.

Schließlich wollen wir die Graphik auf der Karte anzeigen. Das wird durch den Aufruf

```
cgpText.Attach (&cgpDoc);
```

erreicht. Der Methode *Attach* geben wir einen Zeiger auf das Dokument mit, in dem die Graphik gezeichnet werden soll.

4.4.4 Zugriff auf Graphik-Daten

Hier soll nun gezeigt werden, wie auf die Graphikdaten zugegriffen werden kann. Zu jeder Set-Methode existiert eine Get-Methode, mit der bestimmte Eigenschaften der Graphik erfragt werden können. In unserem Beispiel haben wir eine zweite Methode definiert, die die Eigenschaften einer Graphik ausliest und diese in eine Text-Datei schreibt. Ihr übergeben wir ein Objekt vom Typ *CGPGraphic*. Damit erreichen wir, dass sie auch für andere Graphik-

typen verwendet werden kann. *CGPGraphic* enthält Methoden, die allen Graphiken gemeinsam sind. Ausgeschlossen sind hier nur Overlays.

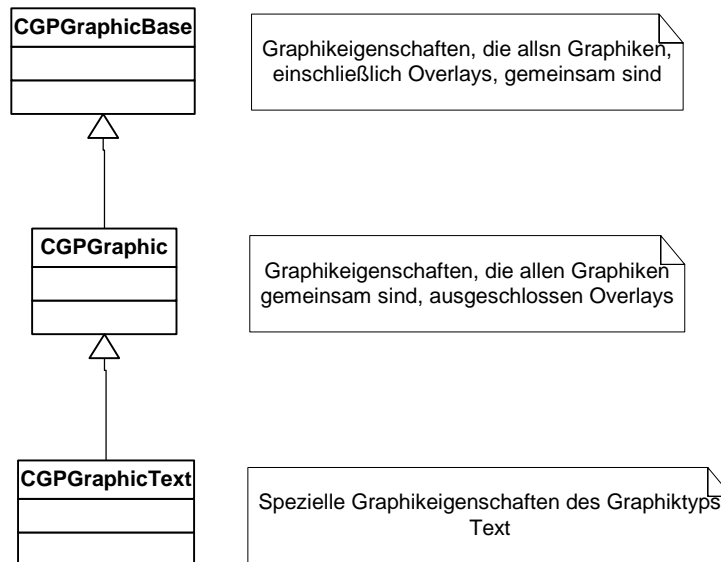


Abbildung 52: Ableitungshierarchie für CGPGraphicText

```

bool CMyPlugin::WriteToFile(CGPGraphic cgpGfx)
{
    GEOGRAPHICTYPE ggt;           // Takes the graphic type of the passed
                                // CGPGraphic - object
    CString strObjName;          // Gets the name of the Text object
    CString strText;             // Gets the actual text
    GEOBRUSH brush;              // Has the text a white background?
    int nRed, nGreen, nBlue;     // What is the colour of the text?
    GEODECCOORD gdcPos;         // The position of the text
    GEOGRAPHICFONT font;        // What kind of font is used?
    long lFontSize;              // The Size of the Text graphic
    long lOrientation;           // Orientation of the Text graphic in
                                // degree

    try{
        // Gets the type of graphic
        ggt = cgpGfx.GetType();
    }
    catch (CGeoPlugInException except) {
        except.ErrorHandler();
        return false;
    }

    switch(ggt) {
    case GGT_TEXT:
        {
            try{
                // Create a Text graphic from the passed graphic that means
                // an Interface pointer to a text object is requested
                CGPGraphicText cgpText(&cgpGfx);

                // Get the name of the Text object
                strObjName = cgpText.GetObjName();

                // Get the actual text
                strText = cgpText.GetText();
            }
        }
    }
}

```

```
// Has the text a white background?
brush = cgpText.GetBrush();

// What is the colour of the text?
cgpText.GetColor(&nRed, &nGreen, &nBlue);

// The position of the text
cgpText.GetCoord(1, &gdcPos);

// What kind of font is used?
font = cgpText.GetFont();

// The Size of the Text graphic
lFontSize = cgpText.GetFontSize();

// Orientation of the Text graphic
lOrientation = cgpText.GetOri();
}
catch (CGeoPlugInException except) {
    except.ErrorHandler();
    return false;
}

// File to serialize the graphic properties
CString strFile = "C:\\Temp\\Graphic.txt";

// Write the information into a text file
BOOL ret = WritePrivateProfileString
    ("TEXT", "Name", strObjName, strFile);

ret = WritePrivateProfileString
    ("TEXT", "Text", strText, strFile);

CString strBrush;
strBrush.Format("%d", brush);
ret = WritePrivateProfileString
    ("TEXT", "Brush", strBrush, strFile);

CString strRed, strGreen, strBlue;
strRed.Format("%d", nRed);
strGreen.Format("%d", nGreen);
strBlue.Format("%d", nBlue);
ret = WritePrivateProfileString
    ("TEXT", "Red", strRed, strFile);
ret = WritePrivateProfileString
    ("TEXT", "Green", strGreen, strFile);
ret = WritePrivateProfileString
    ("TEXT", "Blue", strBlue, strFile);

CString strXPos, strYPos, strHeight;
strXPos.Format("%f", gdcPos.x);
strYPos.Format("%f", gdcPos.y);
strHeight.Format("%f", gdcPos.height);
ret = WritePrivateProfileString
    ("TEXT", "X-Position", strXPos, strFile);
ret = WritePrivateProfileString
    ("TEXT", "Y-Position", strYPos, strFile);
ret = WritePrivateProfileString
    ("TEXT", "Height", strHeight, strFile);

CString strFont;
strFont.Format("%d", font);
ret = WritePrivateProfileString
    ("TEXT", "Font", strFont, strFile);

CString strSize;
```

```
        strSize.Format("%d", lFontSize);
        ret = WritePrivateProfileString
            ("TEXT", "Size", strSize, strFile);

        CString strOrientation;
        strOrientation.Format("%d", lOrientation);
        ret = WritePrivateProfileString
            ("TEXT", "Orientation", strOrientation, strFile);

        break;
    }
    default:
        break;
    }

    return true;
}
```

In der Methode `WriteToFile(CGPGraphic cgpGfx)` werden die Eigenschaften des als Parameter übergebenen Graphikobjekts ermittelt und anschließend in eine Textdatei geschrieben.

Mit dem ersten Aufruf `cgpGfx.GetType()` finden wir heraus, um welche Art von Graphik es sich handelt. Der Rückgabewert ist vom Typ `GEOGRAPHICTYPE`. Wenn Sie unter diesem Schlüsselwort in der Plugin HTML Dokumentation nachschlagen, finden Sie eine Liste der vorhandenen Graphiktypen.

Entsprechend ist es uns möglich ein konkretes Objekt von diesem Typ zu erzeugen. D.h. in dem Konstruktor der `CGPGraphicText`-Klasse wird mit Hilfe des übergebenen Graphik – Objekts ein Zeiger auf die Textgraphik angefordert. Dieser Zeiger erlaubt uns den Zugriff auf die Methoden des `CGPGraphicText`-Objekts. Der von uns verwendete Konstruktor erfordert einen Zeiger auf ein `CGPGraphicBase`-Objekt.

```
CGPGraphicText(CGPGraphicBase *pGraphicBase);
```

Da die Klasse `CGPGraphic` von `CGPGraphicBase` abgeleitet ist, können wir auch die Adresse unseres `CGPGraphic`-Objekts übergeben.

```
CGPGraphicText cgpText(&cgpGfx);
```

Anschließend können wir die Eigenschaften dieser Graphik erfragen.

```
// Get the name of the graphic object
CString strObjName = cgpText.GetObjName();

// Get the actual text
CString strText = cgpText.GetText();

// Does the text have a white background?
GEOBRUSH brush = cgpText.GetBrush();

// What ist he colour of the text?
int nRed, nGreen, nBlue;
cgpText.GetColor(&nRed, &nGreen, &nBlue);

// The text coordinate
GEODECCOORD gdcPos;
cgpText.GetCoord(1, &gdcPos);

// Which font is used for the text?
GEOGRAPHICFONT font = cgpText.GetFont();
```

```
// Get the text size in pixels?  
long lFontSize = cgpText.GetFontSize();  
  
// How many degrees is the graphic rotated counter clockwise?  
long lOrientation = cgpText.GetOri();
```

1. Objektname

Mit `cgpText.GetObjName()`; ermitteln wir den Namen des Graphikobjekts.

2. Text

Der eigentliche Text wird mit `cgpText.GetText()`; ermittelt.

3. Hintergrund

Mit `cgpText.GetBrush()`; wird ermittelt, ob der Text mit einer weißen Fläche hinterlegt ist. Für die Graphiktypen Rechteck, Kreis, Dreieck und Fläche liefert der Aufruf dieser Methode das Füllmuster des jeweiligen Graphik-Objekts!

4. Farbe

Die Farbe lässt sich mit `cgpText.GetColor(&nRed, &nGreen, &nBlue)`; auslesen. Die drei Parameter sind vom Typ Zeiger auf Integer und entsprechen den Anteilen für Rot, Grün, und Blau, aus denen sich die Graphikfarbe zusammensetzt. Es können Werte zwischen 0 und 255 angegeben werden. In unserem Beispiel übergeben wir die Adressen der Integer-Variablen.

5. Koordinate

Die geographische Position der Graphik erhalten wir durch den Aufruf von `cgpText.GetCoord(1, &gdcPos)`; . Die beiden Parameter sind zunächst die Anzahl der Koordinaten, die ermittelt werden sollen und an der zweiten Stelle ein Zeiger auf den bereitgestellten Speicherplatz für die Koordinaten. Diese kann mit dem Aufruf `int nCount = cgpText.GetCoordCount()`; ermittelt werden, falls sich die Anzahl der Koordinaten ändern kann.

6. Schriftart

Die Schriftart des Textes ermitteln wir mit `cgpText.SetFont()`;

7. Schriftgröße

Mit `cgpText.GetFontSize()`; wird die Größe des Textes in Pixel zurückgegeben.

8. Ausrichtung

`cgpText.GetOri ()`; liefert den Drehwinkel gegen den Uhrzeigersinn in Grad. Gedreht wird um den Mittelpunkt der Graphik.

4.4.5 Hello World mit Graphic-Events

Damit ein Plugin auf Aktionen in der Geogrid®-GUI reagieren kann, muss es darüber informiert werden. Diese Funktion übernehmen die sogenannten Events.

Damit ein Event auch bei dem Plugin ankommt muss dieses zunächst aktiviert werden. Das geschieht zweckmäßiger Weise in der Init-Methode mit dem Aufruf

```
SetEnabledEvents(GEO_EVENT ge);
```

Wobei der übergebene Parameter eine Kombination aus den in der nachfolgenden Tabelle aufgeführten Events sein kann:

Geo Event	Description
GE_NONE	All Events Dis-Abled
GE_WINDOW	Window events
GE_MAP	Map events
GE_MENU	Menu events
GE_MOUSE	Mouse events
GE_GRAPHIC	Graphic events
GE_SETTINGS	Settings events
GE_ALL	All Events En-Abled

Tabelle 1: Flags, über die gesteuert wird welche Events an das Plugin gesendet werden

z.B.: SetEnabledEvents(GE_MENU | GE_GRAPHIC); um Menü und Graphik-Events zu aktivieren.

Wir haben eine Graphik auf die Karte gezeichnet aber wir wollen sie nach wie vor unter unserer Kontrolle behalten. Dabei helfen uns die Graphik-Events. Wann immer eine Aktion ausgeführt wird, die in Zusammenhang mit Graphiken steht wird ein solches Event von Geogrid® verschickt. Dabei stehen folgende Nachrichten zur Verfügung:

Graphic Event	Bedeutung
GGE_CREATE	Es wurde eine neue Grapik gezeichnet.
GGE_CHANGE	Eine Graphik wurde verändert.
GGE_DEL	Eine Graphik wurde gelöscht.
GGE_MOVE	Eine Graphik wurde verschoben.
GGE_SEL	Eine Graphik ist selektiert worden.
GGE_UNSEL	Eine Graphik ist de-selektiert worden.
GGE_CREATE_OVL	Ein neues Overlay ist angelegt worden.
GGE_CHANGE_OVL	Ein Overlay ist modifiziert worden.
GGE_DEL_OVL	Ein Overlay ist gelöscht worden.
GGE_CREATE_GRP	Eine Gruppe ist neu angelegt worden.
GGE_CHANGE_GRP	Eine Gruppe ist modifiziert worden.
GGE_DEL_GRP	Eine Gruppe ist gelöscht worden.
GGE_ACTIVE_OVL_CHANGED	Das aktive Overlay hat sich geändert.
GGE_UNGROUP	Eine Gruppe ist aufgelöst worden.

Tabelle 2: Graphic-Events

Damit diese Events ausgewertet werden können muss man die Methode

```
GEOMENUEVENTRET OnGraphicEvent (IGeoGraphicEvent *pGraphicEvent);
```

implementieren. Sie enthält als Parameter einen Schnittstellenzeiger auf ein *IGeoGraphicEvent*. Mit Hilfe dieses Zeigers erzeugt man ein *CGPGraphicEvent*-Objekt.

```
CGPGraphicEvent cgpGraphicEvent (pIGraphicEvent);
```

Zuerst müssen wir nun herausfinden was für eine Nachricht gesendet wurde

```
GEOGRAPHICEVENT graphicEvent;
```

```
cgpGraphicEvent.GetEventType (&graphicEvent);
```

Nun können wir in einem case-Verteiler die Nachrichten verarbeiten. In unserem Programm sind exemplarisch einige Beispiele ausgeführt. Wird eine neue Graphik erzeugt, so werden deren Eigenschaften gespeichert, sofern es sich bei der Graphik um ein Dreieck handelt. Dazu ist die Methode *WriteToFile* entsprechend erweitert worden.

```
case GGE_CREATE:
{
    cgpGraphicEvent.GetEventCreate (&cgpGraphic);
    try{
        // Gets the type of graphic
        GEOGRAPHICTYPE ggt;
        ggt = cgpGraphic.GetType();

        // Save settings if it is a triangle
        if (ggt == GGT_TRIANGLE) {
            WriteToFile(cgpGraphic);
        }
    }
    catch (CGeoPluginException except) {
        except.ErrorHandler();
    }
    break;
}
```

Wenn die Nachricht, dass sich eine Graphik geändert hat, ankommt, wird in unserem Beispiel geprüft, ob es sich um die „Hello World“-Graphik handelt. Zu diesem Zweck haben wir eine Member-Variable zum Speichern einer Graphik ID angelegt.

```
GEOIDENTIFY m_gfxID; // Save Graphic ID of created graphic
```

Zu Beginn der Methode *DrawTextGraphic()* speichern wir die Graphic ID des Textes in dieser Variablen.

```
bool CMyPlugin::DrawTextGraphic()
{
    try
    {
        // Create Text graphic object
        CGPGraphicText cgpText(this);

        cgpText.GetId(&m_gfxID);
        ...
    }
}
```

Jede Graphik, die in Geogrid® gezeichnet wird, erhält eine ID, die während der Geogrid® Laufzeit eindeutig ist. Wir vergleichen die gespeichert ID mit der ID der geänderten Graphik. Handelt es sich um „Hello World“, werden die geänderten Daten in der Textdatei abgespeichert. Zum Vergleich der ID verwenden wir eine kleine Methode, da es sich dabei um ein Byte-Array handelt.

```
case GGE_CHANGE:
{
    CGPGraphic cgpGraphicAlt;
    cgpGraphicEvent.GetEventChange (&cgpGraphicAlt, &cgpGraphic);
```

```
// Get unique identifier of the graphic
GEOIDENTIFY graphicId;
cgpGraphic.GetId(&graphicId);

// If the text graphic has been changed, write changes to file
if (IsEqualGraphicId(m_gfxID, graphicId))
{
    WriteToFile(cgpGraphic);
}

break;
}
```

Mit der Methode *GetEventChange* erhalten wir Zugriff auf die Graphik vor und nach der Änderung. Damit haben wir z.B. die Möglichkeit zu untersuchen, welche Eigenschaften sich geändert haben.

Hat sich die Position unserer „Hello World“-Graphik geändert wollen wir diese Änderung ebenfalls abspeichern:

```
case GGE_MOVE:
{
    cgpGraphicEvent.GetEventMove (&cgpGraphic);

    // Get unique identifier of the graphic
    GEOIDENTIFY graphicId;
    cgpGraphic.GetId(&graphicId);

    // If the text graphic has been moved, write changes to file
    if (IsEqualGraphicId(m_gfxID, graphicId))
    {
        WriteToFile(cgpGraphic);
    }

    break;
}
```

4.5 Hello World mit Menü

Bisher haben wir alle Aktionen über die Methode *Init* gesteuert. Das ist natürlich sehr unflexibel und daher unbefriedigend. Wenn wir schon mit einem Programm arbeiten, das eine graphische Benutzeroberfläche besitzt, wäre es doch schön, wenn wir diese erweitern könnten. Das soll das Thema dieses Kapitels sein. Wir werden sehen wie das Geogrid®-Menü ergänzt werden kann, wie man ein Pop-up Menü erweitert oder ein neues Pop-up Menü anlegt und schließlich wie man eigene Toolbars erzeugen kann.

4.5.1 Ein Hauptmenü in die Geogrid®-Menüzeile einfügen

Für die Erzeugung und Bearbeitung von Menüs steht uns die Klasse *CGPMenu* zur Verfügung. Es bietet sich an, ein neues Menü in der *Init*-Methode unseres Plugins anzulegen, da zum Zeitpunkt des Aufrufs Geogrid® bereits hochgefahren ist.

Zunächst legen wir ein *CGPMenu* Objekt an. Damit wir auch im weiteren Verlauf des Programms immer wieder auf das Menü zugreifen können, verwenden wir für das Plugin-Menü eine Member-Variable.

```
// Eintrag in MyPlugin.h
private: CGPMenu *m_pcgpPluginMenu;

// Eintrag in der Init-Methode von CMyPlugin in MyPlugin.cpp
// Ein Menü-Objekt erzeugen
m_pcgpPluginMenu = new CGPMenu(this);
```

Nun können wir ein Menü anlagen:

```
// Ein Leeres Menü erzeugen
m_pcgpPluginMenu->Create();
```

Das Menü ist zu Beginn Leer. Menü-Einträge können mit den Methoden *AppendItem* oder *InsertItem* hinzugefügt werden. Für jeden Menüeintrag muss zuvor eine eindeutige Menü-ID definiert werden. In der Datei MyPlugin.h schreiben wir dazu

```
#define IDM_HELLO 1
#define IDM_UPDATE_GRAPHIC 2
#define IDM_UPDATE_FILE 3
```

Da wir ein neues Menü anlegen, kommen wir zunächst mit der Methode *AppendItem* aus. Dieser Methode werden drei Parameter übergeben. Ein CString-Objekt, welches den Menü-Text enthält. An zweiter Position übergeben wir die Menü-ID. Schließlich können wir noch eine Kurzbeschreibung hinzufügen, die in der Statuszeile angezeigt wird, wenn man mit der Maus auf diesen Menübefehl kommt. Mit den Befehlen *AppendSeparator* und *InsertSeparator* können wir die Menüeinträge durch das Einfügen von Separatoren gruppieren. *AppendSeparator* fügt einen Separator am Ende des Menüs ein, mit *InsertSeparator* fügt einen Separator vor der angegebenen Position ein.

```
// Das Menü mit Einträgen füllen
m_pcgpPluginMenu->AppendItem( "Hello World!",
                               IDM_HELLO,
                               "Writes the string Hello World! on the
                               map." );
```

```
m_pcgpPluginMenu->AppendSeparator();  
m_pcgpPluginMenu->AppendItem( "Update graphic position from file",  
                                IDM_UPDATE_GRAPHIC,  
                                "Updates the graphic from a file.");  
m_pcgpPluginMenu->AppendItem( "Save graphic properties in file",  
                                IDM_UPDATE_FILE,  
                                "Saves the graphic data into a file.");
```

Nachdem wir ein neues Menü angelegt und mit Einträgen gefüllt haben, können wir das Menü in das Geogrid®-Hauptmenü einhängen oder als Popup-Menü anzeigen.

Um das definierte Menü ins Hauptmenü einhängen zu können, müssen wir zunächst Zugriff auf das Geogrid® Menü haben. Dazu erzeugen wir ein neues Menü-Objekt und rufen die Methode *CreateFromHandle* auf. Der übergebene Parameter legt fest, auf welches Menü wir Zugriff haben wollen. Neben dem Hauptmenü können wir auch auf die Popup-Menüs zugreifen.

```
CGPMenu cgpMenuGeogrid(this);  
cgpMenuGeogrid.CreateFromHandle (GM_APP);
```

Nun fügen wir unser Plugin-Menü in die Menüzeile des Geogrid®-Menüs ein.

```
cgpMenuGeogrid.InsertMenu(m_pcgpPluginMenu, -1, "<< Hello World>>");
```

Der erste Parameter ist der Zeiger auf das Menü, welches wir einhängen möchten. An zweiter Stelle geben wir an, an welcher Position das Menü eingefügt werden soll. Hat der Wert ein negatives Vorzeichen, wird die Position von dem Ende der Menü-Zeile aus gezählt. In unserem Beispiel fügen wir das Menü am Ende der Menü-Zeile ein. Schließlich übergeben wir als dritten Parameter den Namen des Menüs.

Damit haben wir des Geogrid®-Menü erweitert, aber es passiert nichts, wenn wir einen der Menübefehle ausführen. Wenn wir auf das Auslösen eines Menübefehls reagieren wollen, muss uns darüber eine Nachricht erreichen. Die Events, die das Plugin erreichen sollen, müssen explizit eingeschaltet werden. Die Methode mit der dies geschieht haben wir bereits kennengelernt (siehe Kapitel 4.4.5).

```
// Very Important: You don't get any event if you don't have explicitly  
// enabled the  
// corresponding event!!!  
SetEnabledEvents(GE_GRAPHIC | GE_MENU | GE_MOUSE);
```

Die Nachrichten, die nun das Plugin erreichen können nun ausgewertet werden. Dazu muss die Methode

```
GEOMENUEVENTRET OnMenuEvent (IGeoMenuEvent *pIMenuEvent);
```

implementiert werden.

Unser Menü sieht jetzt wie folgt aus:

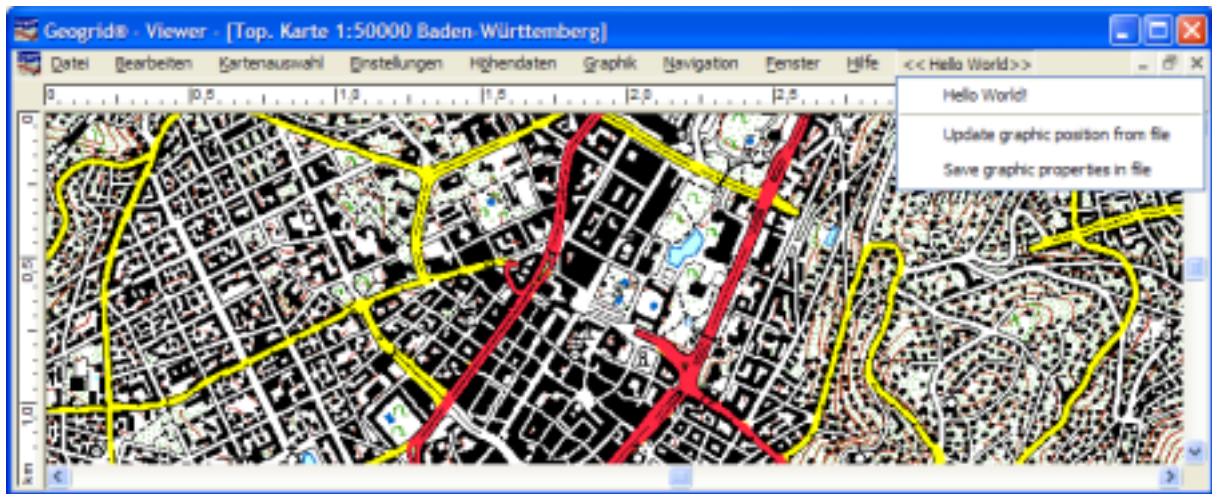


Abbildung 53: Plugin Menü

„Hello World!“ zeichnet die Text-Gaphic in die Mitte des sichtbaren Kartenfensters. Nach dem Zeichnen der Graphik wird dieser Menüeintrag deaktiviert. Wie machen wir das? Ganz einfach:

```
GEOMENUEVENTRET CMyPlugin::OnMenuEvent(IGeoMenuEvent *pIMenuEvent)
{
    // Define and initialize return value
    GEOMENUEVENTRET gmeReturn = GME_CONTINUE;

    // Create Menu-Event object from Interface pointer
    CGPMenuEvent cgpME(pIMenuEvent);

    // Which command has been sent?
    int nCmdID = cgpME.GetCmdId();

    switch(nCmdID) {
    case IDM_HELLO: {
        // Draw Hello World!
        bool fRet = DrawTextGraphic();

        // Disable menu item for drawing "Hello World!"
        m_pcgppPluginMenu->SetItemState(IDM_HELLO, GMS_DISABLED);

        // Event has been processed, the event doesn't need to be sent
        // to other
        // modules
        gmeReturn = GME_BREAK;

        break;
    }
    ...
}
}
```

Zunächst definieren wir einen Rückgabewert und initialisieren ihn mit dem Wert GME_CONTINUE. Dieser wird zurückgegeben, wenn der Menübefehl nicht abgearbeitet wurde. Damit kann eventuell ein anderes Modul auf dieses Ereignis reagieren. Im nächsten Schritt legen wir ein *CGPMenuEvent*-Objekt an und übergeben ihm den Schnittstellenzeiger aus dem *OnMenuEvent* Aufruf. Jetzt stellen wir fest, welche Nachricht gesendet wurde. Handelte es sich um IDM_HELLO, wird zunächst „Hello World! In die Mitte des sichtbaren

Kartenfensters gezeichnet. Damit es nur ein „HelloWorld!“ geben kann, deaktivieren wir den Menübefehleintrag. Nun setzen wir den Rückgabewert auf GME_BREAK und signalisieren damit, dass die Nachricht bearbeitet wurde und sie damit an kein anderes Modul weitergereicht werden soll.

Mit den anderen beiden Befehlen, *Update graphic position from file* und *Save graphic properties in file*, aktualisieren wir zum Einen die Graphikposition mit der Koordinate aus der Textdatei und schreiben zum Anderen die Graphikeigenschaften in eine Text-Datei. Verschieben wir also nach dem Speichern die Graphik auf der Karte, wird sie durch den Aufruf von *Update graphic position from file* an ihre Ausgangsposition zurückverschoben.

```
... case IDM_UPDATE_GRAPHIC:{  
  
    // File which contains the graphic properties  
    CString strFile = "C:\\Temp\\Graphic.txt";  
  
    // Write the information into a text file  
    GEODECCOORD gdcPos;  
    char chXPos[10];  
    char chYPos[10];  
  
    // Read coordinates from file  
    BOOL ret = GetPrivateProfileString(  
        "TEXT", "X-Position", "-200", chXPos, 10, strFile);  
    ret = GetPrivateProfileString(  
        "TEXT", "Y-Position", "-200", chYPos, 10, strFile);  
  
    float fltXPos, fltYPos;  
    sscanf(chXPos,"%f", &fltXPos);  
    sscanf(chYPos,"%f", &fltYPos);  
  
    // Check whether there are saved data, if not, do nothing  
    if ( (fltXPos != -200) && (fltYPos != -200) )  
    {  
        gdcPos.x = fltXPos;  
        gdcPos.y = fltYPos;  
  
        CGPGraphicOvl cgpOvl;  
        cgpDoc.GetActiveOvl(&cgpOvl);  
        CGPGraphic cgpGfx;  
  
        GEOIDENTIFY graphicId;  
  
        BOOL bRet = cgpOvl.GetFirstGraphic(&cgpGfx);  
  
        if (bRet == TRUE) {  
            do {  
                cgpGfx.GetId(&graphicId);  
  
                if (IsEqualGraphicId(m_gfxID, graphicId))  
                {  
                    CGPGraphicText cgpText(&cgpGfx);  
                    cgpText.SetCoord(1,&gdcPos);  
                    gmeReturn = GME_BREAK;  
                    break;  
                }  
            } while(cgpOvl.GetNextGraphic(&cgpGfx));  
        }  
    }  
  
    gmeReturn = GME_BREAK;  
}
```

```
        break;  
    }
```

In diesem Code-Beispiel läuft uns das erste Mal der Begriff Overlay über den Weg. Er kann zwei Bedeutungen haben. Zum Einen handelt es sich um eine Datei, in der Graphik-Informationen gespeichert werden. Diese Dateien besitzen die Endung .ovl. Zum anderen handelt es sich um „Folien“, die über der Karte liegen und auf denen Graphiken gezeichnet werden. Der im obigen Code-Beispiel fett gedruckte Teil bezieht sich auf die Bedeutung in letzterem Sinne.

Jedem Dokument können mehrere Overlays überlagert sein. Es ist immer nur ein Overlay aktiv. Zeichnet man Graphiken mit Funktionen der Geogrid® Benutzeroberfläche, werden sie dem aktiven Overlay zugeordnet. Nach dem Start von Geogrid® existiert nur ein Overlay, welches zugleich das aktive Overlay ist.

Mit

```
CGPGraphicOvl cgpOvl;  
cgpDoc.GetActiveOvl(&cgpOvl);
```

erhalten wir Zugriff auf das aktive Overlay. Nun holen wir uns nacheinander die in diesem Overlay enthaltenen Graphiken und vergleichen die Graphik ID mit der unserer Text-Graphik. Bei Übereinstimmung positionieren wir unsere Graphik auf die Koordinate, die zuvor aus der Datei Graphic.txt gelesen wurde. Auch hier wird am Ende der Rückgabewert auf GME_BREAK gesetzt, um zu signalisieren, dass der Menübefehl abgearbeitet wurde.

Mit *Save graphic properties in file* speichern wir die Graphikeigenschaften unserer Text-Graphik in eine Text-Datei.

```
case IDM_UPDATE_FILE: {
    CGPGraphicOvl cgpOvl;
    cgpDoc.GetActiveOvl(&cgpOvl);
    CGPGraphic cgpGfx;

    GEOIDENTIFY graphicId;

    BOOL bRet = cgpOvl.GetFirstGraphic(&cgpGfx);

    if (bRet == TRUE) {
        do {
            cgpGfx.GetId(&graphicId);

            if (IsEqualGraphicId(m_gfxID, graphicId))
            {
                CGPGraphicText cgpText(&cgpGfx);
                WriteToFile(cgpText);
                gmeReturn = GME_BREAK;
                break;
            }
        } while(cgpOvl.GetNextGraphic(&cgpGfx));
    }

    gmeReturn = GME_BREAK;
    break;
}
```

4.5.2 Popup Menüs

4.5.2.1 Popup Menüs erstellen und anzeigen

Alles was wir wissen müssen um ein Popup-Menü zu erstellen haben wir bereits im vorherigen Abschnitt kennen gelernt. Das von uns erstellte Menü kann ohne weitere Veränderung auch als Popup-Menü verwendet werden. Der einzige neue Befehl, der dazu notwendig ist heißt:

```
m_pcgppopupMenu->TrackPopupMenu2(TPM_LEFTALIGN, lXPos, lYPos);
```

Der erste Parameter legt zum Einen die Anzeigeposition des Popup-Menüs fest. Es gibt folgende Möglichkeiten:

Horizontale Ausrichtung

- **TPM_CENTERALIGN** Zentriert das Pop-up Menü horizontal relativ zur lXPos-Koordinate.
- **TPM_LEFTALIGN** Positioniert das Pop-up Menü so, dass seine linke Seite an der lXPos Koordinate ausgerichtet wird.

- **TPM_RIGHTALIGN** Positioniert das Pop-up Menü so dass seine rechte Seite an der IXPos Koordinate ausgerichtet wird.

Vertikale Ausrichtung:

- **TPM_VCENTERALIGN** Zentriert das Pop-up Menü vertikal relativ zur IYPos-Koordinate.
- **TPM_TOPALIGN** Positioniert das Pop-up Menü so, dass seine obere Seite an der IYPos Koordinate ausgerichtet wird.
- **TPM_BOTTOMALIGN** Positioniert das Pop-up Menü so dass seine untere Seite an der IYPos Koordinate ausgerichtet wird.

Für die Maustasten gibt es die beiden Flags:

- **TPM_LEFTBUTTON** Wenn dieses Flag gesetzt ist, kann ein Menübefehl nur mit der linken Maustaste ausgelöst werden.
- **TPM_RIGHTBUTTON** Wenn dieses Flag gesetzt ist, kann ein Menübefehl mit der linken oder rechten Maustaste ausgelöst werden.

Popup Menüs werden in der Regel nach Mausaktionen angezeigt (Doppelklick, Klick mit der rechten Maustaste, etc.). Da die Auswertung von Mausaktionen im nächsten Kapitel beschrieben wird, gehen wir hier nicht näher darauf ein.

4.5.2.2 Erweitern von Geogrid® Popup-Menüs

Wie bei der Erweiterung des Geogrid® Hauptmenüs, ist es zunächst erforderlich eine Verbindung zu dem Geogrid® Popup-Menü herzustellen, welches wir erweitern möchten. Zum Beispiel das Graphik Popup-Menü:

```
// Extend Geogrid Pop-up menu for Graphics
CGPMenu cgpPopupGfx(this);
cgpPopupGfx.CreateFromHandle(GM_POPUP_MAP_GRAPHIC);
```

Nun fügen wir die beiden Einträge zum Speichern der Graphikeigenschaften und zum Aktualisieren der Graphik-Position am oberen Ende des Graphik-Popup-Menüs ein und grenzen die beiden Einträge durch einen Separator ab.

```
cgpPopupGfx.InsertItem(0, "Update graphic position from file",
    IDM_UPDATE_GRAPHIC2, "Updates the graphic from a file.");
cgpPopupGfx.InsertItem(1, "Save graphic properties in file",
    IDM_UPDATE_FILE2, "Saves the graphic data into a file.");
cgpPopupGfx.InsertSeparator(2);
```

Diese Programmzeilen werden in unserem Beispiel nach dem Zeichnen der Graphik ausgeführt.

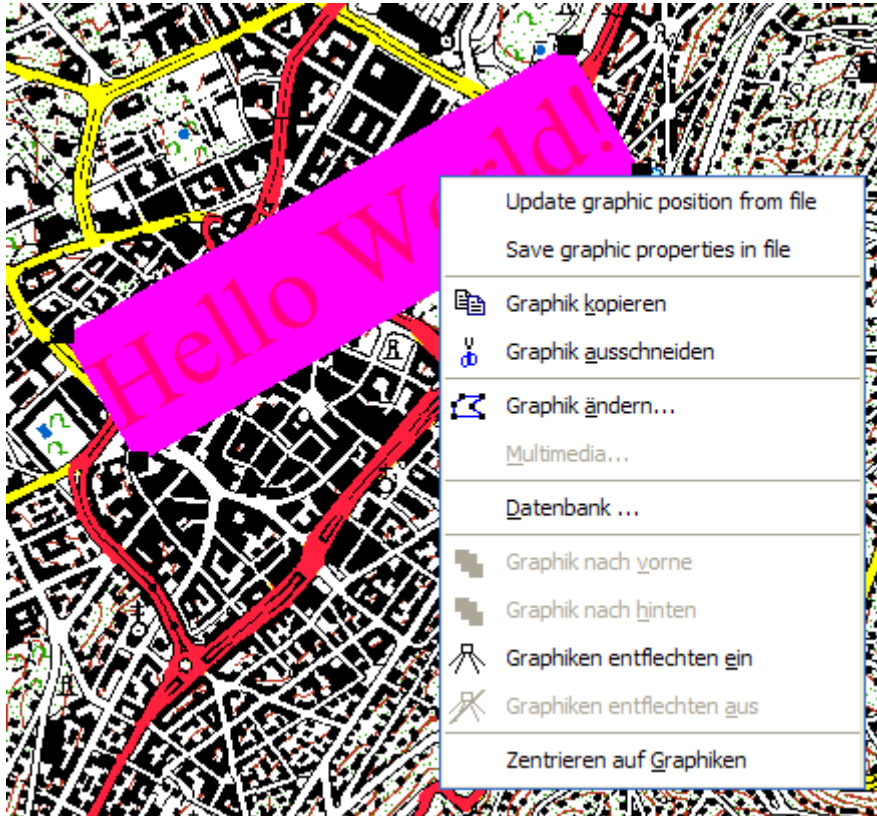


Abbildung 54: Erweiterung des Geogrid Graphik-Popup-Menüs durch das Plugin

Wollen wir die hinzugefügten Menüeinträge wieder entfernen, so erreichen wir das mit dem Aufruf von:

```
cgpPopupGfx.DeleteItem(IDM_UPDATE_GRAPHIC2);  
cgpPopupGfx.DeleteItem(IDM_UPDATE_FILE2);
```

Das geschieht in unserem Beispielprogramm, wenn wir die "Hello World!" Graphik löschen, da die beiden Menüeinträge dann keinen Sinn mehr machen.

4.5.2.3 Untermenüs

Neben dem Erstellen von Menüs haben wir noch die Möglichkeit in ein Plugin- oder Geogrid®-Menü Untermenüs einzufügen. Zunächst definieren wir das Untermenü:

```
// Eintrag im CMyPlugin.h  
private:  
CGPMenu *m_pcgpSubMenu;  
  
// Eintrag in der Init-Methode von CMyPlugin in der Datei CMyPlugin.cpp  
// Ein Menü-Objekt erzeugen  
  
m_pcgpSubMenu = new CGPMenu (this);  
m_pcgpSubMenu->Create();  
m_pcgpSubMenu->AppendItem ("To Reference-Point", IDM_CENTER_REFPOS,  
                           "Positions the map to the reference point!");  
m_pcgpSubMenu->AppendItem ("To Hello World", IDM_CENTER,  
                           "Positions the map to ,Hello World!'");  
  
m_pcgpSubMenu->SetItemState(IDM_CENTER, GMS_DISABLED);
```

Die Schritte bis hierher waren uns schon bekannt. Neu ist das Ankoppeln des Untermenüs an ein bestehendes Pluginmenü. Dazu steht uns die Member-Funktion *AppendMenu* zur Verfügung. Damit schicken wir eine Nachricht an das Plugin-Menü-Objekt, dass wir das Untermenü *m_pcgSubMenu* am Ende einfügen möchten und der Eintrag mit „Center“ bezeichnet werden soll.

```
// Untermenü an Pluginmenü anhängen  
m_pcgPluginMenu->AppendMenu (m_pcgSubMenu, "Center");
```

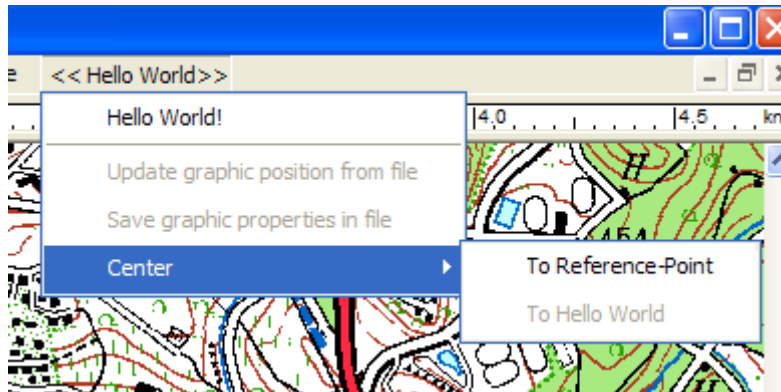


Abbildung 55: Plugin-Menü mit Sub-Menü

Zur Verarbeitung des Menübefehls „To Reference-Point“ verwenden wir eine Methode, mit der wir Geogrid®-Menübefehle aufrufen können. Sie ist in der *CGeoPlugin*-Klasse enthalten und kann daher in unserem Beispiel aufgerufen werden ohne ein neues Objekt anlegen zu müssen. Da dieser Aufruf nur Sinn macht, wenn ein Referenz-Punkt vorhanden ist, prüfen wir das vorher nach und geben ggf. eine Meldung aus.

```
// Center to Reference point  
BOOL bRef = cgpDoc.IsRefPointSet();  
if(bRef ==TRUE) {  
    AppSendWmCommand(CGP_IDM_REF_POSTO);  
}  
else {  
    AppMessageBox( "No Reference-Point set!",  
                  "Center to Reference point");  
}
```

Mit dem Menübefehl „To Hello World“ positionieren wir die Karte auf die Text-Graphik. Dieser Menübefehl wird erst durch das Einfügen des Hello World! Strings, d.h. nach Ausführung des Befehls *Hello World!* aktiv.

```
// Center to Hello World!  
case IDM_CENTER:  
{  
    CGPGraphicOvl cgpOvl;  
    cgpDoc.GetActiveOvl(&cgpOvl);  
    CGPGraphic cgpGfx;  
  
    GEOIDENTIFY graphicId;  
  
    BOOL bRet = cgpOvl.GetFirstGraphic(&cgpGfx);  
  
    if (bRet == TRUE) {  
        do {
```

```
        cgpGfx.GetId(&graphicId);

        if (IsEqualGraphicId(m_gfxID, graphicId))
        {
            CGPGraphicText cgpText(&cgpGfx);
            GEODECCOORD gdcPos;
            cgpText.GetCoord(1, &gdcPos);
            cgpDoc.SetPosition(gdcPos);
            gmeReturn = GME_BREAK;
            break;
        }
    } while(cgpOvl.GetNextGraphic(&cgpGfx));
}

gmeReturn = GME_BREAK;
break;
}
```

4.5.2.4 Toolbars

Toolbars bieten dem Benutzer eine einfache und schnelle Möglichkeit, um Menübefehle aufzurufen. In der Regel werden sie zusätzlich zu Menübefehlen angeboten. Wollen Sie eine Toolbar ohne zugehöriges Menü anbieten, so müssen Sie trotzdem ein CGPMenu-Objekt mit den zugehörigen Einträgen definieren. Sie hängen es allerdings nicht in das Geogrid®-Menü ein, so dass es nirgendwo sichtbar wird. Das Menü ist aber notwendig für die Behandlung der Befehle (MenuEvents)

Wie Menüs erstellt werden haben wir bereits gesehen. Hier nun die notwendigen Aufrufe, um eine Toolbar zu erzeugen.

```
Include-Datei CMyPlugin.h:
CGPToolbar *m_pToolbar;

// Here the Command IDs are from the Sub menu.
try {
    m_pToolbar = new CGPToolbar(this /*,1*/); // 1 is default
                                              // and can be omitted
                                              // 1 → TOP

    m_pToolbar->SetName("Center");
    m_pToolbar->AddUserButton("Hello.bmp",
                             IDM_CENTER,
                             "Center to Hello World.",
                             0);


    //Insert Seperator
    m_pToolbar->AddGeogridButton(0,-1);


    m_pToolbar->AddGeogridButton(CGP_IDM_REF_POSTO,-1);
}
catch (CGeoPlugInException e)
{
    e.ErrorHandler();
}
```

Wir instanziierten zunächst unsere Member-Variable. Dabei können wir die Andockposition angeben (TOP = 1, BOTTOM = 2, LEFT = 3, RIGHT = 4). Im nächsten Schritt geben wir unserem Toolbar einen Namen. Unter diesem Namen erscheint er auch im Dialog Fenstereinstellungen (Menü Einstellungen / Fenstereinstellungen). Schließlich fügen wir die Buttons hinzu. Wir können eigene Buttons verwenden, wobei wir den genauen Pfad angeben müssen. Als dritten Parameter übergeben wir einen kurzen Tooltip-Text. Außerdem können Geogrid®-Buttons verwendet werden. Dazu muss nur die ID des Menübefehls und die

Position angegeben werden. Dabei bedeutet Position am Ende oder am Anfang der Toolbar Leiste. Zum Einfügen eines Separators verwenden wir die Methode `AddGeogridButton` mit der ID 0. Die Zustände (`grayed`, `checked`, etc.) eines Toolbar-Buttons werden über das zugehörige Menü gesteuert. In unserem Fall sind die beiden Buttons nach dem

Programmstart zunächst `grayed` . Mit dem Hinzufügen bzw. Entfernen eines

Referenzpunktes ändert sich das Aussehen des ersten Buttons . Der Zustand des zweiten Buttons ändert sich in Abhängigkeit davon, ob „Hello World!“ eingefügt oder gelöscht

wurde . Bei Ausführung des Befehls *Hello World!* wird unter anderen der Menüeintrag „Center to Hello World“ aktiviert. Da dieser mit dem zweiten Button unseres Toolbars verknüpft ist wirkt sich das Aktivieren auch auf den Zustand dieses Buttons aus.

```
m_pcgppSubMenu->SetItemState(IDM_CENTER, GMS_ENABLED);
```

Beim Löschen einer Graphik wird die Nachricht `GGE_DEL` an das Plugin geschickt. Beim Auswerten dieser Nachricht wird geprüft, ob es sich bei der gelöschten Graphik um die „Hello World!“ Graphik handelt. Ist dies der Fall, so wird der Menüeintrag zum Positionieren der Karte auf „Hello World“ und der zugehörige Button `grayed`.

```
cgpGraphicEvent.GetEventDelete(&Graphic);
GEOIDENTIFY graphicId;
Graphic.GetId(&graphicId);

if (IsEqualGraphicId(m_gfxID, graphicId))
{
    // Hello World! has been deleted, we can enable the "Hello World"
    // Menu again.
    m_pcgppPluginMenu->SetItemState(IDM_HELLO, GMS_ENABLED);
    m_pcgppPluginMenu->SetItemState(IDM_UPDATE_GRAPHIC, GMS_GRAYED);
    m_pcgppPluginMenu->SetItemState(IDM_UPDATE_FILE, GMS_GRAYED);

    m_pcgppSubMenu->SetItemState(IDM_CENTER, GMS_GRAYED);

    // Delete extensions from the Geogrid Pop-up menu for Graphics
    CGPMenu cgpPopupGfx(this);
    cgpPopupGfx.CreateFromHandle(GM_POPUP_MAP_GRAPHIC);

    cgpPopupGfx.DeleteItem(IDM_UPDATE_GRAPHIC2);
    cgpPopupGfx.DeleteItem(IDM_UPDATE_FILE2);
}
```

4.5.2.5 Was ist noch möglich?

Geogrid®-Menüeinträge löschen

```
// Menuhandle für Applikationsmenü holen
CGPMenu menuGeogrid (m_GeoApp);

// Menü-Eintrag "Kreis" vom Menü "Graphik" löschen
menuGeogrid.DeleteAppMenuItem (CGP_IDM_BEAR_CIRC);

// Auch wenn ein Eintrag im Menü gelöscht wurde, kann der Befehl
trotzdem
// ausgeführt werden!
m_GeoApp->AppSendWmCommand (CGP_IDM_BEAR_CIRC);
```

Geogrid®-Menüs löschen

```
// Menuhandle für Applikationsmenü holen
CGPMenu menuGeogrid (m_GeoApp);
menuGeogrid.CreateFromHandle (GM_APP);

// Menü „Höhendaten“ löschen
menuGeogrid.DeleteAppMenuItem (GMI_ELEVATION);
```

Ein- und Ausblenden von Fensterelementen

Mit den nachfolgenden Programmzeilen wird z.B. das Geogrid®-Menü abwechselnd ein- und ausgeblendet.

```
CGPAppSettings cgpSettings (this);
unsigned long ulWindowSettings = cgpSettings.GetWindowSettings();
if( ulWindowSettings & WS_MENU) {
    ulWindowSettings = ulWindowSettings & ~WS_MENU;
}
else {
    ulWindowSettings = ulWindowSettings | WS_MENU;
}

cgpSettings.SetWindowSettings (ulWindowSettings);
```

Die folgenden Flags können mit den Methoden SetWindowSettings / GetWindowSettings verwendet:

Window Settings Flags	Description
WS_MENU	main menu
WS_TITLE	window title
WS_STATUSBAR	status bar
WS_SCROLLBAR	scrollbars
WS_TREEVIEW	tree view
WS_OVERLAY_MANAGER	overlay management dialog
WS_MIL_SYM_CLIPBOARD	clipboard for military symbols

Tabelle 3: Flags zum Steuern der fenstereinstellungen

Für Toolbars verwenden wir die Methoden SetToolbars / GetToolbars mit denen die nachfolgenden Flags verwendet werden können:

Toolbar Flags	Description
TB_TOOLBAR_MAP	toolbar map
TB_TOOLBAR_GRAPHIC	toolbar graphic
TB_TOOLBAR_ROUTE	toolbar route
TB_TOOLBAR_FORMAT	toolbar format
TB_TOOLBAR_SITUATION	toolbar situation
TB_SCALE	scale
TB_RULER	ruler

Tabelle 4: Flags zum Steuern der Toolbars sowie Maßstabsbalken und Lineal

4.5.2.6 Was geht nicht?

Gelöschte Menüeinträge können nicht wieder eingefügt werden!
Gelöschte Menüs können nicht wieder eingefügt werden!

Haupt-Menüeinträge können nicht gegrayed werden.

4.6 Modes

Modes werden dazu verwendet Mouse-Events zu kanalisieren. Wenn man z.B. eine Graphik zeichnen möchte, kann dazu ein Einfügemodus aktiviert werden. Solange dieser aktiv ist können Graphiken eingefügt werden. Zum Beenden dieses Modus wird mit der rechten Maustaste geklickt. Normalerweise würde damit ein Geogrid®-Popup-Menü aktiviert. Durch den gesetzten Mode gelangen Mouse-Events nur noch an das Plugin. Die Nachricht „rechte Maustaste gedrückt“ kann nun ausgewertet und der Modus beendet werden. Da es unterschiedliche Modi geben kann, ist es möglich zuerst den aktiven Mode abzufragen.

In diesem Beispiel zeigen wir die Verwendung von Modes. Die Graphik „Hello World!“ wird mit der Maus eingefügt und an die Klickposition platziert. Dazu wird beim Aufruf des Menübefehls IDM_HELLO zunächst nur ein Mode gesetzt und die Cursorform geändert:

```
GEOMENUEVENTRET CMyPlugin::OnMenuEvent(IGeoMenuEvent *pIMenuEvent)
{
    GEOMENUEVENTRET gmeReturn = GME_CONTINUE;
    CGPMenuEvent cgpME(pIMenuEvent);

    int nCmdID = cgpME.GetCmdId();

    switch(nCmdID) {
    case IDM_HELLO: {
        GEOAPPMODE mode = this->AppGetMode();
        if (mode == GMN_NOMODE)
        {
            m_gamInsertGraphic = GMN_GEOGRIDMODE + 1;
            this->AppSetMode(m_gamInsertGraphic);

            this->AppSetCursor(GCS_CROSS);

            gmeReturn = GME_BREAK;
        }
        break;
    }
}
...
```

Erst bei einem linken Mausklick in die Karte wird bei der Auswertung des Maus-Events „linke Maustaste wurde gedrückt“ die Graphik gezeichnet. Die Methode *DrawTextGraphic* wurde dazu um einen Parameter für die Koordinate erweitert.

```
GEOMENUEVENTRET CMyPlugin::OnMouseEvent(IGeoMouseEvent *pIGeoMouseEvent)
{
    GEOMENUEVENTRET gmeReturn = GME_CONTINUE;
    CGPMouseEvent cgpMouseEvent(pIGeoMouseEvent);

    GEOMOUSEEVENT gmeType = cgpMouseEvent.GetEventType();

    switch(gmeType) {
    case GME_LBUTTONDOWN: {
        GEOAPPMODE mode = this->AppGetMode();

        if (mode == m_gamInsertGraphic) {
            // Get click position of mouse
            GEODECCOORD gdcMousePos[1];
            cgpMouseEvent.GetCoord(gdcMousePos);

            bool fRet = DrawTextGraphic(gdcMousePos);
            m_pMenuHello->SetItemState(IDM_HELLO, GMS_GRAYED);
            m_pMenuHello->SetItemState(IDM_UPDATE_GRAPHIC, GMS_ENABLED);
            m_pMenuHello->SetItemState(IDM_UPDATE_FILE, GMS_ENABLED);
            m_pMenuHello->SetItemState(IDM_POSITION_GRAPHIC, GMS_ENABLED);

            this->AppSetMode(GMN_NOMODE);
            this->AppSetCursor(GCS_DEFAULT);
        }
    }
}
```

Nachdem das Event *GME_LBUTTONDOWN* identifiziert worden ist wird geprüft, welcher Modus gesetzt ist. Ist es der aus dem Aufruf des Menübefehls „Hello“ holen wir uns die Koordinate der Klickposition und rufen *DrawTextGraphic* auf. Anschließend werden die Menüzustände geändert und sowohl der Modus als auch die Cursorform zurückgesetzt.