



Die VC++-Projektarten

# Immer der richtige Typ

Wenn Sie ein **neues Projekt** mit VC++ anlegen, können Sie aus einer **Vielzahl von Projekten** den **richtigen Typ** auswählen. Doch **welcher** ist wofür gut?

THOMAS WÖLFER

**W**as tun die unterschiedlichen Konfigurationen der Projekte? Wie unterscheidet sich zum Beispiel die "Debug"-Konfiguration von der Release-Variante? In diesem Beitrag erfahren Sie all dies und mehr, rund um VC++-Projekte und Konfigurationen.

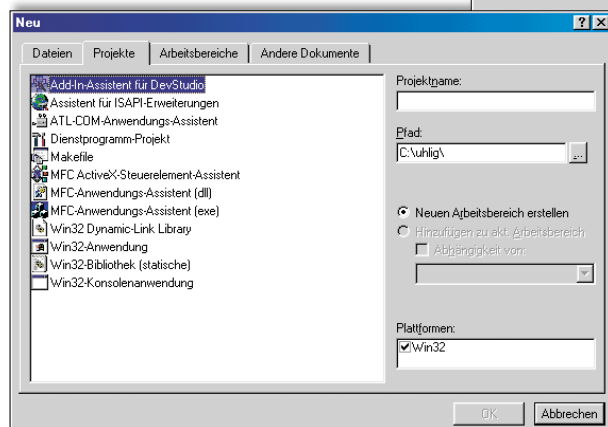
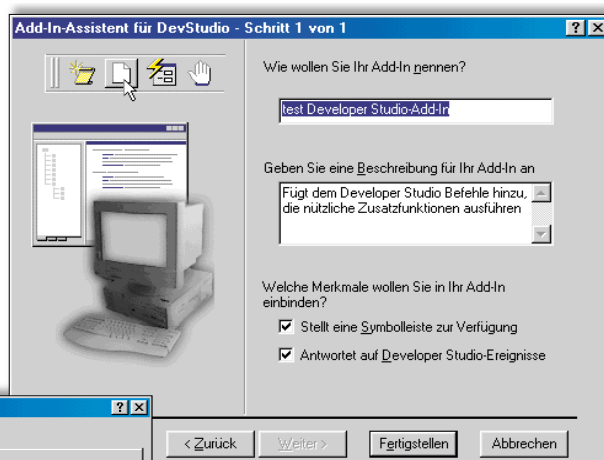
Projekte und Konfigurationen sind eigentlich zwei sehr unterschiedliche Dinge und das erkennt man daran, dass ein Projekt immer eine oder mehrere Konfigurationen enthalten kann, aber das dies nicht umgekehrt möglich ist. Daher werden nun zunächst die einzelnen Projektarten und dann die unterschiedlichen Konfigurationen erläutert.

nach noch einige Parameter in einen Assistenten eingegeben werden. Ein Projekt beschreibt dabei im Wesentlichen das Ziel der Software-Entwicklung: Was soll eigentlich am Ende für ein Produkt herauskommen? Das ist deshalb rele-

lerdings stellen die Assistenten immer gleich einen mehr oder minder sinnvollen Satz an Dateien und Compiler-Einstellungen zur Verfügung. Die müssen meist nur in Spezialfällen verändert werden.

Folgende Projektarten stehen zur Verfügung:

- Add-In-Assistent für DevStudio
- Assistent für ISAPI-Erweiterungen
- ATL-COM-Anwendungs-Assistent
- Dienstprogramm-Projekt
- Makefile
- MFC-ActiveX-Steuerelement-Assistent
- MFC-Anwendungs-Assistent
- Win32 Dynamic-Link Library
- Win32-Anwendung
- Win32-Bibliothek (statische)
- Win32-Konsolenanwendung



**VC++ BIETET EINE GANZE REIHE** von unterschiedlichen Projektarten an.

Ein Projekt legen Sie, wie im Laufe der Beispiele in diesem Sonderheft schon des Öfteren geschehen, mit dem **Neu**-Befehl an. Daraufhin erscheint der Dialog, aus dem Sie eine der Projektarten auswählen können. Je nach Projektart müssen da-

**DER ASSISTENT FÜR ADD-INS** besteht nur aus einem Schritt, in dem Sie im Wesentlichen die für das Add-In zu verwendenden Texte angeben können. Diese können Sie aber später noch ändern.

vant, weil dabei auf mehrere Dinge geachtet werden muss. Die wichtigsten dabei sind die folgenden:

- Welche Dateien und Dateitypen müssen an einem solchen Projekt beteiligt sein?
- Welche Optionen für Compiler, Linker und andere Tools sind notwendig?

Natürlich sind beide Einstellungen nachträglich von Hand veränderbar, al-

## Der Add-In-Assistent für DevStudio

Die VC++-IDE erhält immer wieder verschiedene Bezeichnungen. Das liegt unter anderem daran, dass sich bei jeder Version das Marketing neue Namen ausdenkt, aber auch daran, dass die gleiche IDE für verschiedene Programme verwendet wird. So ist die VC++ in der "großen" Version auch die Entwicklungsumgebung für andere Sprachen wie zum Beispiel für den Basic-Dialekt in ASP. Und bei dieser Projektart ist mit "DevStudio" das "Developer Studio" gemeint: die VC++-IDE.



Die IDE ist ungeheuer flexibel und kann auf verschiedene Arten um neue Funktionen erweitert werden. Im einfachsten Fall können Sie zum Beispiel Tastaturanschläge aufzeichnen und als Makro ablegen. Es gibt aber auch eine vollständige Macro-Entwicklungsumgebung, mit der Sie Macros programmieren können. Diese Macros werden in einem Visual-Basic-Dialekt programmiert. Wenn das nicht ausreicht, der kann auch eigene Funktionen in C++ programmieren und diese eigenen Funktionen dann in die Entwicklungs-

auf das komplette Objektmodell des Visual Studio: Sie können alle internen Funktionen der IDE aufrufen und verwenden und haben natürlich auch Zugriff auf Elemente wie das Build-System und den Debugger.

## ■ Assistent für ISAPI-Erweiterungen

Das Visual Studio definiert einen kleinen Satz an Interfaces, den ein Objekt unterstützen muss, um als DevStudio Add-In zu gelten. Wird dies von einem Objekt getan, kann es als neue Funkti-

on im Developer Studio verwendet werden. Genau diese Möglichkeit bieten aber auch andere Programme, so zum Beispiel der Internet-Information-Server von Microsoft. Der IIS hat dazu eine eigene API, die er für Erweiterungen zur Verfügung stellt, die Internet-Server-API (ISAPI). Mit einer ISAPI-Erweiterung programmieren Sie also im Wesentlichen eine Erweiterung für den Internet-Information-Server. Eine solche Erweiterung

kann die unterschiedlichsten Dinge tun. So ist es zum Beispiel möglich einen eigenen Filter für eingehende Requests zu

Webserver durchlassen, oder auch Anfragen nach bestimmten URLs in Anfragen auf andere URLs verändern.

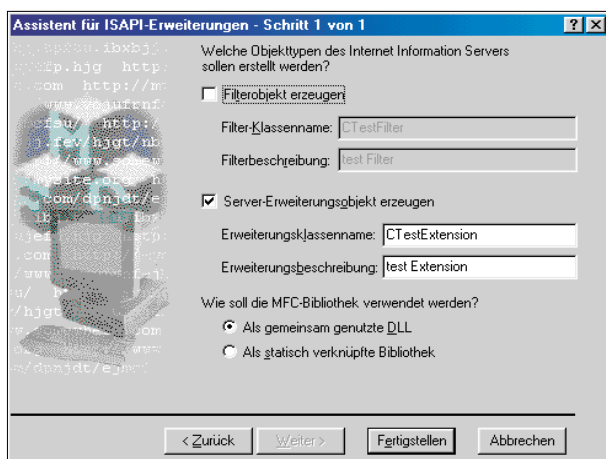
Genau wie bei DevStudio Add-Ins handelt es sich auch bei ISAPI-Erweiterungen um DLLs. Diese können dann in der Management-Konsole für den IIS unter den Eigenschaften einer Website im Reiter "ISAPI Filter" geladen und mit einer gewissen Priorität versehen werden: Dadurch können Sie beeinflussen, in welcher Reihenfolge die einzelnen Erweiterungen die Requests zu sehen bekommen.

## ■ ATL-COM-Anwendungs-Assistent

Mit diesem Assistenten haben Sie an anderer Stelle in diesem Sonderheft bereits gearbeitet. Der Assistent erzeugt den minimal benötigten Code für ein COM-Objekt, das mit der Active Template Library (ATL) implementiert werden soll. Ein einfaches COM-Objekt ist einfach nur ein Objekt, das von anderen Programmen verwendet werden kann. Allerdings gibt es bestimmte Sätze an Interfaces, die von dritten Programmen definiert wurden, die ein COM-Objekt dann zu einem "besonderen" Objekt machen. Das funktioniert ähnlich, wie bei den DevStudio Add-Ins oder den ISAPI-Erweiterungen. Liegen die Interfaces vor, kann das COM-Objekt für ganz bestimmte Zwecke verwendet werden, so kann es zum Beispiel ein HTML-Control für den Internet-Explorer werden oder andere spezialisierte Aufgaben übernehmen. Mehr zu COM-Objekten erfahren Sie im COM-Beitrag in diesem Sonderheft.

## ■ Dienstprogrammprojekt

Ein Dienstprogrammprojekt tut nicht sonderlich viel. Für ein solches Projekt werden weder bestimmte Compiler-Einstellungen noch irgendein Quellcode generiert. Vielmehr ist ein solches Projekt für Folgendes gedacht: Das Resultat des Projekts ist einfach nur ein leerer Arbeitsbereich. Der ist dann dafür gedacht, Projekte von anderer Stelle auf-

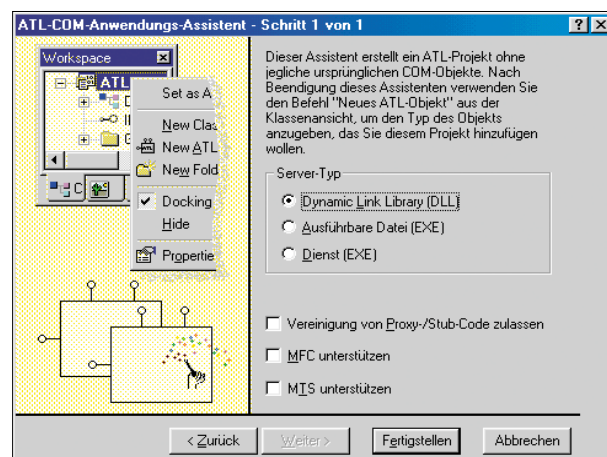


**DER ASSISTENT FÜR DIE ISAPI-ERWEITERUNG** hat nur wenige Optionen anzubieten. Im Wesentlichen müssen Sie sich entscheiden, ob Ihre Erweiterung ein reiner Filter oder auch eine Server-Erweiterung darstellen soll.

umgebung integrieren. Genau dazu ist der Projekttyp "Add-In für DevStudio" da. Dabei erzeugt der Assistent die benötigte Infrakstruktur für eine Developer-Studio-Erweiterung, die bereits alles notwendig enthält: Dazu zählt auch schon ein Button für die Werkzeugleiste und eine Minimal-Funktion, die einen kleinen Meldungstext anzeigt, wenn der Button gedrückt wird.

Wenn Sie den erzeugten Quellcode übersetzen, erhalten Sie als Resultat eine DLL. Diese DLL müssen Sie in ein bestimmtes Verzeichnis kopieren und dann die VC++-IDE einfach neu starten: Die DLL wird dann automatisch geladen. Das Verzeichnis finden Sie unter `\Programme\Microsoft VisualStudio\Common\MSDev\Add-Ins` – wenn es nicht existiert, müssen Sie das Verzeichnis anlegen.

Danach können Sie den neuen Button mit der eigenen Funktion ganz normal über den Dialog zum Einstellen der Werkzeugleisten verwenden. In der eigenen Funktion haben Sie dabei Zugriff



**DEN ATL-COM-ASSISTENTEN** haben Sie bereits im COM-Beitrag dieses Sonderheftes kennen gelernt.

programmieren. Ein solcher eigener Filter könnte zum Beispiel gefährliche Requests erst gar nicht zum eigentlichen

Resultat des Projekts ist einfach nur ein leerer Arbeitsbereich. Der ist dann dafür gedacht, Projekte von anderer Stelle auf-

zunehmen. So etwas kann man zum Beispiel für die Programmdokumentation verwenden. Das Projekt würde dann später durch die entsprechenden Dokumentationsdateien aufgestockt, die zum Beispiel im Word-Format oder in Form von Bildern vorliegen.

### ■ Makefile

Makefiles sind ein Teil der "klassischen" Werkzeuge aus der Software-Entwicklung. Für Makefiles gibt es eine eigene spezialisierte Programmiersprache, die sich ausschließlich mit der Definition von Abhängigkeiten zwischen Dateien beschäftigt. Diese Abhängigkeiten werden auch beim Build-Prozess im Developer Studio verwendet, allerdings passiert dies dort völlig transparent und ohne dass man sich als Programmierer darum kümmern müsste.

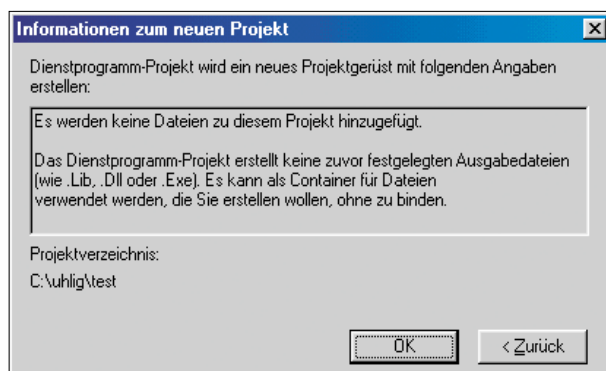
Wenn man aber ohne die IDE arbeiten will – das hat in einigen Fällen Vorteile – benötigt man dennoch eine Datei, die den Build-Prozess steuert: Welche Quellcode-Dateien müssen in welcher Reihenfolge übersetzt und später zusammengelinkt werden. Das beschreibt der Inhalt eines Makefiles. Im Normalfall werden Sie Makefiles nie benötigen.

### ■ MFC-ActiveX-Steuerelement-Assistent

Mit diesem Assistenten legen Sie ein Projekt für das Programmieren eines ActiveX-Controls an, das mit MFC programmiert wird. In den letzten Jahren sind die Grenzen zwischen ActiveX-Controls und COM-Objekten mehr oder minder unsichtbar geworden. Wenn man so will, ist ein ActiveX-Control nichts anderes als ein COM-Objekt, wenn auch eines, das einen bestimmten vordefinierten Satz an Interfaces unterstützt. Der Grund dafür ist einfach der, dass ActiveX-Controls klassischerweise ein ganz bestimmtes Einsatzgebiet haben. Es handelt sich dabei meist um wieder verwendbare Objekte, die mit einem eigenen Userinterface kommen – also zum Beispiel Dialogboxen und Ähnliches. (Das muss aber nicht so sein.).

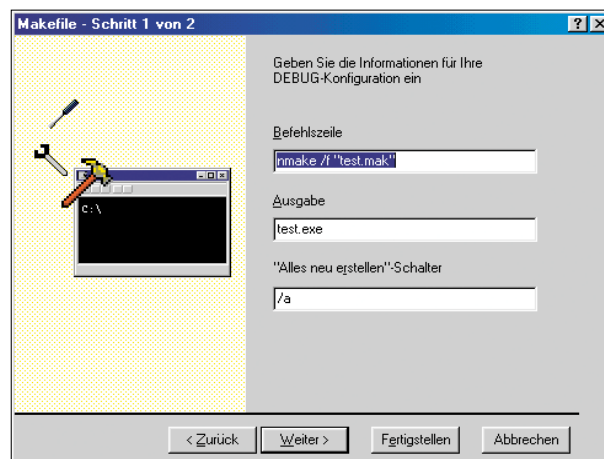
ActiveX Controls lassen sich auch auf Webseiten verwenden. In diesem Fall

würde der Client beim Herunterladen des HTML-Codes einer Webseite eine Referenz auf das ActiveX-Control entdecken, und dann dieses Control ebenfalls herunterladen. Das ist natürlich nicht ganz ungefährlich (für den Client), denn dieser setzt sich natürlich möglichen Angriffen durch das Control aus. Wurde das Control auf einen Rechner heruntergeladen und wird es dann vom Browser gestartet, stehen dem Control alle Möglichkeiten zur Verfügung, die auch ein ganz normales Programm, das auf dem Rechner ausgeführt wird, hätte. Das wären Festplatte formatieren, Da-



**EIN DIENSTPROGRAMM-PROJEKT** enthält von Haus aus nicht viel Interessantes.

teien löschen – oder auch ein hübsches Bild auf der Webseite malen: alles kein Problem. Allerdings sind ActiveX-Con-



**MAKEFILE-PROJEKTE** werden Sie praktisch nie brauchen – und wenn doch, dann werden Sie Ihre Makefiles mit Sicherheit eher von Hand schreiben, als den Assistenten dafür zu verwenden.

trols, die mit MFC programmiert werden, eher nicht für den Einsatz auf Webseiten, sondern mehr für den Einsatz in anderen Anwendungen gedacht. Der Grund dafür ist der, dass ein solches Control zur Laufzeit natürlich die etwas umfangreiche

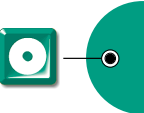
MFC-DLL benötigt – und im Normalfall wird man es zu vermeiden suchen, dass der Client diese DLL ebenfalls herunterladen muss. Daher programmiert man solche Controls besser mit der ATL.

### ■ MFC-Anwendungs-Assistent (dll)

Mit dieser Projektart erzeugen Sie eine DLL, die MFC verwendet. Eine DLL enthält Funktionen, Daten oder auch Klassen, die Sie dann in anderen Programmen wieder verwenden können. Das haben Sie an anderer Stelle in diesem Sonderheft bereits getan – nämlich als Sie ihr erstes Windows-Programm mit MFC erstellt haben. Der Grund dafür ist der, dass der größte Teil der MFC-Klassen ebenfalls in einer DLL vorliegt, nämlich der MFC-DLL. Wenn Sie Erweiterungen für MFC programmieren wollen, ist diese Projektart genau die richtige.

Der Grund, dass es für eine "DLL", die MFC verwendet, eine spezielle Projektart gibt, ist allerdings ein wenig komplizierter, als man sich das zunächst vorstellen kann. Der Hintergrund ist in der Art und Weise zu suchen, in der Windows (und MFC) Ressourcen verwaltet. An anderer Stelle im Sonderheft haben Sie bereits erfahren, dass die Elemente wie Dialogboxen und Bitmaps, die Sie in einem Windows-Programm verwenden, als Teil des ausführbaren Images – also im EXE-File – auf der Festplatte vorliegen. Jede dieser Ressourcen hat nun in Ihrem Programm eine eindeutige numerische ID, und anhand dieser ID laden Sie dann die Ressource. Diese ID hat dann noch einen symbolischen Namen, wie zum Beispiel IDB\_BITMAP1, und im tatsächlichen Quellcode laden Sie die Ressource mit einem Funktionsaufruf, wie zum Beispiel LoadBitmap (IDC\_BITMAP1).

Dieser Ansatz ist jedoch leider noch nicht gut genug. Und der Grund dafür sind eben DLLs. Die ID für eine Ressource wird vom Developer-Studio automatisch generiert, und das Studio kümmert sich dabei auch darum, dass diese ID tatsächlich eindeutig ist und



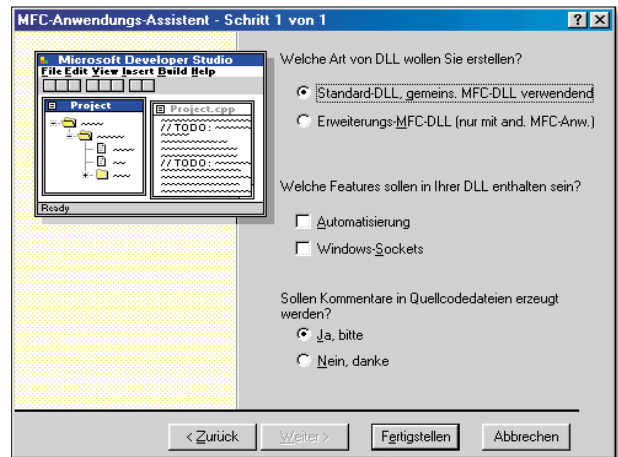
sich nicht mit anderen IDs von anderen Ressourcen im Projekt überschneidet.

Das Problem ist nun, dass die DLLs, die Sie im Programm verwenden nicht notwendigerweise im gleichen Projekt erzeugt werden, wie das Programm, das die DLLs benutzt. Mit anderen Worten: Die DLL wird im Normalfall immer in einem anderen Projekt erzeugt werden – und auch dort stellt das Developer-Studio sicher, dass sich die Ressource-IDs nicht überschneiden: Was es allerdings nicht tut, ist sicherzustellen, dass sich die Ressource-IDs der beiden Projekte nicht überschneiden, und das wäre auch ein bisschen viel verlangt, denn schließlich kann die DLL ja von einer ganz anderen Person auf einem ganz anderen Rechner angelegt werden. Nun ist ein Programm außerdem auch nicht darauf beschränkt nur eine DLL zu verwenden, sodass auch multiple Ressourcen mit der gleiche ID auftreten können. Zur Laufzeit werden nun Ressourcen anhand einer Ressource-Chain (Ressourcen-Kette) gesucht: Dabei wird zunächst das Ressourcen Verzeichnis der EXE-Datei und dann die Verzeichnisse der vom Programm gela-

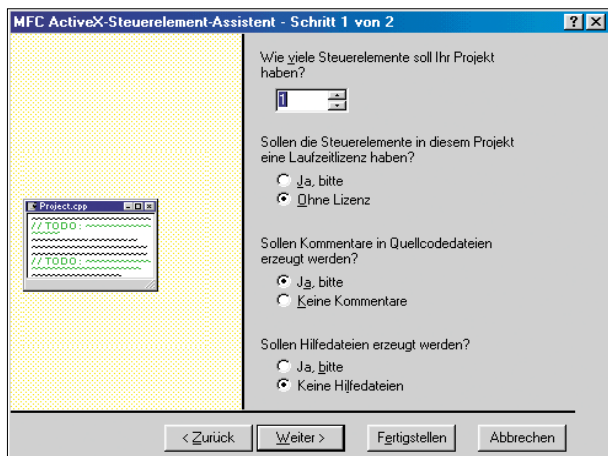
kann dadurch sicherstellen, dass Ihre eigenen sich nicht mit deren überschneiden. Sobald Sie aber eine DLL programmieren, wird das schwieriger.

Im Zusammenhang mit der Nutzung von DLLs gibt es noch weitere Probleme, so zum Beispiel bei der Speicherverwaltung: Der Code aus der DLL wird im gleichen Adressraum mit dem Code aus dem ausführbaren Programm ausgeführt – und verwendet auch die gleiche Speicherverwaltung. Zumindest dann, wenn die normalen Win32-APIs für die Speicherverwaltung benutzt werden. Die MFC hat aber eine eigene Speicherverwaltung – und in diesem Zusammenhang kann es dann durchaus passieren, dass die an einem Programm beteiligten Komponenten sich nicht darüber

heft bereits an einem Beispiel getan. Für derartige Programme bietet der Assistent eine ganze Reihe Optionen an. So können Sie SDI, MDI oder auf Dialog-



**DEN ASSISTENTEN FÜR EINE MFC-ANWENDUNG** kennen Sie bereits aus einem Beispiel in diesem Sonderheft.



**DER ASSISTENT FÜR ACTIVE-X-STEUERELEMENTE** erzeugt den für ActiveX benötigten Rahmencode und kümmert sich auf Wunsch auch darum, dass das Projekt mehr als ein Element enthalten soll.

denen DLLs durchsucht. Die erste Ressource mit der gesuchten ID ist dann die, die geladen wird. Mit anderen Worten: Welche Ressource Sie dann erhalten ist völliger Zufall.

Nun ist es aber so, dass die MFC-DLL eben sehr wohl auch eigene Ressourcen verwendet, und das ist der erste Fall, bei dem Sie Probleme bekommen. Die sind allerdings noch nicht gar so groß, denn das Developer Studio kennt die von den MFC benutzten Ressource-IDs und

einig sind, wer für einen bestimmten Speicherbereich beziehungsweise dessen Freigabe zuständig ist.

Letzten Endes ist es nun mal so, dass die MFC einen riesigen Berg an Funktionalität bietet – wer den aber nutzen will, der muss sich auch an die Regeln von MFC halten und kann nicht einfach völlig frei mit der Win32-API herumprogrammieren. Das trifft im Besonderen auf DLLs zu, die die MFC benutzen wollen – und der Quellcode, den der MFC-Anwendungs-

Assistent (dll) erzeugt, trägt diesem Umstand Rechnung.

## FC-Anwendungs-Assistent (exe)

Diesen Assistenten bzw. diesen Projekttypen werden Sie am häufigsten verwenden, denn damit programmiert man ganz normale Windows-Programme. Diese Programme werden mit Hilfe der MFC Library programmiert – Sie haben das an anderer Stelle in diesem Sonder-

box basierende Programme schreiben, die Toolbars können auf unterschiedliche Arten definiert werden und auch das Layout der Fenster können Sie beeinflussen. So steht ein ganz normales Layout mit einem ganz normalen Arbeitsbereich zur Verfügung, die Fenster können aber auch wie beim Windows Explorer – also mit einer Baumansicht und einem Arbeitsbereich – angelegt werden.

Innerhalb eines solchen Projekts stehen Ihnen alle MFC-Klassen zur Verfügung, und diese bieten einen sehr großen Funktionsumfang. So gibt es bestimmte spezialisierte Fensterklassen, zum Beispiel eine, die Webseiten-Inhalte anzeigen kann, und auch Klassen für andere Funktionen, wie zum Beispiel fürs Abarbeiten von Kommandos oder das einfache Speichern von Objekten. Letzteres wird "Serialisierung" genannt, das Gegenstück dazu – also das Laden – hat den Namen Deserialisierung.

Im Lieferumfang der MFC-Dokumentation ist auch ein Klassen-Diagramm enthalten, in dem alle zu MFC gehörenden Klassen und deren Vererbungsstruktur abgebildet sind. Dieses Diagramm sollten Sie auf jeden Fall studieren – es liefert Ihnen eine sehr gute Übersicht über die Klassen, die Ihnen bei der Arbeit zur Verfügung stehen.

## Win32 Dynamic Link Library

Bei einer Dynamic Link Library handelt es sich einfach um eine andere Form, aus-

föhrbaren Code zu transportieren. Die Bezeichnung "Win32" bedeutet einfach, dass es sich nicht um irgendeine spezielle DLL-Variante handelt, sondern um eine, die die ganz normale Win32-API benutzt. Beim Programmieren ist der wesentliche Unterschied zwischen einer DLL- und einer EXE-Datei der, dass eine DLL keine *main()*-Funktion enthält. Die tatsächlichen Unterschiede gehen aber noch deutlich weiter.

Wie gesagt, "transportiert" man in einer DLL ausführbaren Code. Das hat auch einen Grund, und der liegt in der Möglichkeit der Wiederverwendbarkeit. Je nachdem, wie Sie Ihre DLL anlegen, kann der Programmcode darin entweder von praktisch allen anderen Programmen oder aber nur von Ihrem eigenen Programm – oder bestenfalls von einem Programm, das mit dem gleichen C++-Compiler erstellt wurde – benutzt werden.

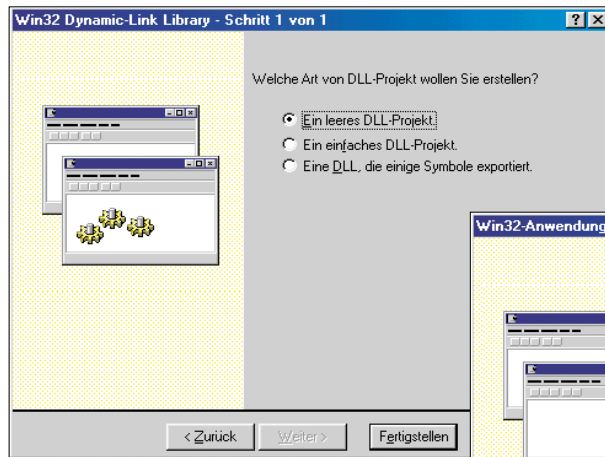
Das hängt damit zusammen, dass Programme, die die Funktionen (oder Klassen) in der DLL nutzen wollen, natürlich irgendetwas über diese Funktionen und Klassen wissen müssen. Bei Funktionen, zumindest wenn es sich um reine "C"-Funktionen handelt, ist das noch relativ einfach, denn wie diese in einer DLL vorliegen, ist durch die Win32-API definiert. Eine reine C-Funktion kann aus der DLL exportiert werden. Das geht entweder mit der von Win32 zur Verfügung gestellten Funktionalität in Form einer .DEF-Datei für den Linker, oder durch VC++ spezifische Schlüsselwörter (*dllimport* und *dllexport*), die aber im Endeffekt die gleiche Wirkung wie ein .DEF-File haben.

Solche Funktionen können dann von anderen Programmen grundsätzlich benutzt werden, zumindest dann, wenn keine Speicherverwaltung ins Spiel kommt. Also "C"-Funktionen, die keine Zeiger als Parameter erhalten und keine Zeiger liefern, gehen immer – kommen Zeiger ins Spiel, wird die Sache komplizierter (aber das sind Sie ja bereits gewohnt). Dabei ist zu beachten, dass eine Funktion dadurch zu einer

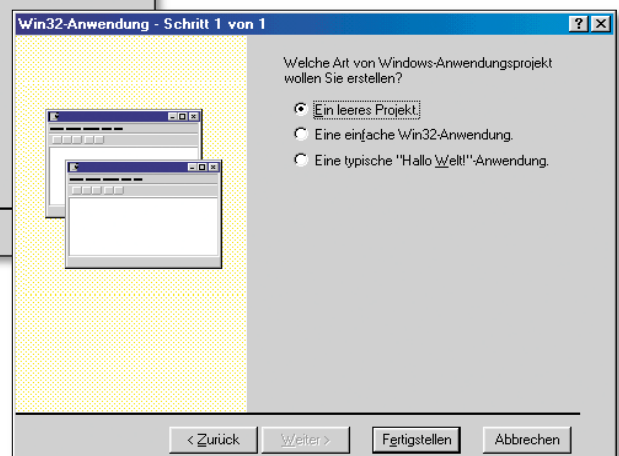
"C"-Funktion wird, indem sie in einer Quelldatei mit der Erweiterung .C vorliegt. Wenn Sie die Datei dann in .CPP umbenennen, wird aus der Funktion eine C++-Funktion – und dann kann Sie nicht mehr von Programmen aufgerufen werden, die nicht mit dem Microsoft

der die Funktion übersetzt hat, diese auch nicht finden.

Wie auch immer: Bei "C"-Funktionen haben Sie damit keine Probleme. Es hält Sie auch nichts davon ab, die Implementierung der Funktionalität innerhalb der DLL komplett in C++ vorzunehmen – Sie müssen dann nur zusätzlich eine "C"-Funktion in der DLL platzieren, von der aus diese Funktionalität erreichbar ist. Was immer auch an Funktionalität in der DLL enthalten ist, wird nach außen nur in Form einer einzigen "C"-Funktion sichtbar.



**BEI DEN DLL-PROJEKTEN** sollten Sie auf jeden Fall die letzte Option wählen: Dann haben Sie gleich ein Beispiel dafür, wie man Symbole aus einer DLL exportiert.



**AUCH BEI WIN32-ANWENDUNGEN** ist eigentlich immer die letzte Option die richtige, denn irgendeine Form von Ausgabe wird das Konsolen-Programm mit Sicherheit benötigen.

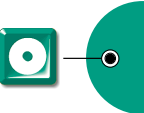
C++-Compiler erzeugt wurden. Man kann allerdings auch "C"-Funktionen in CPP-Dateien anlegen, denn dazu gibt es ein spezielles Schlüsselwort. Der Grund dafür, dass man eine C++-Funktion, die mit VC++ geschrieben wurde, nicht von Programmen benutzen kann, die mit einem anderen Compiler geschrieben wurden, ist die Art und Weise, wie in C++ die internen Namen erzeugt werden. Nachdem Sie in C++-Funktionen mit gleichem Namen unterschiedliche Parameter haben können, reicht es für die Bearbeitung durch den Linker und den Compiler nicht aus, alleine den Funktionsnamen für die Identifizierung einer Funktion zu verwenden.

Statt dessen wird der intern verwendete symbolische Name aus dem Funktionsnamen, dem Datentyp des Rückgabewertes der Funktion und den Datentypen der Parameter der Funktion zusammengesetzt. Wie das allerdings genau geschieht ist in der C++-Spezifikation nicht vorgeschrieben, und deshalb verwenden alle Compiler hier eine andere Methode. Als Resultat haben Sie bei allen Compilern andere Namen für die vorliegenden Funktionen – und darum kann ein anderer Compiler als der,

Ist die Verwendbarkeit in anderen Sprachen kein Thema, können Sie auch komplette Klassen aus einer DLL exportieren, und diese dann genau so in Ihrem Programm verwenden, wie Sie das auch mit den Klassen der MFC tun.

## Win32-Anwendung

Eine Win32-Anwendung ist das Gegenstück zu einer MFC-Anwendung – nur eben ohne die MFC. Ein solches Programm verwendet die reine auf "C" basierende Windows-API, hat eine eigene sichtbare Message Loop und all die anderen unangenehmen Eigenschaften eines "C"-Programms, das ein Windows-Programm sein will. Das geht und wurde lange Jahre so praktiziert (vor allem zu Zeiten von 16bit-Windows, als Klassenbibliotheken für Windows nicht verfügbar waren) – aber heute sollte man so etwas besser unterlassen. Diese Art der Programmierung für Windows ist hochgradig ineffizient. Rein interessenshalber sollten Sie aber zumindest einmal ein solches Projekt erzeugen und den Quellcode studieren – dadurch gewin-



nen Sie auch einen Eindruck, was die MFC-Bibliothek Ihnen alles an Arbeit abnimmt und was sich hinter den wenig eindrucksvollen Klassennamen, wie "CWinApp" verbirgt.

Das vom Assistenten erzeugte Rahmenprogramm ist dabei völlig ausreichend: Einfach zum Ausprobieren sollten Sie in dieses Programm einmal einen neuen Menübefehl, der eine Dialogbox öffnet, einbauen: Wenn Sie dieser Versuch nicht davon überzeugt, Programme zukünftig lieber mit MFC zu bauen, haben Sie vermutlich persönliche Vorbehalte C++ nicht zu benutzen.

## Win32-Bibliothek (statische)

Um Code wiederzuverwenden gibt es eine ganze Reihe von Ansätzen. Der am wenigsten sinnvolle ist dabei der, den Quellcode aus der einen Datei einfach in eine andere Datei zu kopieren. Dadurch entsteht das Problem, dass zwei (oder noch mehr) Kopien des gleichen Codes gewartet werden müssen. Einen anderen Ansatz kennen Sie schon von weiter oben im Beitrag, nämlich die Verwendung einer DLL. Bei einer solchen Bibliothek liegt der Programmcode in übersetzter Form vor, und wird dann von anderen Programmen zur Laufzeit geladen und verwendet. Das ist in vielen Fällen wünschenswert, aber nicht in allen.

Manchmal möchte man den Code zwar wieder verwenden, aber eben nicht in Form einer für jedermann sichtbaren DLL. In diesem Fall verwendet man statische Bibliotheken. Eine statische Bibliothek ist eine Datei, die übersetzten und lauffähigen Code enthält, der aber bereits während des Build-Prozesses vom Linker direkt in die EXE-Datei kopiert wird. Im Gegensatz zu einer DLL, die erst zur Laufzeit geladen wird, ist der Code bei einer statischen Library also schon von vornherein im Programm vorhanden.

Das macht das .EXE-Programm natürlich größer, dafür aber die Ladezeit des Programms unter Umständen kürzer, denn das Programm muss ja kei-

ne DLLs mehr zusammensuchen und die Funktionen darin auflösen. Das hat der Linker bereits erledigt. Ansonsten ist nicht viel zu Win32-Libraries zu sagen – es handelt sich um einen praktischen Mechanismus zur Wiederverwendung von Code zur Linkzeit des Programms.

## Win32-Konsolenanwendung

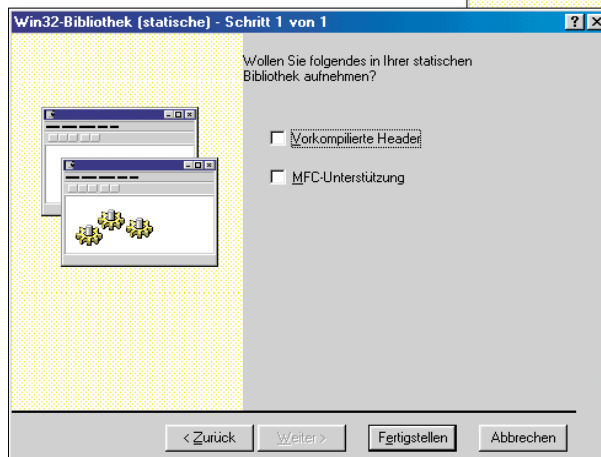
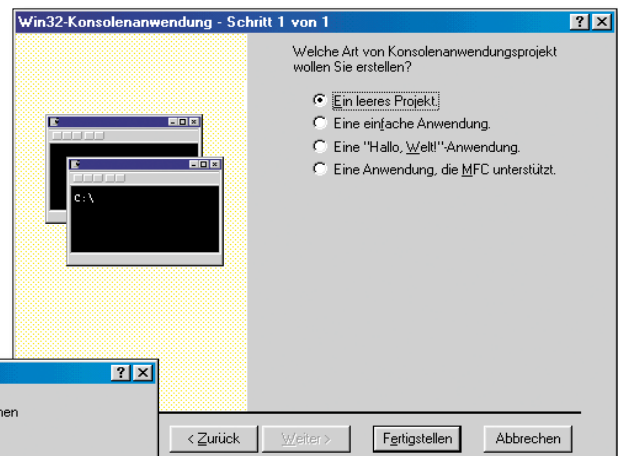
Diese Projektart haben Sie im ersten Teil des Sonderheftes schon oft benutzt, darum soll sie hier nicht ausführlich besprochen werden. Im Wesentlichen handelt es sich bei einem solchen Projekt um eines, das eine "echte" 32bit-Windows-Anwendung erzeugt, allerdings verwendet die Anwendung keinerlei Fenster, sondern ist rein textbasiert. Das ist für viele Zwecke ausreichend, wie Sie an den Beispielen im ersten Teil des Sonderheftes gesehen haben.

## VC++ und die Konfigurationen

Innerhalb eines Projektes können Sie unterschiedliche Konfi-

deren am Build-Prozess beteiligten Werkzeuge. Der Sinn der Sache ist der, mit dem gleichen Quellcode verschiedene Dinge erreichen zu können. So ist das Resultat eines "debug"-Builds zwar das gleiche Programm wie beim "Release"-Build – allerdings sind in der Debug-Variante Informationen für den Debugger im generierten Code enthalten, während im Release-Build der optimierende Compiler zum Zuge kommt und das Programm kleiner und schneller machen kann.

Sie können auch mehr als nur zwei Konfigurationen pro Projekt anlegen. Eine sinnvolle Konfigurationsunterscheidung ist zum Beispiel, eine Projektvariante für Unicode und eine für ANSI anzulegen (das geht rein über Präprozessordefinitionen), oder aber



**OB DIE STATISCHE BIBLIOTHEK** nun vorkompilierte Header verwendet oder nicht, hat auf die Funktionalität darin keinen Einfluss. Das Kompilieren geht aber mit dieser Option meist schneller.

gurationen anlegen, und die Assistenten von VC++ tun dies meist von Haus aus auch. Sie haben praktisch immer eine "Debug"- und eine "Release"-Konfiguration. Diese Konfigurationen unterscheiden sich ausschließlich durch die Compiler- und Linkereinstellungen, bzw. durch die Einstellungen für die an-

**DER WESENTLICHE UNTERSCHIED** bei den Optionen dieser Projektart besteht darin, ob Sie MFC verwenden möchten oder nicht, denn das hat einen starken Einfluss auf die Compiler- und Linker-Optionen.

Sie erzeugen eine Projektkonfiguration, die ganz normal mit DLLs arbeitet, während eine andere möglichst wenig externe Abhängigkeiten hat und den benötigten Code in Form von statischen Libraries linkt: Je nachdem welches Ziel Sie verfolgen, sind all diese Möglichkeiten sinnvoll.

In diesem Beitrag haben Sie erfahren, welche Möglichkeiten Ihnen mit den verschiedenen VC++-Projekten zur Verfügung stehen – und an welchen Stellen bei einigen davon Probleme versteckt sind. Das sollte Sie aber nicht davon abhalten, zumindest einmal alle verfügbaren Projektarten auszuprobieren. Unter Umständen stoßen Sie damit auf bisher ungeahnte Möglichkeiten. ✓ UR