Buildungslücke

Den Open Source Code von InterBase selbst kompilieren

von Karsten Strobel

Haben Sie schon mal versucht, mit selbstgezogenem Gemüse einen Konserveneintopf nachzukochen? Die Zutatenliste auf der Dose hilft nur wenig weiter wenn es gilt, den exakt gleichen Geschmack wie beim Fertigprodukt aus dem Supermarkt hinzukriegen. Greifen Sie also doch lieber zur Instant-Mahlzeit, oder haben Sie den Ehrgeiz, eines Tages als Meister unter den Hobbyköchen das Rezept noch zu verfeinern...?

Im Sommer 2000 veröffentlichte Borland die Quellcodes des Datenbankservers InterBase unter einer der Mozilla Public License ähnlichen Lizenz [1]. InterBase wurde Open Source, und auf dem Markt erschien plötzlich eine hochentwickelte relationale Datenbank für die Plattformen Win32, Linux und Sun Solaris (und der Option auf weitere Portierungen). 16 Jahre Entwicklungszeit oder mehr steckten bereits in diesen Quellen – ein wahrhaftes Kleinod unter den Datenbanken und erst recht unter den Open-Source-Vertretern dieser Gattung, mit geringem Marktanteil zwar, aber unter der eingeschworenen Gemeinde seiner Anhängerschaft herrscht fast ungeteilte Überzeugung von den Qualitäten dieses Produkts.

Quellcode

Den Quellcode finden Sie auf der aktuellen Profi-CD sowie auf unserer homepage unter
www.derentwickler.de

Eigentlich hätten sich Heerscharen von Entwicklern mit Begeisterung auf die Quellen stürzen sollen, um in gemeinsamer Anstrengung und unter Ableistung vieler, vieler ehrenamtlicher Stunden den bisher verborgenen Schatz zu sichten und seinen Glanz mit Bugfixes und neuen Features noch strahlender zu machen. In der Realität indes zeigt sich, dass die Begeisterung für die Open-Source-Bewegung zwar eine breite Basis besitzt, insbesondere weil sie die Lizenzkosten praktisch auf Null reduziert. Zu einer aktiven Teilnahme – sprich: Einarbeiten in den Quellcode, Teilnahme an den Diskussionsforen und Erarbeiten eigener Verbesserungen – sind allerdings nur wenige bereit. Begabte und kreative Programmierer haben zumeist auch so schon viel zu tun und können oder wollen daher für Projekte des IT-Gemeinwohls nicht viel Zeit aufbringen.

Der Weg ins Labyrinth

Im Fall InterBase erschwert ein weiterer Umstand den Start in das Dasein eines ehrenamtlichen Mit-Autors: Der Source ist mit rund einer Million Zeilen C-Code sehr umfangreich, seine Struktur kaum dokumentiert und der Weg vom Quellcode zum fertigen Kompilat (der sogenannte Build-Prozess) erfordert einiges an „Geheimwissen“, das man sich recht mühevoll zusammensammeln muss. Den letztgenannten Missstand soll dieser Artikel, zumindest für die Plattform Win32, wenigstens teilweise beseitigen. Die gute Nachricht ist, dass zum Übersetzen von InterBase kein Bedarf an fremden Zutaten wie etwa Funktionsbibliotheken anderer Softwarehäuser besteht.

Es existiert nur eine Version des Quellcodes, der für alle unterstützten Plattformen verwendet werden kann, was durch intensive Anwendung bedingter Kompilierung erreicht wird. Selbstverständlich benötigt man für jede Plattform einen anderen Compiler. Ich möchte (und kann) mich hier nur mit der Win32-Plattform beschäftigen. Überhaupt werde ich mich in diesem Artikel darauf beschränken, die notwendigen Schritte bis zum erfolgreichen Kompilieren von InterBase zu erläutern. Fragen der Architektur oder Ansätze für eigene Modifikationen an den Quellcodes bleiben unberücksichtigt. Ich möchte lediglich eine Anleitung dazu liefern, die erste große Hürde zu überwinden. Alles weitere überlasse ich ganz der Kreativität und dem Enthusiasmus der Leser.

Man nehme...

Hier eine Liste der Zutaten, die Sie für die Zubereitung des InterBase nach Hausmacher Art benötigen werden:

- Windows NT oder 2000 auf der „Build-Box“
- Einen CVS-Client
- Den InterBase Quellcode
- Microsofts Visual C++-Compiler
- Eine Ersatzlösung für verschiedene Unix-Befehle
- Den Build-Batch
- Tools aus Borlands C++Builder
- Einen fertigen, installierten InterBase 6 Server
- Die Build-Datenbanken
- Einige Pfad-Definitionen und Umgebungsvariablen

Alle Build-Batches und Anleitungen, die man als Zugabe zu den Open-Source-Versionen von InterBase 6 vorfinden kann, sind für Windows NT (respektive Windows 2000) geschrieben worden und enthalten einige Elemente, die unter 9x/ME nicht funktionieren. Es ist zwar durchaus möglich, dass man den Build-Prozess auch unter nicht-NT-Varianten von Windows – mit ein paar Modifikationen – zum Laufen bringt, aber ich habe das selbst nicht versucht und möchte daher empfehlen, wenn möglich einen Rechner mit NT/2000 für das Umwandeln der Quellen zu verwenden.

Um mit dem InterBase-Quellcode arbeiten zu können, müssen Sie diesen erst einmal per Internet von sourceforge.net herunterladen. SourceForge ist ein Serverdienst, der Open-Source-Projekten eine virtuelle Heimat bietet, und der – neben InterBase – den Quellcode zehntausender Projekte verwaltet. Jedermann kann hier ein neues Projekt starten und seine Quelldateien auf diesem Server bereitstellen, sofern gewisse Regeln bezüglich der Lizenzbedingungen eingehalten werden. Leider lassen sich die Quellen vieler Projekte nicht einfach per ftp herunterladen. Source-Forge arbeitet als sogenannter Concurrent Versions System (CVS) Server [2]. Als solcher unterstützt er eine Vielzahl von Funktionen, die für das gemeinsame Arbeiten vieler Programmierer an einem Projekt wichtig sind. Er protokolliert und unterstützt die Dokumentation von Änderungen, stellt ältere Versionen auf Wunsch wieder zur Verfügung, vergleicht Versionen miteinander und verwaltet die Benutzer und deren Rechte. Wir wollen hier aber nicht näher auf die vielfältigen Möglichkeiten des CVS eingehen, sondern uns darauf beschränken, die aktuelle Version des Quellcodes abzurufen. Dazu benötigen Sie zunächst einmal einen CVS-Client, den Sie sich zum Beispiel unter www.cvshome.org herunterladen können. Außerdem finden Sie eine Version auf der Profi-CD und auf den Webseiten des Entwicklers. Es gibt auch eine Reihe grafischer Oberflächen für CVS, aber wir wollen hier ganz einfach die Kommandozeilenversion (CVS.EXE) verwenden (siehe Listing 1).

Diese Anweisungen setzen voraus, dass Sie für CVS.EXE ein Verzeichnis namens *C:\cvs* angelegt haben, und natürlich, dass Sie über eine Verbindung zum Internet verfügen. Mit der Umgebungsvariablen HOME geben Sie den Ort an, an dem der CVS-Client eine Datei namens *.cvspass* anlegen soll. Diese wird mit den *logon*-Kommandos erzeugt und speichert Benutzernamen und Password für den weiteren Zugriff auf den CVS-Server. Selbst für anonyme Zugriffe ohne Schreibrecht – wie hier verwendet – ist ein *logon* obligatorisch, ein *logout* hingegen ist nicht erforderlich. Mit CVSROOT wird das Ziel der darauf folgenden CVS-Aufrufe festgelegt. Das Schema dieser Ortsangabe ist *:modus:user@host:/path*. Der Modus *pserver* bezeichnet eine auf Passwort-Authentisierung beruhende Zugriffsform per TCP/IP und *anonymous* den besonderen Benutzernamen für den Zugriff inkognito. Nach dem @-Zeichen wird der Name des Servers angegeben, hinter dem schließlich der auf diesem Server zu öffnende Pfad angegeben wird. Letzteren muss man aus der Projektdokumentation übernehmen. Das eigentliche Herunterladen der gewünschten Arbeitskopie beginnt mit dem *checkout*-Kommando. Die Option *–z3* gibt dabei den empfohlenen Kompressionslevel für die Übertragung an. Nach dem *checkout*-Kommando wird der Name des herunterzuladenden Moduls spezifiziert. Damit wird ein kompletter Download aller Dateien dieses Moduls ausgelöst, was einige Zeit in Anspruch nimmt. Auf der lokalen Festplatte wird unterhalb des aktuellen Verzeichnisses der komplette Quellcode-Baum mit allen Quelldateien aufgebaut. Wer mehr über die Fähigkeiten des CVS-Clients erfahren will, wird bei einer kurzen Suche im Internet reich belohnt [3].

Mit den hier abgedruckten Kommandos rufen Sie die Quellcodes beider Antipoden – InterBase und Firebird – ab, die

Listing 1

```
C:\cvs>set HOME=C:\cvs
C:\cvs>md Borland
C:\cvs>md Firebird
C:\cvs>cd Borland
C:\cvs\Borland>set CVSROOT=:pserver:anonymous@cvs.interbase.sourceforge.net:/cvsroot/interbase
C:\cvs\Borland>..\cvs logon
(Logging in to anonymous@cvs.interbase.sourceforge.net)
CVS password: <kein passwort eingeben, nur return drücken>
C:\cvs\Borland>..\cvs -z3 checkout InterBase
cvs server: updating InterBase ...
...
C:\cvs\Firebird>..\Firebird
C:\cvs\Firebird>set CVSROOT=:pserver:anonymous@cvs.firebird.sourceforge.net:/cvsroot/firebird
C:\cvs\Firebird>..\cvs logon
(Logging in to anonymous@cvs.firebird.sourceforge.net)
CVS password: <kein passwort eingeben, nur return drücken>
C:\cvs\Firebird>..\cvs -z3 checkout interbase
cvs server: updating interbase ...
...
```

nämlich beide von SourceForge gehostet werden. Achten Sie bei der Eingabe der Modulnamen (*InterBase* bzw. *firebird*) bitte auf die Groß- bzw. Kleinschreibung. Welcher der beiden Geschmacksrichtungen Sie auch immer den Vorzug geben, das im folgenden beschriebene Rezept zur Zubereitung sollte gleichermaßen anwendbar sein. Es soll dabei nicht unerwähnt bleiben, dass das Firebird-Projekt inzwischen (unter dem neuen Modulnamen *firebird2*) im Interesse des Fortschritts mehr und mehr von dem ursprünglichen Zuschnitt des Quellenbaumes abweicht und in dieser neuen Erscheinungsform auch andere Build-Vorschriften erlassen hat. Um aber nicht noch zusätzliche Verwirrung zu stiften, beschränke ich mich hier auf den „klassischen", von Borland gepflanzten Baum, wie man ihn auch bei Firebird noch im Modul *interbase* vorfindet.

Nachdem wir nun Arbeitskopien beider Sourcecodebäume besitzen, müssen Sie einen davon erwählen *(c:\cvs\Borland\InterBase* oder *c:\cvs\Firebird\interbase*) und in ein neu anzulegendes Verzeichnis C:\IB_BUILD_DIR kopieren. Behalten Sie am besten das Original Ihrer Arbeitskopie, denn Sie können damit schnell wieder an einen definierten Ausgangspunkt zurückkehren, ohne den ganzen CVS-Checkout wiederholen zu müssen. Das Ergebnis sollte nach dem Kopieren etwa so aussehen, wie in Abbildung 1 gezeigt. Bitte stellen Sie sicher, dass für alle heruntergeladenen Dateien das Schreibschutzattribut ausgeschaltet ist (z. B. mit „*attrib –R *.* /S*").

Einen ganz groben Überblick über diese Verzeichnisstruktur und die Bedeutung einiger wichtiger Dateien erhält man auf den Webseiten der InterBase Entwickler-Initiative [4].

In fremden Gefilden

Für viele wird folgendes ziemlich überraschend klingen: Die InterBase-Quellen werden mit Microsoft Visual C++ übersetzt. Warum hat sich Borland hier gegen den hauseigenen C-Compiler und für ein fremdes Produkt entschieden? Man muss etwas in der Zeit zurückgehen, um diesen Entschluss verstehen zu können. Als Borland die Portierung von InterBase für Win32 begann, verfolgten die Redmonder mit Windows-NT noch eine Multiplattform-Strategie. Windows NT 3.x gab es außer für Intel-PCs auch noch für die Architekturen Alpha, PowerPC und MIPS. InterBase wurde zunächst zwar nur für Intel vom Stapel gelassen, aber man wollte sich damals bei Borland den Weg zu NT-Versionen für andere Prozessoren nicht verbauen. Daher gab man dem C-Compiler von Microsoft den Vorzug, denn allein dieser stand auch auf anderen NT-Plattformen zur Verfügung. Seitdem hat sich Microsoft mit NT 4.0 teilweise und mit Win2000 ganz von der Vision eines Windows für alle Hardwarewelten verabschiedet, womit die Entscheidungsgrundlage für den Compiler dieses Konkurrenten eigentlich hinfällig geworden ist. Aber der Wechsel zu Borland C++ wäre aufwändig [5] und würde kaum einen technischen Vorteil mit sich bringen, weshalb man auf dem einmal eingeschlagenen Weg blieb und wohl auch weiter bleiben wird.

Sie brauchen also Microsoft Visual C++ in der Version 6. Die Standard-Edition ist ausreichend. Der Service Pack 5 für den Compiler sollte installiert sein. Falls Sie diesen Service Pack nicht auf CDROM besitzen und ihn via Internet beschaffen wollen, so stellen Sie sich bitte auf eine längere Downloadzeit ein, denn es ist kein separater Service Pack für MSVC++, sondern nur für das Bundle Visual Studio verfügbar [6]. Dieser Service Pack (*vs6sp5.exe*) aktualisiert alle zu Visual Studio gehörenden Produkte (Visual Basic, Fox Pro etc.), die auf Ihrem Rechner installiert sind, also gegebenenfalls auch nur den C-Compiler. Mit einer Größe von über 128 MB kann man diese Datei aber wohl nicht gerade als handlich bezeichnen. An einigen Stellen der original Build-Instruktionen wird übrigens auch MSVC++ Version 5 SP3 empfohlen. Es ist also davon auszugehen, dass verschiedene Konfigurationen funktionieren, selbst getestet habe ich aber nur die zuvor beschriebene.

Der Ort der Installation von MSVC++ ist nicht weiter wichtig, sofern die Kommandozeilenversion des Compilers über die Umgebungsvariable PATH gefunden werden kann. Dies ist nach einer normalen Installation der Fall. Sie sollten zum Beispiel per Kommandozeile CL.EXE aufrufen können, ohne dafür in einen bestimmten Pfad wechseln zu müssen.

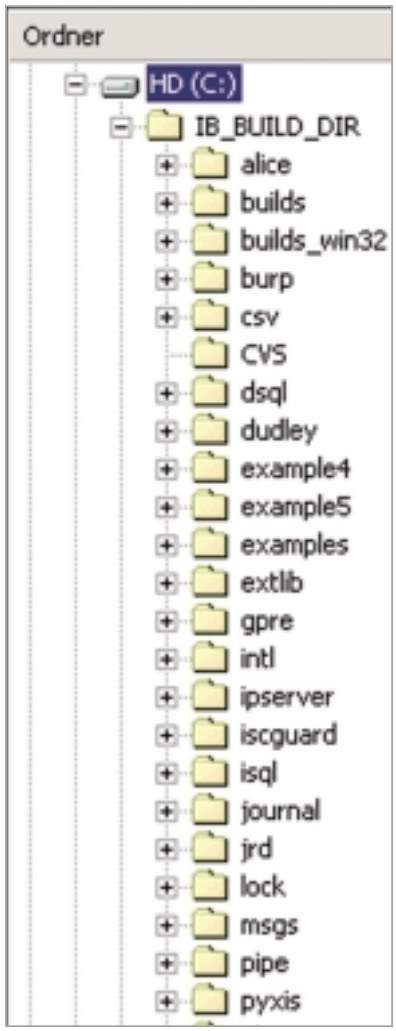

Abb.1: Der Quellcodebaum

Die Build-Instruktionen und -Skripte, die Sie nach dem Herunterladen Ihrer Arbeitskopie des CVS-Baums vorfinden, sind für den eingefleischten WinDos-Entwickler leider alles andere als einfach anwendbar. Erste Informationen über das Kompilieren für Windows finden Sie unter *builds\original\build_windows.txt*. Wenn Sie die Anweisungen in dieser Datei durchackern, treffen Sie unter der Überschrift „Before starting" auf den Hinweis, dass Sie den „MKS Shell Toolkit" – für Leute ohne Unix-Erfahrung ist das im Allgemeinen ein Fremdkörper – installieren sollen. An anderer Stelle heißt es, man könne es ebenso gut auch mit „bash" versuchen. Was hat es damit auf sich? In bei-

den Fällen sind Portierungen von unter Unix gängigen Shells für Win32 gemeint, die als Kommandointerpreter für die Abarbeitung einiger Skripte verwendet werden sollen. In Ihrem Verzeichnis *c:\IB_BUILD_DIR* finden Sie die Datei *setup_dirs.ksh*, die Sie laut den Anweisungen in *build_windows.txt* ausführen sollen. Die Namenserweiterung *.ksh* steht für „Korn Shell", eine Gattung dieser Unix-üblichen Interpreter. Dieses Script ist zusammen mit *setup_build.ksh* für die Vorbereitung des eigentlichen Build-Prozesses zuständig. Es legt einige Verzeichnisse an, kopiert Dateien und nimmt Anpassungen von Pfadbezeichnungen etc. vor.

Es hat sich gezeigt, dass die Bereitschaft der Entwickler, die an Windows gewöhnt sind und keine oder wenig Unix-Kenntnisse haben, sich mit einer zusätzlichen Shell herumzuschlagen, durchweg gering ist. Eigentlich werden auch nur ganz wenige Befehle dieser mächtigen Interpreter wirklich benötigt. Paul Reeves von IBPhoenix hat sich daher der Aufgabe angenommen, den komplizierten Build-Vorgang unter NT etwas zu vereinfachen und mit einem einzigen Batch abzuwickeln, der ohne MKS Shell oder bash auskommt. Auf dem Ergebnis seiner Arbeit aufbauend, habe ich unter dem Dateinamen *IBFB_Build_Win32.bat* einen Batch bereitgestellt, der im weiteren verwendet werden soll. Außer den unter NT gängigen Befehlen werden darin nur noch die von Unix entlehnten Toolprogramme *cat.exe* (Dateien verbinden), *cp.exe* (ein copy-Ersatz), *sed.exe* (der sog. Streameditor, ein Textprozessor zum Suchen und Ersetzen von Ausdrücken) und *tail.exe* (ein Pendant des Befehls *more*, der die letzten Zeilen einer Textdatei ausgibt), ferner *mv.exe* (Move) und *cmp.exe* (Compare) verwendet, die unter GNU-Lizenz [7] als freie Software auch für Win32 zur Verfügung stehen [8]. Die Batchdatei und die sechs Programmdateien finden Sie im Web bzw. auf der Profi-CD in der Archivdatei *IBFB_Build.zip*. Extrahieren Sie bitte *IBFB_Build_Win32.bat* nach *c:\IB_BUILD_DIR* und die Hilfsprogramme in Ihr Windows-Verzeichnis (i.A. *c:\WINNT*) oder an einen anderen Ort auf Ihrer Festplatte, der über den eingestellten PATH erreichbar ist.

Es sind außerdem noch einige weitere Hilfsmittel notwendig, die alle Bestandteil von Borlands C-Compiler sind. Der eigentliche C-Compiler wird zwar nicht verwendet, dafür aber *make.exe* von Borland. Auch wenn Microsofts VC++ als Compiler dient, verwendet man bei Borland doch das hauseigene make-Tool zum Steuern des Compilers. Ferner greift der Build-Prozess von InterBase auch noch auf *brc32.exe* (Borland Resource Compiler) und *implib.exe* (Generierungstool für Import-Libraries) zurück. Alle diese Werkzeuge finden Sie zum Beispiel bei C++Builder 5 im *\Bin Verzeichnis*, oder auch in der frei erhältlichen Borland C++ Compiler Version 5.5 [9]. Sie müssen also nicht unbedingt eine Lizenz des C++Builders erwerben. Stellen Sie sicher, dass auch das *\Bin*-Verzeichnis Ihres Borland C-Compilers in PATH aufgeführt ist, und zwar vor allen anderen Pfaden zu Produkten, die andere Make-Versionen enthalten könnten (z. B. *Delphi\Bin*). Definieren Sie PATH und andere Variablen am besten nicht mit SET für ein einzelnes Konsolenfenster sondern systemweit (SYSTEMSTEUERUNG>SYSTEM>UMGEBUNGSVARIABLEN), damit die Einstellungen dauerhaft erhalten bleiben.

Henne oder Ei?

Um InterBase übersetzen zu können, brauchen Sie InterBase. Überlassen wir es den originären Entwicklern, sich über dieses Paradoxon den Kopf zu zerbrechen, und akzeptieren es einfach als eine Tatsache. Sie können eine beliebige Version von InterBase 6 oder Firebird verwenden (siehe Kasten). Und wieder müssen Sie die Auflistung Ihrer PATH-Systemvariablen erweitern, und zwar um das *\Bin*-Verzeichnis Ihrer InterBase-Installation (i.d.R. *C:\Programme\Borland\InterBase\Bin*), damit die Build-Batches Tools wie *isql.exe*, *gbak.exe* oder *gpre.exe* von beliebiger Stelle aufrufen können.

Aber der InterBase-Server wird auch noch anderweitig gebraucht. Während

InterBase 6.0x Erscheinungsformen

Borlands Open Source Builds

Unter www.borland.com/devsupport/interbase/opensource werden Binärfiles angeboten, die mit Januar 2001 datieren, also inzwischen nicht mehr ganz aktuell sind. Die Dateien sind ausdrücklich als *unzertifiziert* gekennzeichnet.

Borlands zertifizierte Builds

Zertifizierte, also intensiv getestete und für gut befundene Versionen einschließlich Dokumentation, gibt es nur gegen Lizenzgebühr. Die Preise entsprechen ziemlich genau denen der Vorgängerversion 5.x. Trials sind unter www.borland.com/interbase/downloads verfügbar. Derzeit aktuell: V6.0.1.6. Ein Update auf V6.5 wird noch für dieses Jahr erwartet.

Täglich frische Borland Builds von MER Systems

Jede Nacht führt ein Rechner des kanadischen Borland-Vertriebspartners MER Systems Inc. automatisierte Builds aus dem Borland-Quellenbaum durch. Die Binärfiles kann man herunterladen von mers.com. Tip: Die aktuelle „Tagged Version" (V6.0.1.6) entspricht wohl weitgehend der zur Zeit aktuellen zertifizierten Version von Borland und hat sich bereits gut bewährt.

Reinrassige Open Source Alternative „Firebird"

Firebird, eine eigenständige Abspaltung von den InterBase-Quellen, stellt hierfür wohl die einzig wirklich lebendige Entwickler-Community dar. Firebird ist schon aus Prinzip und auch aufgrund von Lizenzbestimmungn nicht-kommerziell. Die aktuellen, zur Borland-Version kompatiblen Binärfiles (V0.9.x) sind verfügbar bei www.ibphoenix.com/ibp_downloads.html. Weitere Links, Infos und Resourcen (auch und besonders für Entwickler) ebenfalls bei www.ibphoenix.com.

des Build-Prozesses wird der Preprozessor *gpre.exe* intensiv verwendet, um Quelldateien mit der Endung *.e* in C-Quellfiles zu übersetzen und dabei Embedded-SQL-Anweisungen in C-Code umzuwandeln. Dabei muss *gpre* sich mit der Zieldatenbank verbinden, um Zugriff auf die Tabellenstrukturen zu erhalten. Hauptsächlich geht es dabei um die Systemtabellen (mit dem Prefix RDB$). Die wichtigste Build-Datenbank heißt sinnvollerweise METADATA.GDB. Von ein paar Besonderheiten abgesehen, ist dies eigentlich eine völlig leere Datenbank, die eben nur die Systemtabellen enthält. Wenn man – anders als wir – nicht in der komfortablen Lage ist, die METADATA.GDB *für* InterBase 6 *mit* InterBase 6 erstellen zu können, dann müssen die evtl. neu hinzukommenden System-Metadaten hier „von Hand" hinzugefügt werden. Diese schwere Prüfung brauchen wir aber nicht abzulegen, denn andere haben dieses Henne-Ei-Problem schon für uns gelöst. Eine weitere Datenbank, die während des Builds benötigt wird, ist MSG.GDB (alias *master_msg_db*), in der von den Entwicklern Meldetexte gespeichert worden sind, die extrahiert und zu der Datei *interbase.msg* umgeformt werden. Weitere Build-Datenbanken betreffen die Benutzerdatenbank, das Hilfesystem für QLI, oder sind für die Beispielanwendungen von Bedeutung.

Für den Build-Prozess werden zwei Benutzerkonten benötigt: „SYSDBA" (der Standard-Benutzername des Datenbankadministrators) mit dem Standardkennwort „masterkey" (also entgegen sonstiger Gewohnheit nicht ändern) und ein Benutzer namens „builder" mit gleichlautendem Kennwort „builder". Eine einfache Möglichkeit, „builder" zu definieren, ist die Verwendung des Konsolenmodus-Programms *gsec.exe*:

```
C:\>gsec –USER SYSDBA –PASS masterkey
GSEC> add builder –PW builder
GSEC> quit
```

Die Build-Datenbanken können Sie entweder lokal auf Ihrem Build-Rechner oder auf einem anderen Server, der über LAN erreichbar ist, platzieren. Ich gehe hier vom erstgenannten Fall aus. (Wenn Sie aber die Build-Datenbanken doch auf einen Remote-Server legen, dann müssen Sie auch dort die oben genannten Benutzer einrichten.) Legen Sie ein Verzeichnis *c:\IB_BUILD_DB* mit den in Abbildung 2 gezeigten Unterverzeichnissen an. Auch die einzelnen Dateien müssen Sie von Hand an Ort und Stelle kopieren.

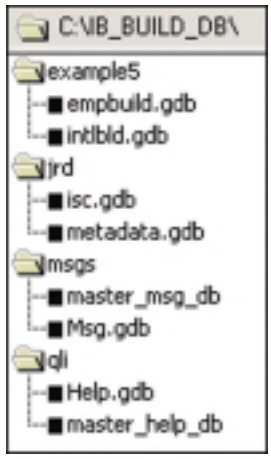

Abb. 2: Die Build-Datenbanken

Die Dateien *empbuild.gdb* und *intlbld.gdb* sind ganz einfach umbenannte Kopien der Demodatenbanken *employee.gdb* bzw. *intlemp.gdb*. Sie kopieren diese beiden Dateien von dem fertig installierten InterBase 6 Server und benennen sie, wie in der Abbildung 2 gezeigt, um. Beide Dateien finden Sie in oder unterhalb des Ordners INTERBASE\EXAMPLES (der exakte Ort variiert zwischen den in Umlauf befindlichen Versionen). Auch *isc.gdb* ist nichts anderes als einfach eine umbenannte Kopie aus der installierten Interbase-Version, und zwar von *isc4.gdb* aus dem InterBase-Stammverzeichnis. Die Datei *metadata.gdb* brauchen Sie nicht selbst zu erstellen; sie wird beim Build-Prozess automatisch erzeugt.

Die Dateien *msg.gdb* und *help.gdb* müssen Sie aus Backup-Dateien erstellen, die im CVS-Baum enthalten sind. Bei der Borland-Version finden Sie diese als *builds\original\dbs\msgs\msg.gbk* bzw. *builds\original\dbs\qli\help.gbk*. Bei der Firebird-Variante sind dies die Dateien *misc\msg.gbak* bzw. *misc\help.gbak*. Es ist wichtig, dass Sie das Restore dieser Backups unter dem Benutzer „builder" durchführen. Nachfolgend die notwendigen Kommandos in beiden Varianten (siehe Listing 2). Sie sollten die Build-Datenbanken in *IB_BUILD_DB\msgs* und *IB_BUILD_DB\qli* aus dem CVS-Baum heraus erstellen, den Sie auch kompilieren wollen. Es ist nämlich zum Beispiel durchaus möglich, dass die Firebird-Version von *msg.gdb* einige Meldetexte enthält, die der Borland-Version fehlen oder umgekehrt.

Jetzt fehlen nur noch die Dateien *master_msg_db* und *master_help_db*. Diese sind nichts anderes als Kopien von *msg.gdb* bzw. *help.gdb*. Während des Build-Prozesses werden diese Datenbanken manchmal unter dem einen, dann wieder unter dem anderen Namen angesprochen. Eigentlich wäre das doppelte Vorhandensein dieser Dateien unnötig, aber für den Anfang ist es einfacher, die Kopien anzulegen, als die Build-Skripte zu optimieren.

Auf der Zielgeraden

Jetzt trennen uns nur noch wenige Schritte von dem ersten Versuch eines Build-Laufes. Wir benötigen nur noch die Umgebungsvariable INTERBASE. Diese muss auf das Stammverzeichnis der installierten InterBase-Version zeigen, also z. B. *INTERBASE=c:\Programme\Borland\InterBase*.

Und nun kommen wir zum eigentlichen Ereignis: Wechseln Sie in einem Konsolenfenster in das Verzeichnis *c:\IB_BUILD_DIR* und starten Sie unseren Build-Batch exakt wie folgt:

```
C:\IB_BUILD_DIR>IBFB_Build_Win32.bat localhost:
        c:/IB_BUILD_DB >IBFB_Build_Win32.log 2>&1
```

Der Batch erwartet als Parameter den Pfad zum Verzeichnis der Build-Datenbanken, und zwar in der bei InterBase üblichen Notation *server:laufwerk:pfad*. Achtung: Der Pfad muss in diesem Fall unbedingt mit vorwärtsgerichteten Schrägstrichen (/) statt mit Backslash (\) eingegeben werden!

Wenn alles glatt läuft, sollten sich am Bildschirm zwei weitere Konsolenfenster öffnen, in denen der Build-Prozess seine Aktivitäten durch eifrige Textausgaben unter Beweis stellt (siehe Abb. 3). Diesen Trick vollbringt der Build-Batch übrigens durch zweimaligen Aufruf des Prograämmchens *tail.exe*. Tail gibt dann permanent die letzten Zeilen der *.log*-Dateien aus, deren es zwei gibt (daher auch zwei Ausgabefenster), nämlich *IBFB_Build_Win32.log* und *build_lib.log*. Die erstgenannte enthält die Ausgabe des von Ihnen aufgerufenen Build-Batches, also hauptsächlich das Protokoll der nötigen Vorarbeiten wie das Erzeugen von Verzeichnissen und Kopieren von Dateien. Die letztere ist wesentlich umfangreicher und enthält das Protokoll der abgearbeiteten Make-Files, also die Ausgaben des eigentlichen Compilers. *Build_lib.log* weist eine große Zahl von Compilerwarnungen, aber keine Fehler auf. Dies vorausgesetzt, wird am Ende von *IBFB_Build_Win32.log* irgendwann die Anweisung ECHO SUCCESS ausgegeben, mit der uns zu einem erfolgreich verlaufenen Build gratuliert wird.

Richtfest

Die Früchte Ihrer Arbeit finden Sie im Verzeichnis *IB_BUILD_DIR\interbase*. Die Unterverzeichnisse und Dateien sind Ihnen wahrscheinlich geläufig. Was Sie vorfinden, entspricht weitgehend der Verzeichnisstruktur eines ganz normal installierten InterBase-Servers. Um die Kompilate zu testen, können Sie z. B. das Verzeichnis des bereits installierten InterBase-Servers umbenennen und den neu kreierten Pfad an dessen Stelle setzen, wobei der InterBase-Server natürlich beendet werden muss. Besondere Aufmerksamkeit müssen Sie noch der Datei *gds32.dll* widmen. Diese finden Sie einerseits im neuen *\bin*-Verzeichnis, andererseits auch noch – als Relikt der Original-Installation – in *WINNT\System32*. Benennen Sie die *gds32.dll* in System32 einfach um; dadurch wird die neue Version in *interbase\bin* verwendet. Wenn Sie einen neuen Build-Lauf starten wollen, sollten Sie dies mit Hilfe der Original-Version von InterBase tun, d.h. Sie machen die vorgenannten Umbenennungen wieder rückgängig. Am einfachsten ist es aber, die selbst gebaute Datenbanksoftware auf einem separaten Rechner zu testen, und sich so das komplizierte Hin und Her zu ersparen.

Listing 2

```
REM —- Dies ist die Borland-Variante —-
C:\IB_BUILD_DIR>cd builds\original\dbs\msgs
C:\..\msgs>gbak –R –USER builder –PASS builder msg.gbk \IB_BUILD_DB\msgs\msg.gdb
C:\..\msgs>cd ..\qli
C:\..\msgs>gbak –R –USER builder –PASS builder help.gbk \IB_BUILD_DB\qli\help.gdb

REM —- Und hier die Firebird-Variante —-
C:\IB_BUILD_DIR>cd misc
C:\..\misc>gbak –R –USER builder –PASS builder msg.gbak \IB_BUILD_DB\msgs\msg.gdb
C:\..\misc>gbak –R –USER builder –PASS builder help.gbak \IB_BUILD_DB\qli\help.gdb
```

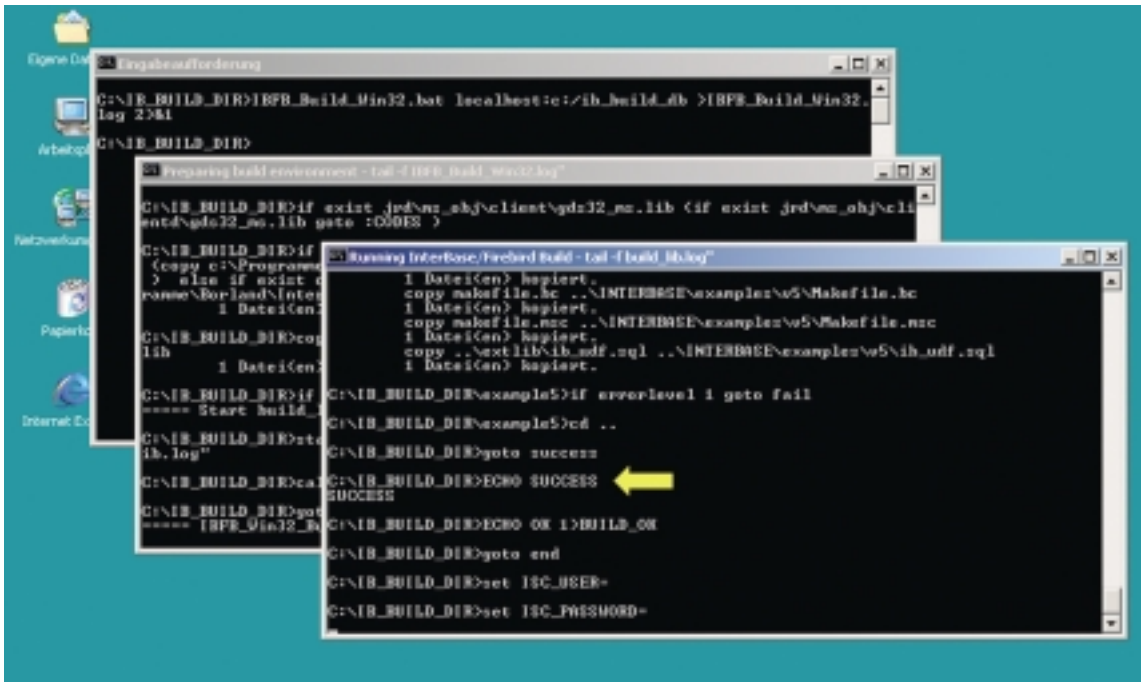

Abb. 3: Der Build-Batch in Aktion

Wenn Sie den Build-Batch zweimal hintereinander aufrufen, so werden Sie feststellen, dass der ganze Vorgang wesentlich schneller geht. Das Make-Tool wiederholt nämlich nicht jeden Compilerlauf, sondern beschränkt sich darauf, geänderte Quellcodes neu zu übersetzen. Falls Sie dies vermeiden möchten, können Sie Ihren Build-Baum nach getaner Arbeit wieder lichten, d.h. die Kompilate, Objekt- und Debugdateien wieder löschen, und zwar mit:

```
C:\IB_BUILD_DIR>IBFB_Build_Win32.bat cleanup
```

Einsichten eines Kammerjägers

Mit der zuvor beschriebenen Vorgehensweise erstellen Sie ein sogenanntes *produktives Build*, d.h. Sie erzeugen optimierte InterBase-Binärdateien ohne Symbolinformationen, die Sie (vielleicht) als Datenbankserver einsetzen können. Warum sollten Sie sich aber die Mühe machen, eigene Binärdateien aus den offenen Quellcodes zu erzeugen, wenn Sie solche – einschließlich Installationsdateien – von verschiedenen Internet-Seiten herunterladen können?

Möglicherweise wollen Sie sich aber auch tiefer in die Architektur der InterBase- bzw. Firebird-Quellcodes einarbeiten, und vielleicht sogar am Wirken der Community teilnehmen. Dann wollen Sie wahrscheinlich auch die Möglichkeit haben, den InterBase-Server zu debuggen. Dazu müssen Sie den Build-Prozess allerdings mit Debug-Option starten, indem Sie den Build-Batch mit dem Schalter "–DDEV" starten:

```
C:\IB_BUILD_DIR>IBFB_Build_Win32.bat localhost:
  c:/IB_BUILD_DB –DDEV >IBFB_Build_Win32.log 2>&1
```

Auf diese Weise erzeugen Sie debug-fähige Binärdateien, einschließlich der notwendigen Program-Database-Dateien (.pdb). Die Ausgabe erfolgt diesmal in das Verzeichnis *IB_BUILD_DIR\ib_debug*. Um dieses Build zu aktivieren, gehen Sie genauso vor, wie oben schon für das produktive Build beschrieben, also ersetzen Sie den installierten InterBase-Server durch Ihre selbst erzeugten Dateien und stellen Sie sicher, dass ausschließlich mit der neuen *gds32.dll* gearbeitet wird.

Wenn Sie InterBase mit der IDE von MSVC++ debuggen wollen, starten Sie den Server zunächst als Applikation (nicht als NT-Dienst), indem Sie an der Kommandozeile den Schalter *–A* mit angeben (*ibserver.exe –A*). In der Startleiste zeigt der Prozess jetzt ein Trayicon an. Starten Sie Visual C++, ohne ein Projekt oder einen Arbeitsbereich zu öffnen, und wählen Sie über den Menüpunkt ERSTELLEN | DEBUG STARTEN | VERBINDEN MIT PROZESS... den Eintrag *ibserver*. Beenden Sie jetzt den InterBase-Prozess auf die normale Art (Menüfunktion SHUTDOWN am Trayicon). Sobald Sie jetzt in der IDE eine Ausführungs-Funktion wählen (F5 für Programm starten oder F10/11 für die Ausführung Schritt-für-Schritt), wird *ibserver.exe* unter der Kontrolle des Debuggers ausgeführt. Zuvor müssen Sie allerdings über das Menü PROJEKT | EINSTELLUNGEN in *Debug* als Programmargument „-A" einstellen.

Leider steht Ihnen keine Projektdatei zur Verfügung, die die einzelnen Quellcode-Dateien lokalisiert. Aber Sie können sich hier recht einfach selbst behelfen: Führen Sie InterBase schrittweise aus (F10). Sobald eine Quelldatei erforderlich ist, fordert Sie die IDE mit einer Art Datei-öffnen-Dialog automatisch auf, diese manuell aufzufinden. Suchen Sie die fragliche Datei (der gesuchte Name wird jeweils angezeigt) im Quellcode-Baum aus. Die Quelldatei wird geöffnet und schon stehen Ihnen alle Möglichkeiten des symbolischen Debuggers zur Verfügung. Wenn Sie einige Quelldateien auf-

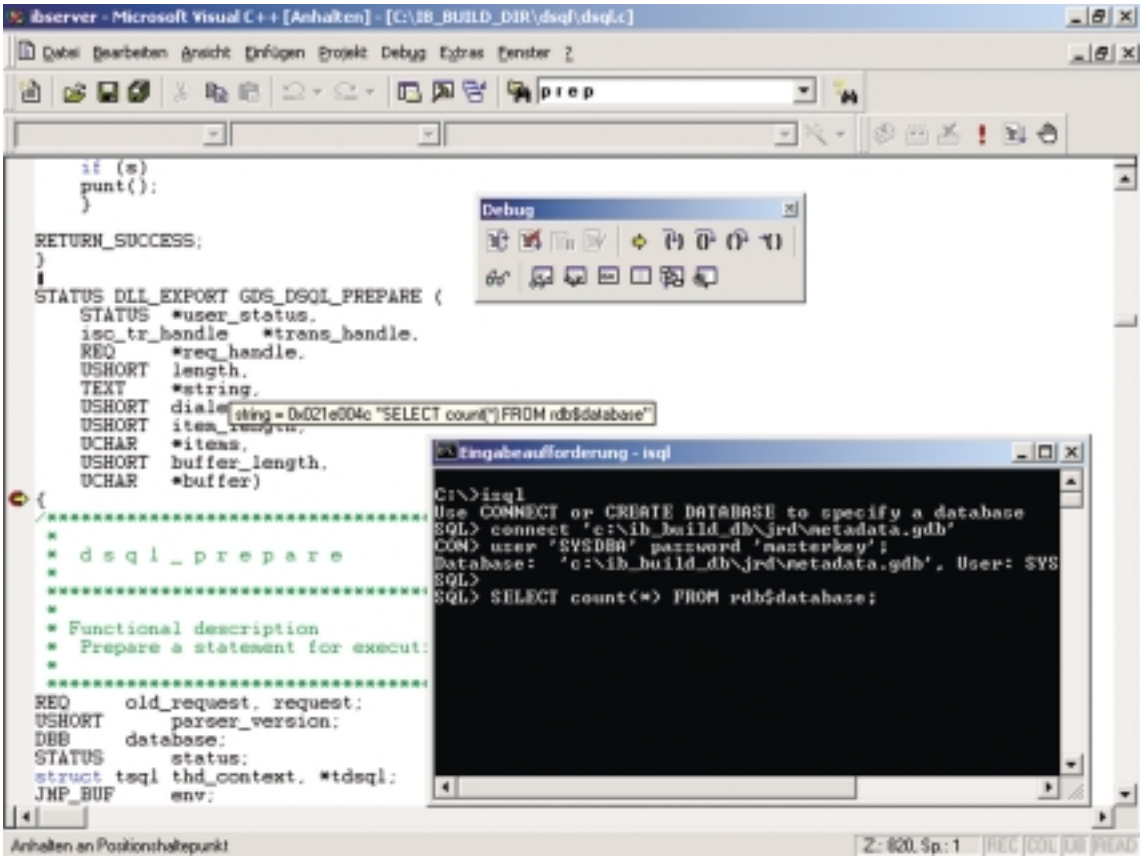

Abb. 4: InterBase unter Debugger-Kontrolle

gesucht haben, dann können Sie diese Informationen mit der Menüfunktion DATEI | ARBEITSBEREICH SPEICHERN sichern, womit Sie vermeiden, dass Ihnen diese Detektivarbeit nach dem nächsten Starten der IDE erneut abverlangt wird, denn Sie können Ihre Festlegungen mit DATEI | ZULETZT GEÖFFNETE ARBEITSBEREICHE wieder herstellen.

Üblicherweise stellt man beim Debuggen eine bestimmte Situation her, und lässt den Debugger auf diese mit Hilfe von Breakpoints lauern. Abbildung 4 zeigt beispielhaft, wie in der Datei *dsql.c* ein Breakpoint beim Eintreten in die Funktion GDS_DSQL_PREPARE ausgelöst wird, sobald mit einem Datenbankclient (hier *isql* im Konsolenfenster) eine SQL-Anweisung ausgeführt wird.

Fazit

Jenseits aller Faszination – wozu kann es Ihnen nützen, Ihr eigenes Build der InterBase- bzw. Firebird-Datenbank zu besitzen? Ich möchte nicht dazu raten, ein selbsterstelltes Build für ein reales Datenbankprojekt einzusetzen, es sei denn, man verfügt bereits über reiche Erfahrung als Entwickler in diesem Open-Source-Projekt. Andere Builds, die im Internet frei verfügbar sind und die auf den selben Quellcodes basieren, sind besser getestet bzw. bereits bewährt und außerdem sind sie mit einer Setup-Lösung ausgestattet. Borland verfügt über eine enorm umfangreiche Testfall-Bibliothek, die erfahrensten Entwickler und hohe Qualitätsstandards, und betreibt großen Aufwand beim Entwickeln und Erstellen der zertifizierten InterBase-Versionen, die dafür allerdings nicht kostenlos sind. Aber auch andere, allen voran die Firebird-Leute, legen sich bei der Qualitätssicherung ins Zeug.

Die Fähigkeit, mit den Quellcodes dieser Datenbank selbst jonglieren zu können, soll Sie also vor allem in die Lage versetzen, als (Co-) Entwickler tätig zu werden. Das kann (muss aber nicht unbedingt) heißen, eigene kleine Teilprojekte oder Bugfixes zu übernehmen, das Ergebnis der Community zur Verfügung zu stellen und so in künftige Versionen einfließen zu lassen, womit man nicht zuletzt die Prioritäten der eigenen Wünsche an die weitere Entwicklung dieses Produktes sehr effektiv zu beeinflussen vermag. Man kann sich aber natürlich auch darauf beschränken, den Quellcode als eine Art Nachschlagewerk zu gebrauchen, etwa um Hintergrundwissen für spezielle, wenig dokumentierte Lösungen (z. B. BLOB-Filter oder UDF-Parameter über Deskriptoren) zu gewinnen oder ganz allgemein einen tieferen Einblick in die Arbeitsweise der Software zu erhalten.

Links & Literatur

[1] www.borland.com/devsupport/interbase/opensource/IPL.html
[2] Andreas Bleul, Herbert Schmid, Versionskontrolle, in: c't 11/01
[3] http://cvsbook.red-bean.com/translations/german
[4] www.interbase2000.de/misc/src_overview.htm
[5] www.ibphoenix.com/ibp_c_compiler.html
[6] http://msdn.microsoft.com/vstudio/sp/vs6sp5/default.asp
[7] www.gnu.org
[8] http://unxutils.sourceforge.net
[9] www.borland.com/bcppbuilder/freecompiler