

Writing your first Rocket U2 Application

Brian Leach

Writing Your First UniVerse or UniData Application

Brian Leach

About this Book

This book was written for the International U2 User Group as a contribution to their Learner Pack.

The book is available in published form from Brian Leach (www.brianleach.co.uk) or as a **free download** in PDF format from the U2UG website (www.u2ug.org)

About the Author

Brian Leach is an independent U2 and MultiValue consultant based in the UK. Brian has worked with UniVerse since 1989 and has designed some of the most advanced development and reporting tools in the U2 market.

Brian is currently serving as Past President of the International U2 User Group, of which he was a founder board member.

You can contact the author at www.brianleach.co.uk.

Table of Contents

About this Book	4
About the Author	4
Introduction	7
A Note about UniVerse and UniData	8
About the International U2 User Group	9
Thank You	9
Acknowledgements	10
Chapter 1 Welcome	11
What This Guide Will Teach You	12
Who Should Read This Guide	13
Scene Setting – The UniVerse and UniData Platforms	14
Some Bad Naming - UniVerse, UniData and "U2"	15
Getting Up and Running	15
Downloading the Personal Editions	16
Installing the Database Server	17
Installing Dynamic Connect	18
Installing the Middleware	19
Installing the Demonstration Database	19
Chapter 2: Exploring the Database	21
Connecting to the UniVerse or UniData Command Shell	22
Logging Into the UniVerse or UniData Environment	24
Entering the Account Path	25
The Command Shell	26
Disconnecting Safely	28
Exploring the U2 Architecture	29
Account Structure	30
Account Syntax	31
Moving between Accounts	32
Files	33
Listing the Files in your Account	33
The Demonstration Files	34
Navigating through Listings	34
Listing File Content	35
Records and Keys	35
The Default Listing	36
Files and Dictionaries	36
Listing a File Dictionary	37
Listing Fields	38
Virtual Fields	39
Related Fields	40
XML Listings	40
MultiValued Data	42
Inside the Record Structure	43
Chapter 3: Starting your Application	44
Starting your Application	45
Installing Visual Basic.NET	46
Introducing UniObjects	48
Starting an Application	49

Adding the Namespace	50
Creating the Book Titles Form	52
Creating the Connection	53
Closing the Connection	55
Adding the Title Controls	56
Reading and Writing Records	57
Opening a File	58
Reading a Record	59
Dynamic Arrays	59
Triggering a Read Event	61
Locking Records	63
Writing Changes	64
Adding New Records	64
Running Commands and Selections	65
The UniCommand Object	65
The UniXML Object	66
Adding a Browse List	67
Selecting the Title from the Grid	68
Chapter 4: Server Coding	70
UniVerse Server Coding	71
Server Side Programming	71
Creating and Running Programs	72
Your First Server Program	73
Compiling your Program	74
Running your Program	75
Examining the Syntax	76
Reading Data	77
Using Dynamic Arrays	79
Using Constants	81
Creating a UniBASIC Subroutine	82
Calling a Subroutine from the Server	85
Chapter 5: Integrating Server Code	87
Calling Subroutines from the Client	88
The UniSubroutine Object	89
Adding the Subroutine Call to your Code	90
Server side updates	92
Server Side Validation	94
Calling Internal Subroutines	95
Adding Locking	98
Variable Scope and Persistence	100
Creating a Common Block	100
Placing File Variables in Common	101
Conclusion	103
Next Steps	103
From the Author	104

Introduction

This guide is written for developers and students who are new to the UniVerse and UniData business platforms from Rocket Software. Its aim is to get you up and running with your first UniVerse or UniData project, to whet your appetite to learn more about this exciting environment and to answer the question we are asked most frequently – just where do I start?

The UniVerse and UniData platforms are deep and complex environments. They contain a bewildering set of features: a MultiValue database for information storage and retrieval, customized enquiry languages, reporting features, spooler and device interfaces, and their own rich application language covering everything from developing robust business applications to network and systems management operations.

Beyond these you will discover a range of APIs that allow you to use UniVerse and UniData to power client/server and web applications, support for web services and XML, gateways for ODBC, OleDB and JDBC, message queues, SOAP/JSON and HTTP services and an open architecture that participates fully as consumer and provider in service oriented solutions.

And to top it off, a 40 year legacy that stretches back to the earliest days of database systems and a terminology that is entirely its own. Small wonder that new developers often find it hard to get a purchase!

This guide is not intended to teach you all about the UniVerse or UniData platform. Instead, it will show you some of the major features and guide you step by step through creating your first UniVerse or UniData application. At the end, you should have sufficient oversight to be able to delve into each of the areas of interest in more depth, and to understand where to look as you take the next steps.

A Note about UniVerse and UniData

UniVerse and UniData are sister products. Like all sisters, they share many characteristics, have a shared history but are also jealously independent and can squabble from time to time.

But take heart - whilst there are many syntactic and internal differences, most of the material in this guide and the understanding of the fundamental concepts behind these products will be just as applicable to its younger sibling.

Over recent releases, the two databases have become closer by sharing the same middleware, so the interaction between UniVerse and a .NET client, described here, is broadly the same for UniData.

With the release of UniVerse 11, the engineers have added some key technologies from the UniData stable, paving the way for even closer coherence between these two platforms.

About the International U2 User Group

The document has been produced for the International U2 User Group (U2UG), a volunteer not-for-profit organization formed to assist new and existing UniVerse and UniData technology users.

The U2UG promotes this database model through a range of services including technical forums, email lists, a knowledge base, enhancements committee, wiki and incubator projects.

You can find more articles, tutorials, resources and technical materials on the U2UG website:

<http://www.u2ug.org>

Thank You

Many thanks to Susan Joslyn of SJPlus (www.sjplus.com) and to Glenn Sallis for proof reading and offering deeply insightful comments.

My thanks to my fellow board members for their assistance, enthusiasm and positive criticism in preparing this.

Brian Leach

Acknowledgements

UniVerse, UniData and UniObjects are registered trademarks of Rocket U2.

PICK and Pick Systems are registered trademarks of Raining Data Corporation.

Access Query Language, AQL, D3 and FlashBASIC are trademarks of Raining Data Corporation.

UNIX is a registered trademark in the United States of America and other countries, licensed exclusively through X/Open Company Ltd.

Microsoft, Windows, Windows NT, Windows 2000 and MS-DOS are trademarks of Microsoft Corporation. ActiveX, JScript, Visual Studio, and Visual Basic are either registered trademarks or trademarks of Microsoft Corporation.

Adobe and Acrobat are trademarks of Adobe Systems Incorporated.

All other company or product names mentioned are trademarks or registered trademarks of their respective trademark holders.

The example companies, organizations, products and domain names depicted herein are fictitious. No association with any real company, organization, and product or domain name is intended or should be inferred.

Chapter 1 Welcome

Contents:

Who should read this Guide
What are the UniVerse and UniData Platforms?
UniVerse, UniData and 'U2'
What This Guide Will Teach You
Installing UniVerse Personal Edition
Installing the Demonstration Database

Objectives:

This chapter introduces the UniVerse and UniData business platforms and the tools that you will need to install to complete the application in this guide.

What This Guide Will Teach You

In this guide you will learn how to create a simple Windows .NET based application that will allow you to present and update information held in a UniVerse or UniData database.

Along the way, you will be given an overview of the platform architecture, visit the command shell, stop off to admire the file system and take a detour around the enquiry language.

You will also be given a guided tour around some of the prominent features of the main application server language, and pointed to various sources of additional information.

When you have finished, you will have gained an insight into the development process and the most popular API for building UniVerse and UniData applications.

Who Should Read This Guide

Creating a general guide for a complex technical system involves a number of choices. Do we include everything you might need to know and produce a book of several hundred pages that would take weeks to wade through, or something brief and pithy that falls short of telling some essential details and leaves the reader feeling lost and confused?

This guide is written for experienced programmers, who are encountering the UniVerse or UniData platform for the first time. It is written by developers, who understand that you will want to get your hands dirty at the code face at the first opportunity. We will of course stop to explain certain features as we go along, but we won't have too many detours along the route.

You will hopefully be familiar with modern languages like C#, java or Visual Basic.NET. If not, you will still be able to complete this guide but may find some of the concepts in the later chapters need some additional reading. Fortunately the .NET platform is well documented and there is a wealth of books that can guide you in your learning.

You may even be familiar with developing data driven applications using mainstream three tier or 'n' tier data models. If so, you may be pleasantly surprised at how much easier UniVerse or UniData development can be.

The guide does not presuppose that you have any more than general knowledge of databases and of database design. If you are a database architect of distinction, please approach this with an open mind – much of what you will encounter in this guide may challenge the design principles you have learned elsewhere.

Finally, this guide should prepare you to be able to research further on your own. For this reason, we will skim the surface of the languages and features where deeper information is readily available, and concentrate instead on the big picture of how to make sense of it all.

Scene Setting – The UniVerse and UniData Platforms

Databases come in all shapes and sizes and there are many distinctions to be drawn between them. UniVerse and UniData are often categorized as *embedded databases*, which differ from the more familiar database servers in wrapping their database features within a complete application development environment.

Embedded databases typically include one or more high level languages for generating business applications, customized reporting features, enquiry facilities, device control, printing and sometimes even systems administration commands. These are generally hidden away from their users, who see only the applications that they are running. Embedded databases are also typically light, fast, robust and require minimum administration: all of which are important since the database should be invisible.

It follows that embedded databases are also free to adopt data models that are different and more complex than those used by regular database servers, since their designers may be given the latitude to come up with advanced facilities that are specifically targeted at their own data models rather than having to worry so much about compatibility and adherence to mainstream industry standards. This can often lead to more efficient or scalable processing, but at the cost of having to learn new techniques of working.

For example, embedded databases rarely need the sort of three tier architectures or persistence backbones that have become common for mainstream database server based applications. This makes development quicker, easier and cheaper.

The UniVerse and UniData platform belongs to a family of embedded databases also known as *MultiValue Databases* (MVDBMS). Rocket Software is one of the largest suppliers of MultiValue Database systems, but there are equivalents available from other suppliers including open source implementations. Multivalue databases are found in every sector, often running large scale or highly complex applications such as trading and financial systems, local and central government, freight and ERP systems. There are some close parallels between the MultiValue platform and the NoSQL databases that have become popular for very large websites.

To those of us who work across a number of database models, MultiValue offers a compelling mixture of capability and performance – and its own quirks.

Some Bad Naming - UniVerse, UniData and "U2"

When you start to read up about the UniVerse or UniData platform, you will inevitably see the name 'U2' used. This does not refer to the rock group, but to a product family (courtesy of IBM, with congratulations to the marketing division for choosing such an easily searchable and unique name).

Rocket Software owns two database platforms that are both based on the same MultiValue model, and both of which were purchased from IBM: UniVerse and UniData. Both are very similar and spent much of their existence as competing products from separate suppliers.

Getting Up and Running

Now that is enough scene setting for now— time to get started!

To complete building the application in this guide, you will need to install a number of pieces of software. You can choose to load these directly onto a PC or laptop, or you might want to load them into a virtual machine such as the (free) Microsoft Virtual PC or VMWare Workstation. You can install both database on the same PC, but that leads to complications and is not advised.

Rocket provides Personal Editions of the U2 databases that can be used for training and evaluation.

In order to complete the application, you will need the following:

- UniVerse or UniData Personal Edition
- UniObjects Middleware
- Microsoft Visual Basic .Net
- Wychbooks Demonstration Database

You may also find it helpful to download the free mvDeveloper Personal Edition from www.brianleach.co.uk. This is a free and simple to use Windows based U2 editor available from the author's website; or you can use the Basic Developers Toolkit supplied with the database client.

Downloading the Personal Editions

The easiest way to learn UniVerse or UniData is to install your own local copy onto a Windows based PC or virtual machine. UniVerse and UniData both run on Windows, Linux and various forms of UNIX, but to keep this guide as simple as possible all of the examples will assume that you are running on Windows.

NOTE: You will need to have local administrator rights on that PC to perform the installation and to make the initial connections to UniVerse or UniData.

To download a Personal Edition of the database and tools, first open up your preferred web browser, and navigate to the U2 section of the Rocket software web site at:

<http://www.rocketsoftware.com/u2>

On the opening page you will see a tab for Resources, which will lead you to the Downloads section, which may look similar to the page below.

The screenshot shows the Rocket Software U2 Downloads page. The page has a navigation bar with links to Products, Solutions, Success Stories, Support, Training, Resources, How To Buy, and About Us. A search bar is also present. The main content area is titled 'Downloads' and lists various software packages. The list is organized into a table with columns for Name, Version, Type, Expires, Size, and Operating system.

	Name	Version	Type	Expires	Size	Operating system
1.	Basic Developer Toolkit (BDT) Supports UniData 7.2 and UniVerse 10.3 or higher. For information on how to run with earlier versions see Technote 1392476.	1.2.1	Free	N/A	286 MB	Windows
2.	UniVerse Clients for Windows Supports UniVerse 10.3 and 11.1	11.1	Free	N/A	602 MB	Windows
3.	UniVerse Personal Edition for Windows*	11.1.1	PE	31 July 2012	30 MB	Windows
4.	UniVerse Personal Edition for Linux	11.1.1	PE	31 July 2012	155 MB	Windows
5.	wintegrate 6.2.x (60-day trial)	6.2.x	Trial	11 Feb 2011	28 MB	Windows
6.	UniData Personal Edition for Linux*	7.2.7	PE	31 Mar 2012	100 MB	Linux
7.	UniData Clients for Windows	7.2A	Free	N/A	542 MB	Windows
8.	UniData Personal Edition for Windows*	7.2.7	PE	31 Mar 2012	48 MB	Windows

From the search list above you will be offered a number of choices for downloading UniVerse or UniData. You will need to download two packages:

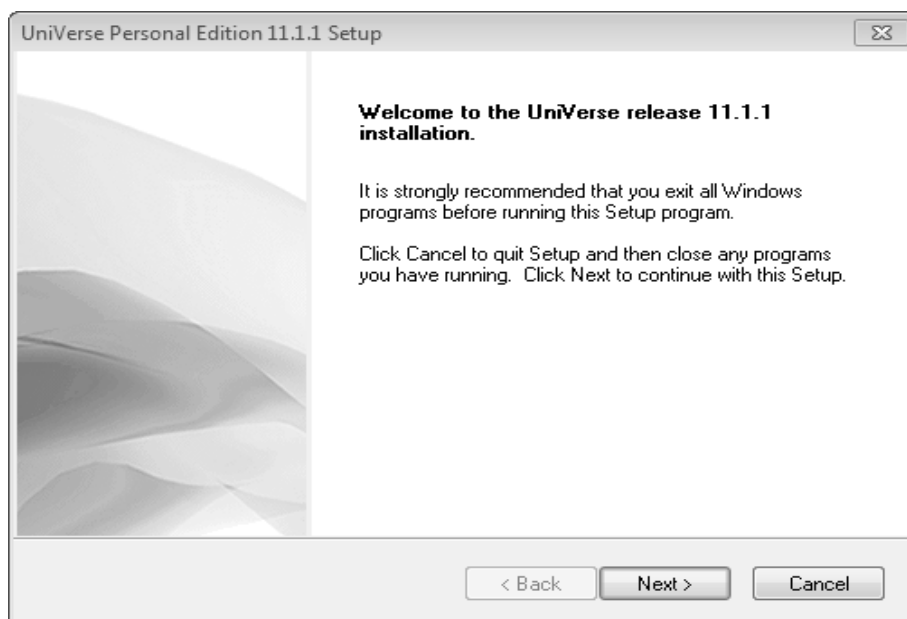
- UniVerse or UniData Personal Edition for Windows
- Clients Package for UniVerse or UniData

You will need to sign up on the Rocket website to download these editions. At time of writing this just leads to a welcome email.

Installing the Database Server

Once you have downloaded the two packages, you can proceed to install the UniVerse or UniData server.

You will need to log into your PC using the credentials of a user who has local administrator rights. To install UniVerse onto your PC, open up the UniVerse Personal Edition package and find the file named AUTORUN.EXE. This will display an installation menu from which you can select the option **UniVerse RDBMS** or **UniData RDBMS** to run the setup and follow the defaults throughout.

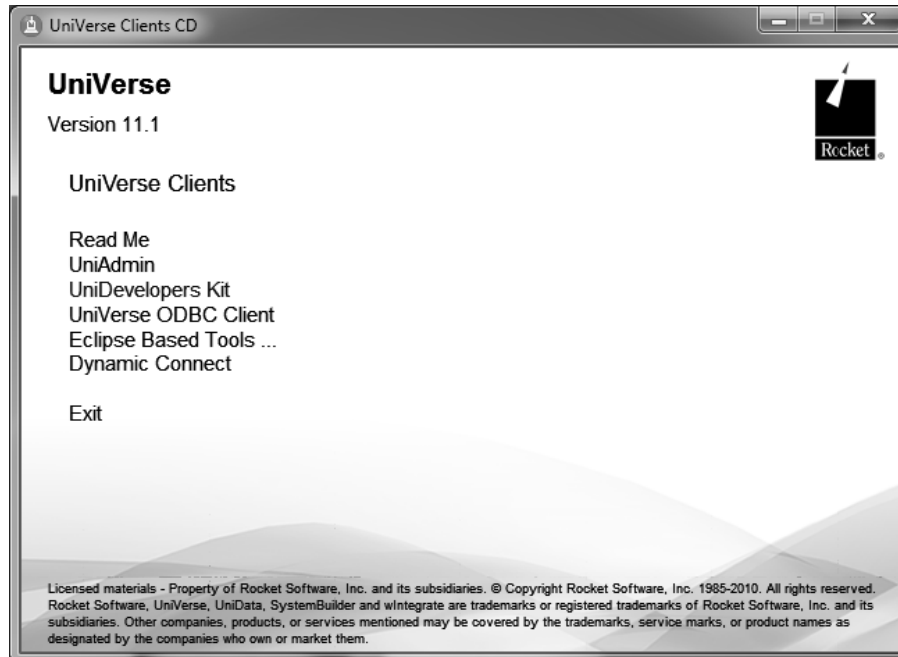


Do not select the NLS (National Language Support) or UVNET options for UniVerse: they are beyond the scope of this guide and require additional configuration.

Installing Dynamic Connect

The UniVerse and UniData Client downloads are zip files that contain a number of tools and middleware for working with UniVerse or UniData systems.

Amongst these tools you will find the Dynamic Connect terminal emulator. We will be using the terminal interface to start exploring the platform in the first part of this guide.



Select the **Dynamic Connect** option to run the setup, and once again just select the default options throughout.

Installing the Middleware

For the final part of your UniVerse or UniData setup you will need to install a piece of middleware that will allow a client application such as Visual Studio to talk to your copy of UniVerse or UniData.

The U2 platforms offer a range of middleware, both standard and proprietary. The middleware we will be using for this guide is called the UniObjects Developers Kit, or UniDK, and is found in the same Client download packages as Dynamic Connect.

Select the **UniDK** option from the installation menu to run the setup.

Select all of the options to install and just accept the defaults throughout. You will only be using the UniObjects.NET version for this guide, but you can find guides for using the other features on the U2UG Knowledge Base and the U2UG Incubator.

There is one final step before you can begin this course: installing the demonstration database onto your new UniVerse or UniData server.

Installing the Demonstration Database

This guide will be teaching you to build a UniVerse or UniData application, but you won't be starting from scratch. The information you will be using is to be found in the WYCHBOOKS demonstration database. This may be downloaded from the author's website at:

<http://www.brianleach.co.uk>

If you are downloading a copy from the website, make sure that you download the correct version - this will be one of:

- Wychbooks ZIP file for UniVerse Personal Edition.
- Wychbooks ZIP file for UniData Personal Edition.

The WYCHBOOKS demonstration database is designed to show the features of U2 applications, and is used for a number of publications including this guide, self paced learning courses by the author, and the tutorials and help documents to be found on the U2UG Knowledge Base.

To install the training database, simply unzip the content of the WYCHBOOKS_PE.zip package into a directory, preferably named C:\wychbooks. You can in fact locate a UniVerse or UniData database anywhere on your file system but this will keep the path short and consistent.

The database will be ready to use straight away, so in the next chapter you will begin your exploration of the UniVerse or UniData platform.

Chapter 2: Exploring the Database

Contents:

- Connecting to UniVerse or UniData*
- The Command Shell*
- Disconnecting Safely*
- Accounts*
- Account Flavor*
- Files*
- Listing File Content*
- Dictionaries*
- Listing Dictionaries*
- Dictionary Definitions*

Objectives

This chapter will teach you how to connect to (and safely disconnect from) a UniVerse or UniData system using a terminal emulator and how to use the command shell.

This chapter will then familiarize you with the fundamental architecture of a U2 database application, and will introduce the ways in which UniVerse and UniData store information. You will learn how to find files, list their content and describe the information available. You will also learn how to list real and virtual fields.

Connecting to the UniVerse or UniData Command Shell

UniVerse and UniData applications, like most other applications, are generally client/server or web based. The application you will be creating in the later part of this guide will be a .NET driven Windows Forms application.

But before you begin writing your application, you will need to discover some of the fundamental aspects of your database, especially where it differs from the sort of databases you may have used in the past.

And when it comes down to learning how the database works, the command shell is the best way to navigate around a UniVerse or UniData database.

The command shell connects you to a UniVerse or UniData session, typically using a TELNET or TELNET/SSL connection. For this you need a terminal emulator, such as the Dynamic Connect emulator you installed in chapter 1.

So to begin, start Dynamic Connect from your Programs menu.

If this is the first time you have run Dynamic Connect, the New Session Wizard will start automatically. If not, select the Session Wizard option from the File menu:



Click the **Next** button and on the next page, select **Windows Sockets** as the transport.

On the next page you will be asked for the name of the host system to which you wish to connect. Enter the name '**localhost**' to refer to the local machine:



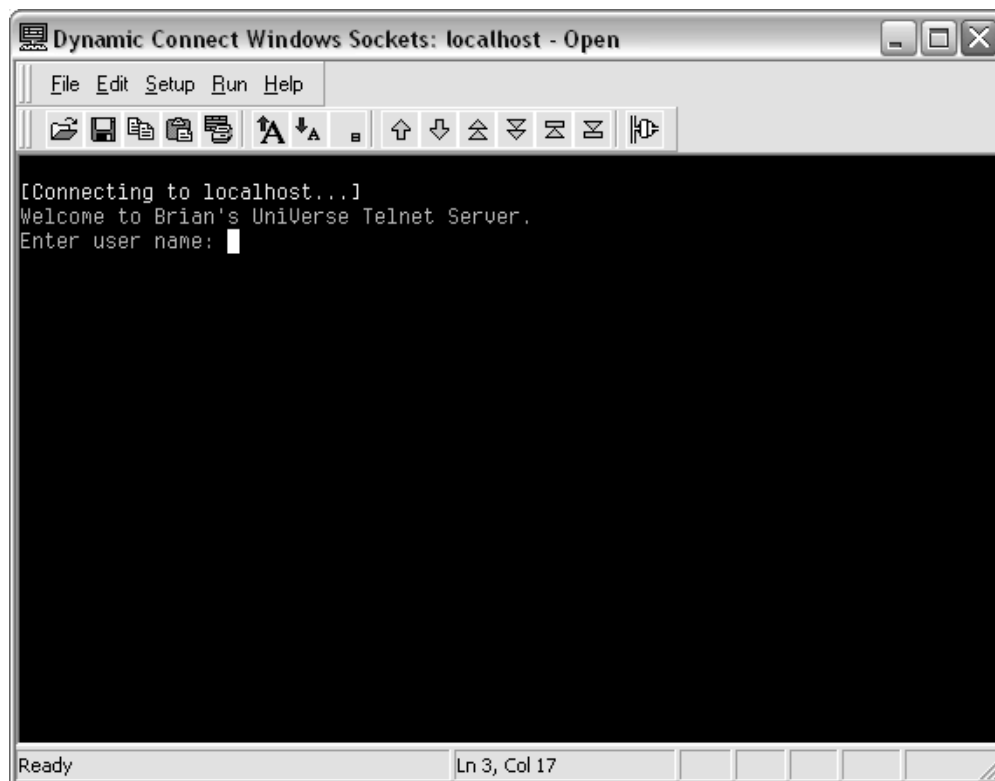
Click the **Next** button to run through the rest of the pages in the Session Wizard. You can just accept all of the defaults for these until you reach the final page.

Click the **Finish** button to start the connection.

Logging Into the UniVerse or UniData Environment

Any database or application environment needs to verify user credentials and to restrict access. Initially, UniVerse and UniData both leave the authentication up to the operating system. Secure sites can add certificate based authorisation, endpoint verification and encryption to provide extra layers of security - but that is beyond the scope of this guide.

When you connect to UniVerse or UniData using Dynamic Connect, you will be asked for a user name and password. Both will always allow the local administrator to connect, so for now enter the name and password for a user who has local administration rights on your PC.



Non-administrator users can connect to UniVerse, but need to be explicitly declared before UniVerse will allow them entry.

Entering the Account Path

A live UniVerse or UniData server may typically run a number of separate applications, or may be storing different sets of information used by different people or departments.

To keep each of these applications apart, and to make administration simpler, these are organized into a number of separate working areas, which for historical reasons are known as 'Accounts'. These are roughly analogous to working with different databases or schemas in other models.

Whenever you are working inside the UniVerse or UniData environment, you are placed in an account. The UniVerse TELNET server will ask which account you wish to enter once you have entered your user name and password. The UniData telnet will automatically place you in the UniData demo account.

UniVerse:

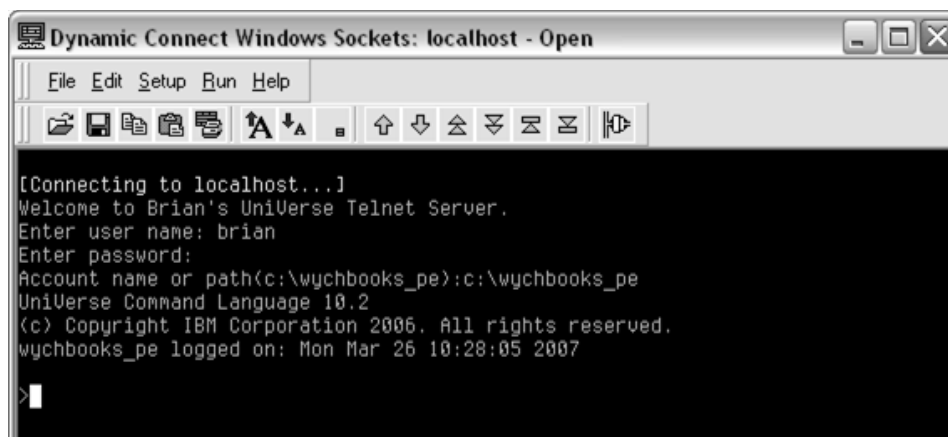
Enter the pathname where you installed the demonstration database, for example **C:\wychbooks\wychbooks_pe** at the *Account Name or Path* prompt. This will take you into the command shell in the demonstration account.

UniData:

After connecting you will be automatically placed into the demo account. To move to the demonstration database you will need to issue the command:

logto c: \wychbooks_pe

*or wherever you placed the demonstration database.



You may see a prompt asking you to update the account to match your current version of the database. If so, enter a Y at the prompt and let the update proceed.

The Command Shell

The Command Shell provides a text based interface to a UniVerse or UniData database. The Command Shell lets you type commands and see the response immediately (similar to the CMD shell under Windows). This makes it the best place to learn how the database works, but it may seem a little alien if you are more accustomed to graphical, point and click environments.

The Command Shell presents a prompt in the shape of a greater-than sign (>) on UniVerse or a colon (:) on UniData. This is known as the TCL (Terminal Control Language) or ECL (English Control Language) prompt. Now for some good news - most UniVerse and UniData commands are English-like, and generally make sense!

Type the command **who** and press the enter key. Don't worry about whether it appears in upper or lower case: the database will answer with a response similar to the one below:

UniVerse:

```
>WHO
1292 wychbooks_pe From brian
```

The WHO command identifies your command session. The first value (1292) is your user number: this is a unique session identifier that keeps track of your connection. An administrator can use this number if they need to disconnect you, send you a message or to release any resources that you have left behind.

The second value (wychbooks_pe) tells you in which account you are working. It is important to check this before you run any other commands so that you know that you will be working with the correct information.

The third piece of information (brian) reports your user name. You can open several connections using the same user name and password, and once again this is mainly for the benefit of administrators.

NOTE

If you are running UniVerse, you may find that your keyboard operates in a different case when you type in the command shell.

This is an idiosyncratic feature left behind by the original UniVerse developers - as most UniVerse commands are in upper case and most UNIX commands in lower case, they chose to invert the case whenever working in UniVerse. This is frankly annoying to the rest of us.

You can turn this off by typing the command:

PTERM CASE NOINVERT

UniData:

```
:who  
demo 192.168.163.128 09:22:24 May 02 2011
```

The UniData **who** command displays your user name and the IP address from which you are connecting. If several users are logged on, the who command will list these.

You can cross reference this with the **listuser** command which provides some more useful additional information:

```
:listuser  
  
UDTNO  USRMBR   UID  USRNAME   USRTYPE  TTY  IP-ADDRESS  
1       2176   197608  demo      udt      pts/1 192.168.163.128
```

UniData keeps two identifiers: a 'UniData' number (UDTNO) and a User number (USRNBR). The UDTNO is a sequentially assigned number assigned to each UniData process. The USRNBR is the process id assigned to that session by the operating system.

Either of these can be used to uniquely identify a command session, and different administration commands expect different identifiers. An administrator can use these numbers to perform tasks such as clearing locks. Also, different applications may elect to use one or the other for tasks such as creating unique workfiles or reporting individual user sessions: the UDTNO is, as the name suggests, specific to UniData where the user number is portable between UniData and UniVerse.

The **where** command tells you in which account you are working:

```
:where  
C:\IBM\uc72\demo
```

It is important to check this before you run any other commands so that you know that you will be working with the correct information.

Disconnecting Safely

When you have finished with the database, it is important that you disconnect safely. This will ensure that any resources you have used are released and decrement the license count for the number of concurrent users.

If you do not disconnect safely from the command shell, you may be blocked from connecting again until you reboot because the database believes you are still connected. The Personal Editions allow you to run two concurrent sessions.

UniVerse:

UniVerse provides two commands for disconnecting safely, the OFF and QUIT commands. These will terminate your session and close your connection.

UniData:

For UniData, you should use the QUIT command.

Type the **OFF** or **QUIT** command to finish your session and close the connection in Dynamic Connect.

Exploring the U2 Architecture

Now you know how to connect and disconnect safely, you can use the command shell to discover the architecture of the database. The rest of this chapter will introduce you to the main architectural features and to the information available in the demonstration database.

The U2 model uses an architecture that differs from the relational databases you may have used. It also uses different terminology, so we will introduce some of the terms – that way when you encounter them in the user guides you will have an understanding of what they mean.

There is a lot to cover so we will cover each of the main elements as briefly as possible. As you work through the exercises and build your application in the next chapter, some of these will come more clearly into focus.

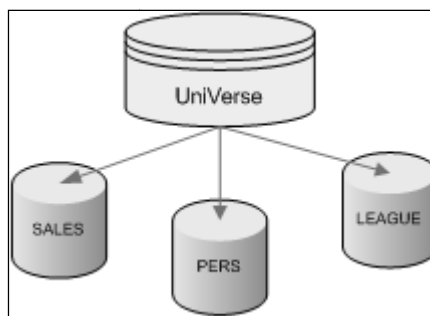
One frequent criticism that is often levelled at the U2 documentation is that it always seems to describe the system to people who already understand it.

Account Structure

At the very top level, a U2 system is divided into a number of separate areas known as Accounts. An Account is simply a work space that provides access to a subset of the information and procedures available in a system. An account might hold one particular application, various sets of data and/or a number of stored procedures. A UniVerse account may also contain a SQL database schema (UniVerse offers a hybrid MultiValue/Relational model, but that is outside the scope of this guide).

The boundaries that define an Account are actually quite porous – an Account can directly reference entities that are held in other Accounts, so the rules for how a system should be divided into accounts are, like many things where the U2 platform is concerned, flexible and open to interpretation.

Generally an Account contains a set of related information and functions that together make up part or all of an application. For example, a typical system might include three accounts: the first holding an order processing system, the second holding a personnel application, and a third holding football scores for the company league:



Physically, each account resides in a separate directory in the underlying file system. The demonstration database you have installed in Chapter 1 is a typical example.

You can decide which entities can be visible from another account. In the example above, the order processing application might need to refer to the personnel files to record the name of the person responsible for taking an order or to record a sale against their targets, but should not be able to see other personnel information - such as workers' salaries or directors' expenses!

Account Syntax

Just as SQL is fractured into many competing dialects today (SQL*Plus is not compatible with TSQL, for example), so different MultiValue database manufacturers have introduced differences into their syntax over the years. UniVerse and UniData both found ways to support these variations.

Fortunately, as with SQL, there is a base line of generic functionality that works across most such systems and only just stops short of doing anything genuinely useful. Fortunately also the differences generally fall into a number of 'families'.

UniVerse:

UniVerse supports these differences by giving each account a 'flavor', which determines how it interprets certain commands.

You can find the account flavor by typing:

.L RELLEVEL

UniData:

UniData approaches this differently from UniVerse. In UniData the emulation is decided by various settings that can be modified at run time. The advantage is that you can change emulation part-way through an application, if a particular emulation better suits a specific task. The disadvantage is exactly the same, as it may be more difficult to support a system where that happens.

The overall UniData command set is controlled by the ECLTYPE command. This switches between two major command modes – ECLTYPE P (for PICK) or ECLTYPE U (for UniData and PRIME).

For this course we will be using

ECLTYPE P

You can also fine-tune the emulation by setting various environment options using the UDT.OPTIONS command, but I will leave you to look those up and play with them at your leisure.

Moving between Accounts

You can navigate between accounts by using the LOGTO or CHDIR command. This moves to another account and runs any initialization code for that account.

UniVerse:

Every UniVerse system has an administration account called 'uv'. To move to this account, type the command:

LOGTO uv

You can use the **WHO** command to verify that this worked.

UniData:

You have already navigated between accounts, when you used the LOGTO command to move from the default demo account to the wychbooks demonstration database.

The LOGTO command accepts the pathname to an account.

LOGTO c:\wychbooks\wychbooks_pe

You can learn more about accounts in the UniVerse System Administration Guide and the UniVerse System Description Guide, or in the Using UniData Guide. All the UniVerse and UniData documentation can be downloaded (free) from the Rocket website.

Files

UniVerse and UniData store data and programs in structures called Files.

The File structure is more flexible than a standard database table, and you can think of a file as being very similar to a filing cabinet in an office. A filing cabinet typically contains many records: each set of cabinets may be dedicated to holding just one type of record (customer details, invoices, despatch notes or similar) or may hold several different types stuffed together into the same drawers.

UniVerse and UniData files work in the same way. Each file stores any number of pieces of data organized into *records*. Just like the paper forms in a filing cabinet, these records generally contain similar information - but unlike most other database models, they do not have to.

A single file may hold hundreds of thousands or millions of records. Just like any good filing system, the records are organized in such a way that the database can retrieve any particular record instantly.

Listing the Files in your Account

Because the databases hold all of their data in files, the first stage when looking around an unfamiliar database is to discover which files are visible in your account.

To see the files in the demonstration account, type the command **LISTFILES** or **LISTF** for short:

LISTF

The LISTFILES command displays a listing of all of the files that are directly accessible from your account. The listing will include any files that have been created in that account, and certain files that have been made shared from other accounts.

The Demonstration Files

You will see a group of files all prefixed with the word BOOK. These are the demonstration files that you will be using for the rest of the exercises in this guide.

```
BOOK_AUTHORS  
BOOK_COMMENTS  
BOOK_READERS  
BOOK_SALES  
BOOK_SUPPLIERS  
BOOK_SUPPLIES  
BOOK_TITLES
```

This demonstration database represents an audio bookshop.

The audio book details are held in the BOOK_TITLES file, author details in the BOOK_AUTHORS file, and sales orders in the BOOK_SALES file. The rest should be equally obvious.

Navigating through Listings

The LISTF command displays one page at a time.

At the bottom of each page, you will see the message:

```
Press any key to continue.
```

Or

```
Enter <New line> to continue ...
```

For most commands where this prompt appears you have three choices:

- Press the Enter key to display the next page.
- Type Q (short for QUIT) to stop the list at that point, and to return to the command prompt. This prevents you having to page through the rest of the listing.
- Type N (short for NOPAGE) to turn off the paging for this listing. This will then run through the rest of the listing without pausing.

Listing File Content

When you start looking through the list of files, the names that appear may or may not give a clue to the information that the files are holding. So the next step will be to start looking through some of the file content.

Each record in a file is divided into a set of *fields*.

You can get a *summary* view of the information held in a file by using the command:

```
LIST filename
```

where filename is the name of the file to be listed.

Type the following command to see the Book Titles:

```
LIST BOOK_TITLES
```

LIST BOOK_TITLES	PAGE	1
Key.. Title.....		Author Name.....
10 Hancock a Comedy Genius (BBC Radio Collection)		CAST
11 I'm Sorry I Haven't a Clue: Vol 8 (BBC Radio Collection)		CAST
12 Friends, Lovers, Chocolate		ALEXANDER MCCALL SMITH
13 The Legend of Spud Murphy		EOIN COLFER
14 Farmer Giles of Ham and Other Stories		J. R. R. TOLKIEN
15 The Lord of the Rings: Complete & Unabridged		J. R. R. TOLKIEN

Records and Keys

When you use the LIST command, you will see an additional column to the left of the fields you request. This is the record key, or @ID field.

Each record in a UniVerse or UniData file must have a unique identifier, or record key. The database uses the key when storing and retrieving records.

Applications can work directly with record keys to give a very fast and light response. For this reason, developers generally choose meaningful keys: a customer account number, a short descriptive word, a date.

In the next chapter you will learn to work directly with record keys in your application.

The Default Listing

When you use the LIST command in this way, it shows you a summary view.

Specifically, the LIST command shows a set of fields termed the "default listing".

The information that appears in the default listing is decided by the developer or administrator responsible for the file. U2 records typically hold a large amount of information – for various reasons these records are often much larger than those in relational databases – and may easily run to hundreds of fields. So the default listing just shows a sample of the most useful columns.

If you list a file and see only a single column, this means the developer has not created a default listing for you and so the database is just showing the record keys. You will need to discover the contents by another means.

Files and Dictionaries

MultiValue databases have a number of features that set them apart from the rest of the database world. One of the most important of these is the File Dictionary.

Each file is made up of two separate entities:

- a *Data* section that holds the records.
- a *Dictionary* section that describes the content.

The File Dictionary holds *metadata* ("data about data") that describes the content of the file. This metadata is the key to understanding the database.

The BOOK_TITLES file, for example, contains records defining each audio title in the bookstore catalogue. Each audio title record contains fields that hold a short title of the book, the author and reader, the ISBN number, the genre, department, price and stock level, as well as the media format, publisher and allocation.

How do you find out about these? By looking in the Dictionary of the BOOK_TITLES.

Listing a File Dictionary

The file dictionary is always referenced using the name:

```
DICT filename
```

You can see the content of the dictionary in the same way that you listed the content of the data file: by using the LIST command

```
LIST DICT filename
```

Type the following command to view the dictionary of the BOOK_TITLES file:

```
LIST DICT BOOK_TITLES
```

DICT BOOK_TITLES Page 1						
Field.....	Type &	Field.....	Conversion..	Column.....	Output	Depth &
Name.....	Field.	Definition...	Code.....	Heading.....	Format	Assoc..
	Number					
TITLE_ID	D	0		Id	5R	S
@ID	D	0		Key	5R	S
TITLE_NAME	D	1		Title	50T	S
SHORT_TITLE	D	1		Title	30T	S
AUTHOR_ID	D	2		Author	5L	S
READER_ID	D	3		Reader	5L	M
TYPE	D	4	MCU	Type	10L	S
UNITS	D	5	MD0	Units	5R	S
FORMAT	D	6	MCU	Format	5L	S
ISBN	D	8		Isbn	10L	S
PRICE	D	9	MD2	Price	10R	S
DEPT	D	11	MCU	Dept	10L	S
GENRE	D	12	MCU	Genre	15L	S
PUBLISHER	D	13		PUBLISHER	20L	S
PUB_YEAR	D	14		PUB_YEAR	4R	S
ALLOCATED	D	15		ALLOCATED	3R	S

The dictionary holds various pieces of information about the file content.

At the start of each dictionary listing are the field definitions. These are identified by a 'D' (standing for Data) in the Type column of the list.

Listing Fields

Once you know what fields are available, you can use the familiar LIST command to display those columns from the data file by using the syntax:

```
LIST filename field field...
```

Type the following command to see the title next to the ISBN:

```
LIST BOOK_TITLES SHORT_TITLE ISBN
```

```
LIST BOOK_TITLES SHORT_TITLE ISBN PAGE      1
Key.. Title..... Isbn.....

 10 Hancock a Comedy Genius (BBC   0563525452
    Radio Collection)
 11 I'm Sorry I Haven't a Clue:    0563495421
    Vol 8 (BBC Radio Collection)
 12 Friends, Lovers, Chocolate     1405500530
 13 The Legend of Spud Murphy       0141805293
 14 Farmer Giles of Ham and Other   0001056107
    Stories
 15 The Lord of the Rings:          0007192614
    Complete & Unabridged
```

UniVerse:

If you are running UniVerse and are more comfortable using SQL, you can perform the same command using a UniVerse SQL SELECT statement:

```
SELECT @ID, SHORT_TITLE, ISBN FROM BOOK_TITLES;
```

UniVerse supports two enquiry languages: a native enquiry language called Retrieve and a specialized variant of SQL. Both use the file dictionaries and they can be used largely interchangeably.

Most UniVerse users prefer Retrieve which is more English-like and has a more friendly syntax, and steer away from SQL.

Virtual Fields

Dictionaries go further than merely defining the file content. The real power of the dictionary lies in some of the other definitions – in particular the use of virtual fields.

Virtual fields are based on calculations or expressions. These are used in reports and enquiries in exactly the same way as real fields, so that once a developer has created them there is an easy and consistent means of accessing derived information. Unlike the column expressions that make up part of the query syntax in other databases, these are stored as a fixed part of the data definition and can contain complex expressions: there is a whole expression language including functions, operators, calls to stored procedures and the like.

Look further down the listing for the dictionary of the BOOK_TITLES and you will find another set of definitions, this time identified with an “I” (Interpreted) or “V” (Virtual) in the Type column. These are the calculated fields.

The SHORTFALL field, for example, reports the difference between the number of units in stock and the number that have been ordered. Other files in the demonstration database have virtual fields that define pricing calculations, tax, line totals, order balances and similar financial operations. “Real” databases may have fields that perform complex operations, retrieve data from remote databases, perform translations – in short, all kinds of useful things.

Type the following command to view the units in stock, allocated stock and shortfall (calculated):

```
LIST BOOK_TITLES SHORT_TITLE UNITS ALLOCATED SHORTFALL
```

Virtual fields can also be used to combine data for selection or to reformat data for display and selection. The DEPT_GENRE virtual field simply combines the DEPT and GENRE to make selection simpler:

```
LIST BOOK_TITLES SHORT_TITLE DEPT GENRE DEPT_GENRE
```

Try listing out the other fields in the BOOK_TITLES dictionary.

Related Fields

You can list the name of the author beside each book title as follows:

```
LIST BOOK_TITLES SHORT_TITLE AUTHOR_NAME
```

But hold on - the AUTHOR_NAME is not part of the BOOK_TITLES: it is found in the BOOK_AUTHORS file. What's going on?

One particularly powerful feature of virtual fields is that they can be used to look up data from another file. This means a virtual field can be used to present data as if it were part of the file, when in reality it is held somewhere else. This makes database enquiries that need to span multiple files very much simpler!

The author name resides in the BOOK_AUTHORS file, which is related to the BOOK_TITLES using the AUTHOR_ID as a foreign key. But rather than using a complex enquiry statement or a view to join the two files together – as you would in SQL - the dictionary simply provides a virtual field called AUTHOR_NAME to perform the lookup – and hides away the structure of the database.

XML Listings

UniVerse Retrieve, UniVerse SQL and UniData UniQuery can all create XML recordsets. These can be captured and exported through various APIs for interchanging data with other systems, and as a convenient means of gathering data for web and graphical client applications. You will be using this technique in your application in the next chapter.

The **TOXML** keyword converts the listing into an XML format.

Type the command:

```
LIST BOOK_TITLES TOXML
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
<BOOK_TITLES _ID = "10" SHORT_TITLE = "Hancock a Comedy Genius (BBC Radio Collec
tion)" SHORT_AUTHOR = "CAST"/>
<BOOK_TITLES _ID = "11" SHORT_TITLE = "I'm Sorry I Haven't a Clue: Vol 8 (BBC Ra
dio Collection)" SHORT_AUTHOR = "CAST"/>
<BOOK_TITLES _ID = "12" SHORT_TITLE = "Friends, Lovers, Chocolate" SHORT_AUTHOR
= "ALEXANDER MCCALL SMITH"/>
<BOOK_TITLES _ID = "13" SHORT_TITLE = "The Legend of Spud Murphy" SHORT_AUTHOR =
"EOIN COLFER"/>
<BOOK_TITLES _ID = "14" SHORT_TITLE = "Farmer Giles of Ham and Other Stories" SH
ORT_AUTHOR = "J. R. R. TOLKIEN"/>
<BOOK_TITLES _ID = "15" SHORT_TITLE = "The Lord of the Rings: Complete & Una
bridged" SHORT_AUTHOR = "J. R. R. TOLKIEN"/>
<BOOK_TITLES _ID = "16" SHORT_TITLE = "The Navy Lark: Taking Some Liberties v. 1
```


You can convert this to an alternate format by adding the keyword ELEMENTS:

```
<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
<BOOK_TITLES>
  <_ID>10</_ID>
  <SHORT_TITLE>Hancock a Comedy Genius (BBC Radio Collection)</SHORT_TITLE>
  <SHORT_AUTHOR>CAST</SHORT_AUTHOR>
</BOOK_TITLES>
<BOOK_TITLES>
  <_ID>11</_ID>
  <SHORT_TITLE>I'm Sorry I Haven't a Clue: Vol 8 (BBC Radio Collection)</SHORT_TITLE>
  <SHORT_AUTHOR>CAST</SHORT_AUTHOR>
</BOOK_TITLES>
<BOOK_TITLES>
  <_ID>12</_ID>
  <SHORT_TITLE>Friends, Lovers, Chocolate</SHORT_TITLE>
  <SHORT_AUTHOR>ALEXANDER MCCALL SMITH</SHORT_AUTHOR>
</BOOK_TITLES>
```

In the next chapter you will use this feature to pull search data from the database.

We are nearly through our tour around the main database features, but there is one essential concept we still need to cover, and from which the database model gets its name: the use of multivalued data.

MultiValued Data

Most databases use flat representations of data - information is stored in two dimensional tables of columns and rows like a spreadsheet. In these systems if you want to hold something that is three dimensional, like a sales order with a number of separate item lines, you need to break this across two tables: one for the header information and one to hold each detail line. This is known as a 'Master-Detail' or 'Parent-Child' relation.

The problem with the flat design is that having pulled these rows apart, you then need to join them together again whenever you need to reassemble that sales order for display. This is a costly operation in terms of memory and processing power and, whilst academically useful as a model, is practically inefficient. Readers of an RDBMS disposition should note that Codd himself described this as a design, and not an implementation, model.

MultiValued databases let you to store three dimensional data much more simply, by allowing each field to hold more than one value - an arrangement that more closely matches the way information is handled in real life. The result is more like an XML document than a spreadsheet.

So a MultiValued database does not need to use two separate files to hold Master-Detail records, such as a sales order. Take a look at the BOOK_SALES file in the demonstration database.

LIST BOOK_SALES

```
BOOK_SALES. 13661*55800*1
Sale Date.. 26 MAY 2005
SURNAME.... MAGGS
FORENAME... CARL
TITLE_ID QTY PRICE GIFT Title..... Author Name.....
      49   1  9.99   1 Jingo                Terry Pratchett
      47   1 49.99   0 Harry Potter and the J.K. Rowling
                        Goblet of Fire (Book
                        4 - Unabridged 14
                        Audio Cassette Set)
      11   1 12.99   0 I'm Sorry I Haven't Cast
                        a Clue: Vol 8 (BBC
                        Radio Collection)
```

In the BOOK_SALES file, each sales order is held in a single record that contains both the header information – the customer details and the time and date of the order – and fields that contain multiple entries making up the individual item lines for the audio titles and their quantities. The LIST command helpfully presents the 'detail' fields side by side if at all possible.

As a matter of fact, MultiValued Databases can go a level further. Each field can contain multiple values, each of which can in turn contain multiple sub-values.

Inside the Record Structure

UniVerse and UniData records are stored as delimited text. Each record is held as a string, and divided into fields and values using special reserved marker characters. These marker characters have default values, but in UniVerse these can be adjusted for national language support:

Delimiter	ASCII Value	Known as
Field Mark	254	@FM
Value Mark	253	@VM
Subvalue Mark	252	@SVM

For example, a customer record containing a surname, forename and multivalued address might look like the one below, where ^ represents a field mark and] represents a value mark:

SMITH^JANE^1 HIGH STREET]LITTLETOWN]SMALLSVILLE

Actually storing these records requires a very complex storage mechanism ... but that is beyond the scope of this guide.

For now, we will leave the exploration of the UniVerse architecture and the LIST command, and begin the serious work of building your first application.

You can learn more about UniVerse enquiry in the UniVerse Guide to Retrieve, or by following one of the self paced learning guides by the Author. For UniData see the UniQuery Commands Reference and the Using UniData Guide.

Chapter 3: Starting your Application

Contents:

- Introducing UniObjects
- Creating your Application
- Reading and Writing Records
- Running Commands and Selections

Objectives

This chapter will introduce you to client/server development for UniVerse and UniData. You will install a copy of Visual Basic .NET and learn about the UniObjects middleware. From there you will build a simple Windows application to display and to update UniVerse and UniData records.

Starting your Application

So far you have explored the UniVerse or UniData platform through the command shell. This is a primitive but very useful interface to the database, and indeed there are still a surprising number of text based applications that run directly inside the command shell. Don't worry though – we will choose something a bit smarter for your first application!

UniVerse and UniData applications can be built from most modern development environments using a choice of different protocols: from java to Delphi, PHP to Flash. For your application we will choose one of the most popular environments for writing applications (at the time of this article): Microsoft .NET. Even if you go on to develop UniVerse applications using other languages and tools, the fundamentals of working with UniVerse and UniData will remain the same.

Installing Visual Basic.NET

For simplicity we will be using Visual Basic .NET. Despite the snobbery found amongst certain C# developers (and yes, I use C#) VB.NET is perfectly capable of building robust and powerful applications.

The self paced training works by the Author demonstrate both VB.Net and C#.

If you have access to a copy of Visual Studio 2010 or later, you can make full use of the facilities to generate your UniVerse or UniData application. If not, you can download the free Visual Basic Express Edition from the Microsoft web site, at:

<http://www.microsoft.com/express>



Select the Windows option from the Visual Studio Express home page and download Visual Basic from the Windows page:



You can also download the other Express components if you wish.

The express edition has a number of limitations but will give you all of the tools you need to write a simple UniVerse application for your own training.

Introducing UniObjects

Rocket U2 supports a number of different forms of middleware that can link UniVerse and UniData to the .NET framework environment. These range from OleDb and ODBC drivers that can be used – at a pinch – for compatibility with other products through to SOAP and JSON web services that can expose business rules and logic. But the most popular choice amongst UniVerse and UniData developers is the UniObjects family.

UniObjects (UO) is a fast, stateful middleware that encapsulates the native parts of a UniVerse or UniData application into a set of easy-to-use objects. UniObjects is available in various different flavours, each of which is built to follow the conventions of a particular language: initially it was provided as a set of COM objects for native Windows programming and scripting and it is now available in java and as a fully managed .NET library.

The nice thing about UO is that it simplifies the U2 model for developers who are new to U2 development, allowing you to dip your toes into the world of U2 development without getting bogged down in technicalities. It also smoothes out some of the differences between UniVerse and UniData by presenting them both through a single API.

Some of these Objects represent the parts of the U2 business platform that you have met already in the previous chapter:

Session Object	Connection to a U2 environment.
Command Object	UniVerse or UniData command shell.
File Object	UniVerse or UniData data file.
UniXML Object	Enquiry or SQL XML command.

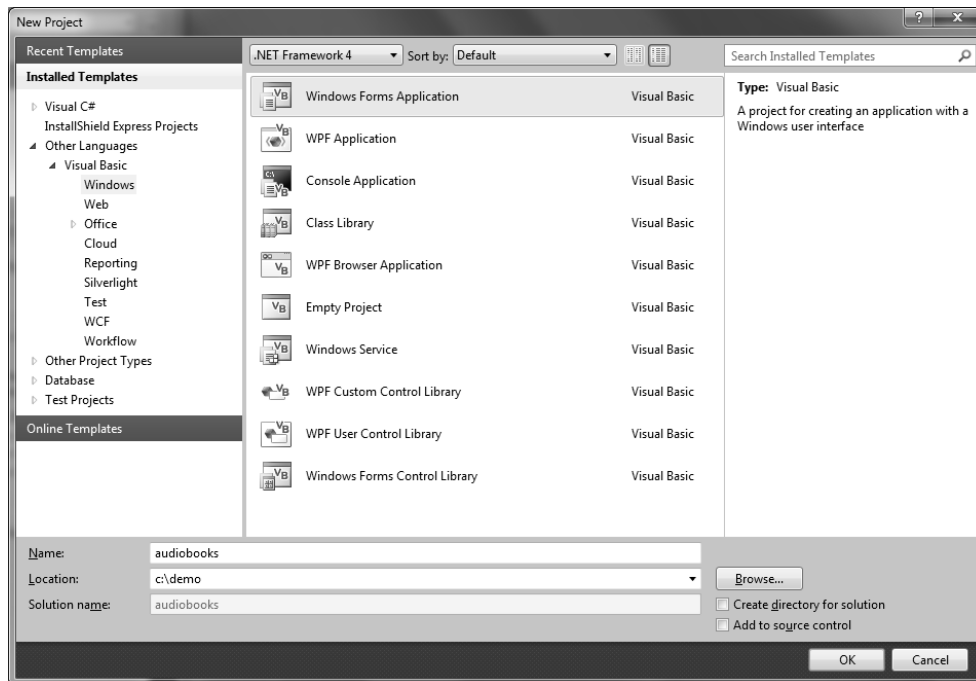
You can begin to use these immediately to start building your application.

Starting an Application

Start Visual Basic Express Edition or Visual Studio if you are fortunate enough to have a copy available, and select the **New Project** option from the **File** menu.

Visual Studio can create a very wide range of application types, ranging from Windows to web or mobile apps, most of which can work well with UniObjects. For this guide we are choosing a Windows Forms application – a regular Windows desktop application – as that is the easiest to get to grips with.

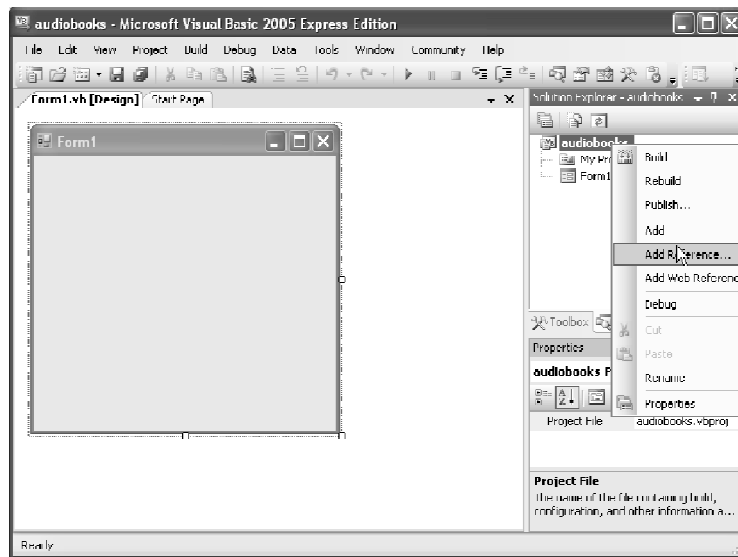
Choose a Visual Basic Windows Form application and name it **audiobooks**.



This will create the skeleton of a Windows application with a single default form.

In a few moments you can begin to populate this with some useful UniVerse information delivered through the UniObjects.NET middleware. Before you can do so, you will need to tell Visual Studio that you will be using UniObjects in your application: the easiest way to do this is to add a reference to the UniObjects library to your project.

The files and references that make up your application are listed in the Solution Explorer, usually found docked to one of the sides of your application window. Right click on the audiobooks folder in that window to display a context menu, as below:



Select the Add Reference... option from that menu and you may see the UniObjects library listed under the .NET Libraries tab. If not, you can find it by selecting the Browse tab and navigating to:

C:\U2\UniDK\uonet\bin\UODOTNET.dll

NOTE: if you are using an earlier version of UO.NET installed, it may be found under **C:\IBM\UniDK\uonet\bin\UODOTNET.dll**

Adding the Namespace

Before you go any further, you can save yourself a bunch of work by also adding the UniObjects namespace to your project.

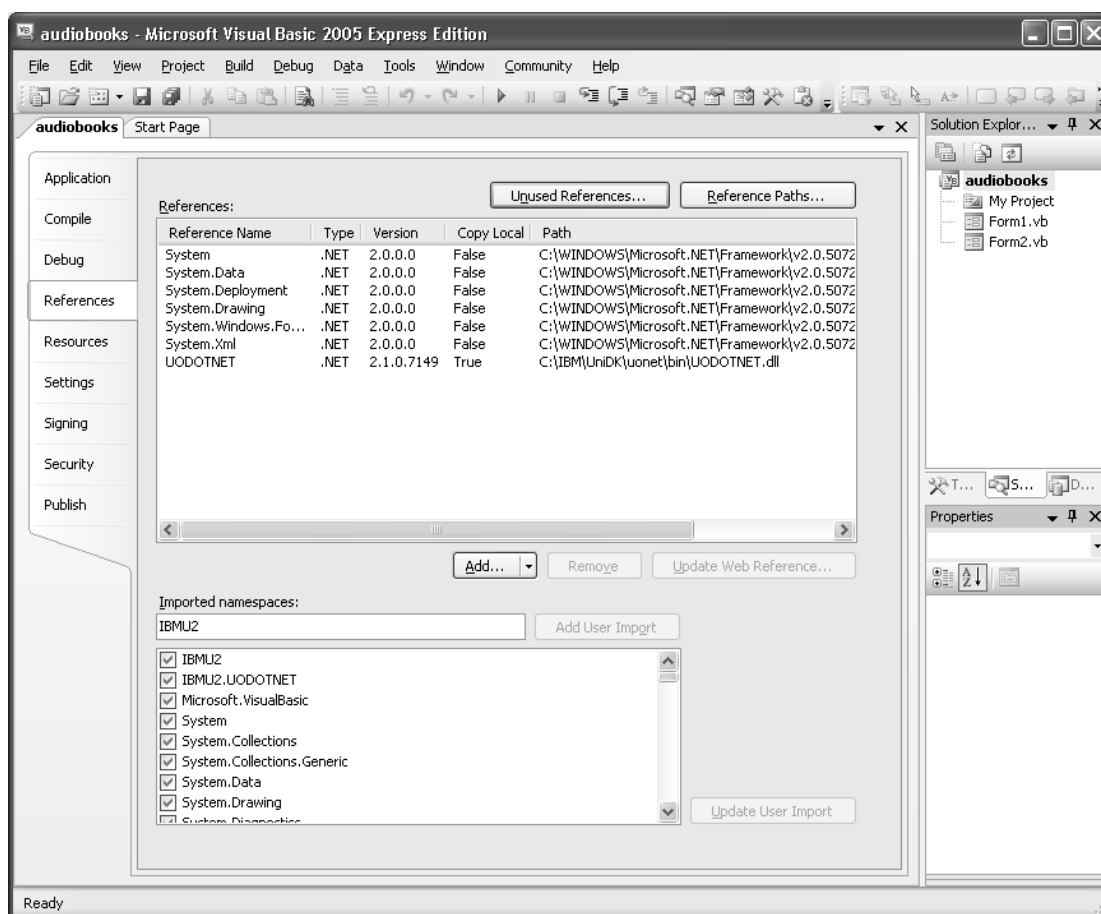
The .Net framework is huge and can be extended by many third party libraries. To organize all of this Microsoft structures the framework into a series of *namespaces*. A namespace is a long ("fully qualified") name that uniquely identifies a library or set of classes, making it easier to locate the classes you require and preventing vendors from overwriting each others classes. The namespace applied to UniObjects is: **U2.UODOTNET** or **IBMU2.UODOTNET** for earlier versions.

The namespace should be added to the front of every reference to anything held in the UO library - which is a mouthful and a lot of painful typing.

Fortunately you can save a load of this work by *importing* the namespace. The compiler checks any unqualified class names against a list of imported namespaces to find a match, allowing it to construct the full reference, which saves you having to type the reference yourself each time. You can import namespaces individually into each form and module, or for Visual Basic.NET you can import these globally as part of the project.

Click on the Project menu and select Properties. Then select the tab on the left marked References. You should see your UODOTNET reference already added to the project.

On the lower half of the references tab you will find a list of imported namespaces. Add the U2 and U2.UODOTNET (or earlier IBMU2) references as below:



Close the properties tab and we can move on to creating your first form.

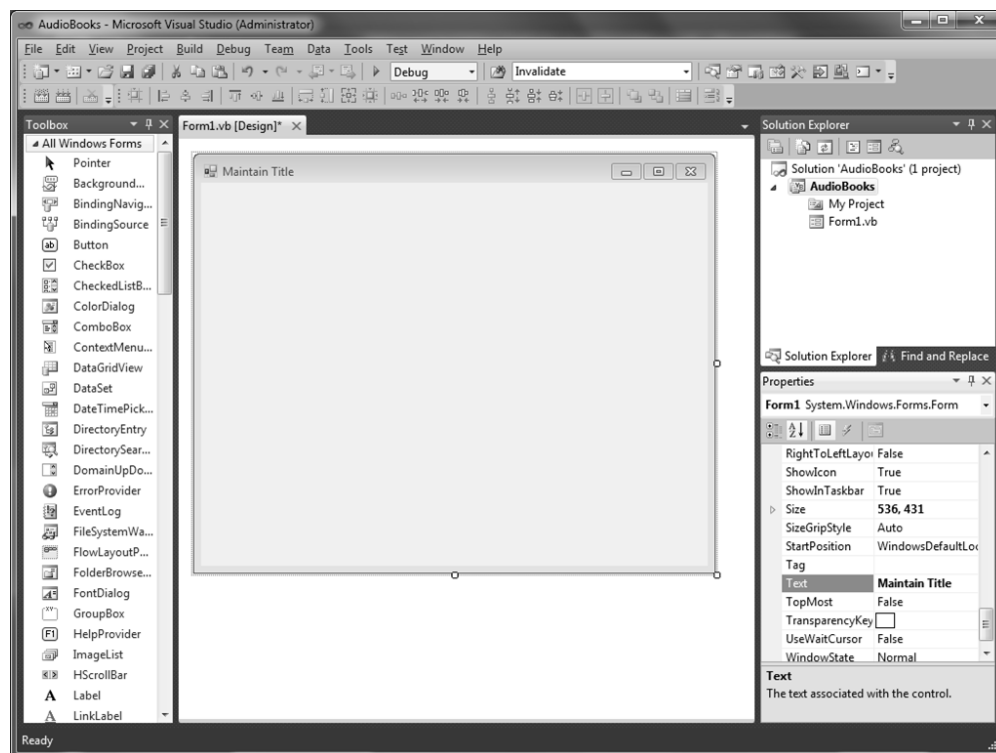
Creating the Book Titles Form

Your first task in this application will be to create a form to display title details drawn from the BOOK_TITLES file you listed in the last chapter. This will later be turned into a maintenance page where you can modify the title information or add new titles to the file.

Windows Forms applications, as the name suggests, are made up of a number of 'Forms'. The first one is automatically added when you create a new Windows Forms application. To build the application you modify the properties of the forms, add controls to interact with the user and code that will hold your business logic.

You can change some of the properties of the form whilst in design mode. You should find a panel called Properties, which by default opens beneath the Solution Explorer: but these can be moved around to suit so it may be docked to one of the other edges of the Visual Studio window. If the properties panel is not visible, select View->Properties Window to display it.

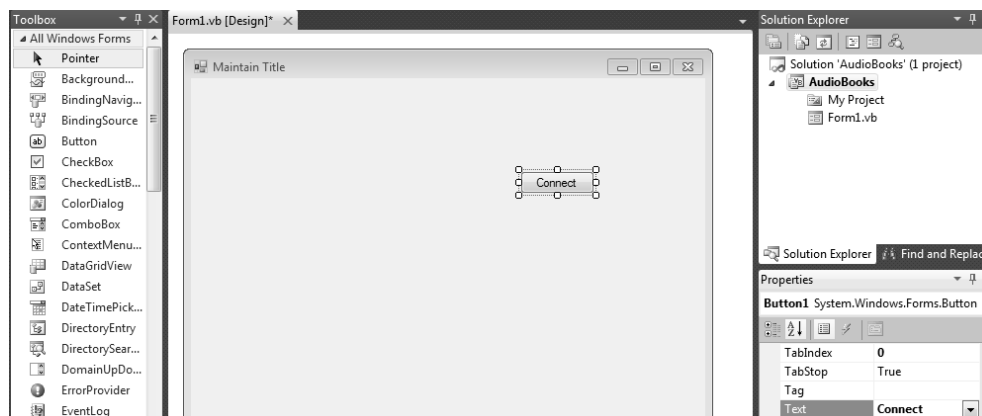
To prepare the form, you first need to change the Name property of the default Form1 to 'fBookTitle' and the Text property to read 'Maintain Title'. You can do this by finding these properties in Properties window and typing their new values as below:



Creating the Connection

Before you can work with the demonstration database in your project, your application will need to connect to the database just as you did through the terminal emulator in Chapter 2. UniObjects clients run separate UniVerse or UniData sessions and remain connected until you formally close their connection or until your application terminates.

Without worrying too much about aesthetics, drag a button from the Tool Box (on the right hand side in the illustration above) and drop it on the surface of the form in the designer window. Using the Properties window set the Text property of the button to *Connect*.



UniObjects.NET provides access to the database through a top level Object¹ called a **UniSession**. A UniSession encapsulates an open connection to a database account, similar to a user login through a terminal. Once a UniSession has been created and the connection established, it acts as a conduit for other UO.net Objects that need access to that account.

If you double click the button on your design form, Visual Studio will open the Code view where you can type the Visual Basic.NET code associated with this form. Here you will add a variable named Sess to reference the UniSession to your form code, and a second variable (Connected) to track the connection status. The content of the code window will then look similar to the code below:

```
Public Class fBookTitle
    Private Sess As UniSession
    Private Connected As Boolean = False

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        End Sub
End Class
```

¹ I will be using Object (title case) to reference a UniObjects entity, as opposed to object which is a Visual Basic/.NET object.

NOTE: This guide is not going to explain the Visual Basic .NET language. Hopefully most of it will make sense, and if not there are plenty of other books on the subject.

Next you will add some code to the button that will run when the button is clicked

Each Windows form and control responds to user actions through *Events*. An Event is an action that is fired when something happens – a user clicks on a button, enters text into a field, resizes or closes a form. Most controls support a variety of possible events. As a developer you create the logic for your application by attaching code to these events. This code is said to ‘handle’ the event.

The *Click* event is the default event for a button, and so when you double clicked the button on the design surface Visual Studio helpfully generated the skeleton of the event code for you:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
```

Any code you place between the Private Sub and the End Sub will run whenever the button is clicked at run time.

Add the following code to the Button1_Click event handler. You will need to substitute the user name and password that you used to connect with in the last section just for now but we will remove that in a later step - of course, you would never hard code a user name and password into a real application!²

```
Try
    Sess = UniObjects.OpenSession("localhost", "your_user_name", "your_password",
    "c:\wychbooks_pe")
Catch ex As Exception
    MsgBox("Cannot open UniObjects Session Error = " & ex.Message)
    Close()
End Try
Connected = True
MsgBox("Connection Established")
```

You can now run the application by pressing F5 or selecting Debug -> Start Debugging. If you entered the connection details correctly you should see the message **Connection Established** when you click the button.

You may see an error when you close the form including the text "Safe handle closed". You can safely ignore this error for now - we will deal with it later.

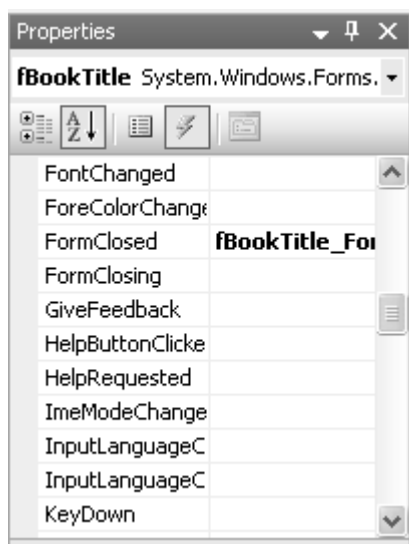
² You might want to create a temporary local demo user for your examples.

Closing the Connection

When your application stops, you will need to close the session to prevent the error above and to tidy up correctly. The `UniObjects.CloseSession()` method is used to close an open session.

The application will stop when the main form is closed. Fortunately there is an Event raised whenever a form is closed, so you can handle this event to disconnect nicely.

Switch to the tab marked "Form1.vb [Design]" and select the `fBookTitle` in the Properties window. Click on the button marked Events and find the `FormClose` event. This will run when the maintenance form is closed.



Double click in the `FormClosed` box to let Visual Studio create a new closed event for you. This will open the code window again with the event code displayed. Add the following lines to close the session safely when the application terminates:

```
Private Sub fBookTitle_FormClosed(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.FormClosedEventArgs) Handles MyBase.FormClosed
    If Connected Then
        UniObjects.CloseSession(Sess)
    End If
End Sub
```

So far your application is not much to look at! In the next section you will add the visual controls to your form to display a record from the `BOOK_TITLES` file.

Adding the Title Controls

Now you have a working connection, you can start to build the maintenance form.

Switch to the tab marked "Form1.vb [Design]" and drag a Label control onto the form surface from the Toolbox as you did with the connection button above.

Change the label **Text** to say **Title ID**.

Next drag a TextBox control next to the label, and change its **Name** to **txtID**.

To complete the form, add the following controls by dragging the appropriate tools from the Toolbox:

Label	Control Type	Control name
Short Title	TextBox	txtTitle
Author	TextBox	txtAuthorID
Media Type	ComboBox	cboType
Number in Stock	TextBox	txtStock
ISBN	TextBox	txtISBN
Price	TextBox	txtPrice
Department	ComboBox	cboDept
Genre	ComboBox	cboGenre

The complete form should look similar to the one below:

The screenshot shows the Visual Basic IDE with the 'Form1.vb [Design]' tab selected. The form, titled 'Maintain Title', is displayed. It features a 'Connect' button at the top left. Below this button are several input fields arranged in a form-like structure: 'Title Id' (a small text box), 'Short Title' (a wide text box), 'Author' (a text box), 'Media Type' (a dropdown menu), 'ISBN' (a text box), 'Department' (a dropdown menu), 'Genre' (a dropdown menu), 'Number in Stock' (a text box), and 'Price' (a text box). The form is designed in a simple, functional style typical of older database applications.

Reading and Writing Records

You should by now be familiar with the idea that UniVerse and UniData hold data as records in files. You may also recall that each record has a unique key, and that the database uses the key to identify the record.

UniVerse and UniData maintain information by reading, writing and deleting individual records. This makes it lightweight and efficient, even when handling huge data sets, as only one record is loaded at a single time.

It is possible to perform these operations using enquiry commands as you would for a SQL database, but this is almost never used in practise. Instead, UniVerse and UniData give developers the ability to directly read, write and delete records through code. This is much faster and more efficient than calling up an intermediate enquiry layer and is the key to the performance and scalability of a U2 application.

The pattern for maintaining information in a U2 database is typically:

- Read a record by providing the record key.
- Change the in-memory copy of the record.
- Write the updated record again providing the record key.

Almost all U2 applications work this way- though of course there may be many other steps in-between and surrounding these: selecting the records to maintain, presenting them in various different formats and requesting input through various mechanisms. We will look at some of these as you go through building the application.

The basic rule is always as follows:

Records are read as whole units, their content is modified and they are then written as whole units.

Opening a File

Access to individual data files is provided by the **UniFile** Object. This represents a single database file or dictionary, and offers methods for reading, writing, deleting and selecting the records in that file.

The file itself must belong to, or at least be visible from, the account to which you connect; and since that is determined by your open connection it makes sense that a UniFile Object must be created from an open UniSession:

MyFile = MySession.CreateUniFile("FileName")

The maintenance form will be updating the BOOK_TITLES file using a UniFile Object. The UniFile instance will be held as a private variable of the form and created when you first connect.

Add the following code to your connection function to create the UniFile instance referencing the BOOK_TITLES file:

```
Public Class fBookTitle
    Private Sess As UniSession
    Private BOOKTITLES As UniFile

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        Try
            Sess = UniObjects.OpenSession("your_server", "your_user", "your_pass",
"your_account")
        Catch ex As Exception
            MsgBox("Cannot open UniObjects Session Error = " & ex.Message)
            Close()
        End Try
        MsgBox("Connection Established")
        Button1.Visible = False

        Try
            BOOKTITLES = Sess.CreateUniFile("BOOK_TITLES")
        Catch ex As Exception
            MsgBox("Cannot open the BOOK_TITLES file")
            Close()
        End Try
    End Sub
```

Reading a Record

Once a UniFile Object has been created, you can use it to read and write records in the database file that it represents.

To read a record you must supply the record key as an argument to the UniFile.Read() method. If the record id is not found, an exception will be raised.

Add the following simple function to your form. This will read a record from the BOOKTITLES UniFile instance with a given key and will return a Boolean (true or false) value to report whether or not the read was successful:

```
Public Function ReadTitle(ByVal TitleId As String) As Boolean
    Try
        BOOKTITLES.Read(TitleId)
    Catch ex As Exception
        MsgBox(ex.Message)
        Return False
    End Try
    Return True
End Function
```

You will call this function to read the record: the next step is to find out how to display the individual fields from that record.

Dynamic Arrays

Following a successful Read operation, the UniFile places the record content into a property called Record. But the Record may contain many fields and values, so how can you access these to display the individual fields on your form?

The Record property is a UniDynArray Object, an implementation of a U2 programming feature called a Dynamic Array. UniVerse and UniData use Dynamic Arrays throughout their native programming model to represent record contents.

A dynamic array, as the name suggests, is an array of values. Where the array holds a record, each element in the array represents a field of that record.

You can access the individual fields by using the Extract method of a UniDynArray object. This does not remove the element, but returns a copy for you to work with:

```
txtTitle.Text = BOOKTITLES.Record.Extract(1).ToString
```

Now, just as UniVerse and UniData fields can contain multiple values and sub-values, so the Extract method also allows you to extract those values and sub-values from a field by specifying their ordinal positions as a sequence:

MyVariable = MyArray.Extract(FieldNumber,ValueNumber)

NOTE: All elements are numbered from 1.

You can have everything you need to populate the form controls in your ReadTitle function. Notice the use of the UniSession **OCONV** method for the price: this simply applies a format mask to ensure that the price is presented with two decimal places.

You should find that the Visual Studio IntelliSense offers you assistance as you type these, so it is not as onerous as it seems:

```
Public Function ReadTitle(ByVal TitleId As String) As Boolean
    Try
        BOOKTITLES.Read(TitleId)
    Catch ex As Exception
        MsgBox(ex.Message)
        Return False
    End Try

    txtTitle.Text = BOOKTITLES.Record.Extract(1).ToString
    txtAuthorId.Text = BOOKTITLES.Record.Extract(2).ToString
    cboType.Text = BOOKTITLES.Record.Extract(6).ToString
    txtISBN.Text = BOOKTITLES.Record.Extract(8).ToString
    txtStock.Text = BOOKTITLES.Record.Extract(5).ToString
    txtPrice.Text = Sess.Oconv(BOOKTITLES.Record.Extract(9).ToString,
"MD2")

    cboDept.Text = BOOKTITLES.Record.Extract(11).ToString
    cboGenre.Text = BOOKTITLES.Record.Extract(12).ToString

    Return True
End Function
```

Before we finish this exercise, here is a quick test - how do you know which fields are held in which ordinal positions?

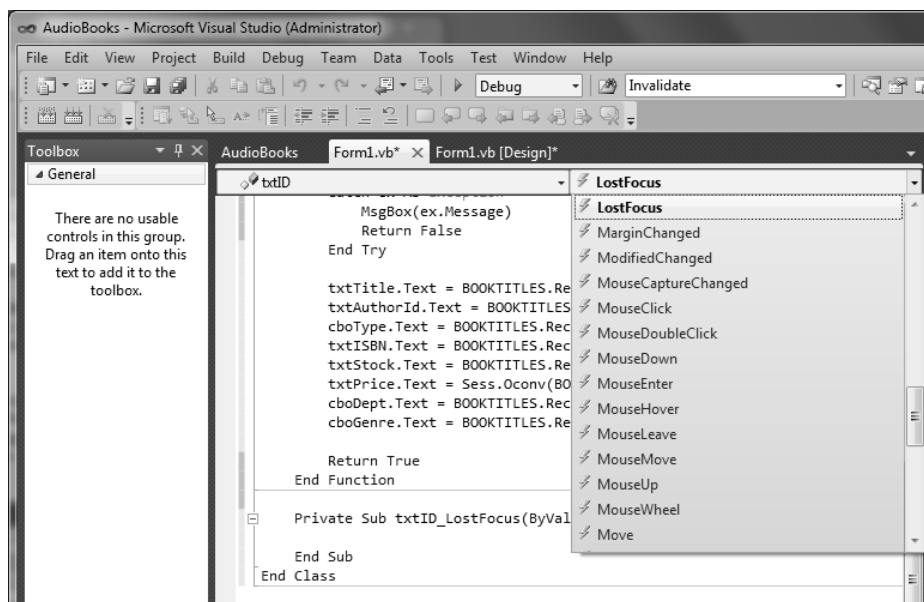
Answer - by listing the file dictionary as you did in the previous chapter. Of course, for live applications these should be represented by constants or by enumerations rather than numbers, and those would probably be automatically built from the dictionary to ensure correctness.

Triggering a Read Event

Now for the exciting bit! We have had to cover a lot of background to get to this point, but having completed the `ReadTitle()` function you can now hook up the final event to call this to read and display a `BOOK_TITLE` record when a number is typed into the Title ID textbox at the top of your form.

We want to read the record after you have entered the record id into the `txtID` text box – the first box on the form. For this you can use the **LostFocus** event of that text box, which is triggered when the focus moves away from the text box and onto the next control.

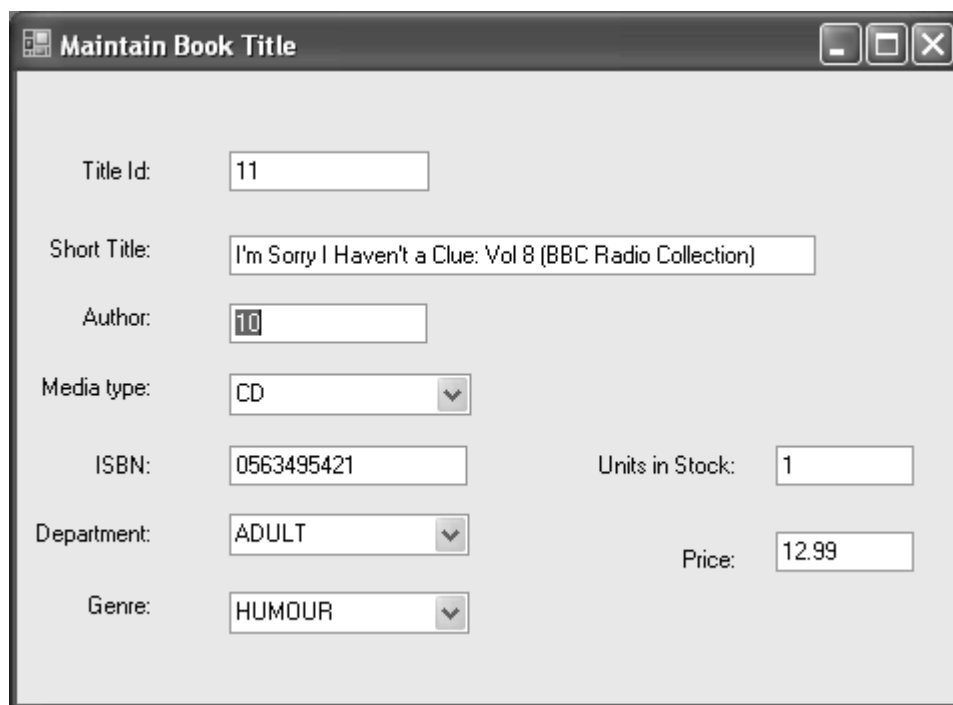
The code page for the form contains two dropdown boxes above the code area. From here you can select the objects on the form and the events you want to handle. Select the **txtID** control in the left hand box, and then pull down the list of events for that control from the right hand box. Select the **LostFocus** event to build the handler as shown below:



The `LostFocus` event will call the `ReadTitle` function you just created. If the function fails, you will place the cursor back into the ID field so that the operator can try again:

```
Private Sub txtID_LostFocus(ByVal sender As Object, ByVal e As System.EventArgs)
Handles txtID.LostFocus
    If ReadTitle(txtID.Text) = False Then
        txtID.Text = ""
        txtID.Focus()
    End If
End Sub
```

Now run the form and enter a Title id (any number between 1 and 200). The details for that title will be displayed.



The screenshot shows a window titled "Maintain Book Title". Inside the window, there are several data entry fields:

- Title Id: 11
- Short Title: I'm Sorry I Haven't a Clue: Vol 8 (BBC Radio Collection)
- Author: 10
- Media type: CD (dropdown menu)
- ISBN: 0563495421
- Units in Stock: 1
- Department: ADULT (dropdown menu)
- Price: 12.99
- Genre: HUMOUR (dropdown menu)

Well done – you have your first working form!

There is still some tidying up required. You should add a ClearFields() subroutine that will clear the content of the form before the record is read.

```
Private Sub ClearFields()  
    txtTitle.Text = ""  
    txtAuthorId.Text = ""  
    cboType.Text = ""  
    txtISBN.Text = ""  
    txtStock.Text = ""  
    txtPrice.Text = ""  
    cboDept.Text = ""  
    cboGenre.Text = ""  
End Sub
```

Locking Records

You have seen how the UniFile Object can be used to read a record. The UniFile Object can also update a record by calling its Write method. But to do so safely requires record locking.

U2 Databases use pessimistic locking by default. Because records are read and written individually, UniVerse or UniData will lock an individual record on demand to prevent other users making updates to that record that might potentially cause problems with changes being overwritten.

The UniVerse BASIC Manual has a whole chapter devoted to locking. You should read that before attempting to write any commercial quality code – even if you are using UniData.

The UniObjects family introduces a concept of a Locking Strategy. This can be set globally at the session level, or for individual file objects, and controls the setting and release of record locks. The UniFile Object provides this in the shape of two properties: a **UniFileLockStrategy** that allows you to state when a record should be locked, and a separate **UniFileReleaseStrategy** that specifies when the lock should be released.

To ensure that the screen locks each record for update, set the locking strategy to lock on read and to unlock on write as part of the ReadTitle function:

```
BOOKTITLES.UniFileLockStrategy = 1  
BOOKTITLES.UniFileReleaseStrategy = 1
```

Add the following code to explicitly unlock the previous record for safety before reading the next record if you have not written away any changes:

```
If BOOKTITLES.RecordID <> "" Then  
    BOOKTITLES.UnlockRecord()  
End If
```

After setting the locking strategy you can add an update facility to your form.

Writing Changes

Drag a new Command button onto the form and set its **Text** property to **Update**. Create a **Click** event that will replace the fields in the Record dynamic array with the contents of the controls on your form. This will be the mirror image of the Read operation.

At the end, this will call the Write method to write the newly modified copy of the record back onto the BOOK_TITLES file:

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click

    BOOKTITLES.Record.Replace(1, txtTitle.Text)
    BOOKTITLES.Record.Replace(2, txtAuthorId.Text)
    BOOKTITLES.Record.Replace(6, cboType.Text)
    BOOKTITLES.Record.Replace(8, txtISBN.Text)
    BOOKTITLES.Record.Replace(9, Sess.Iconv(txtPrice.Text, "MD2"))
    BOOKTITLES.Record.Replace(11, cboDept.Text)
    BOOKTITLES.Record.Replace(12, cboGenre.Text)

    Try
        BOOKTITLES.Write()
    Catch ex As Exception
        MsgBox("Cannot write data")
    Exit Sub
End Try
MsgBox("Record updated")
ClearFields()
txtID.Focus()
End Sub
```

Adding New Records

The U2 model does not artificially differentiate between an insert and an update. If you wish to add a new record to the file, you can simply write a record with a key that does not previously exist.

Running Commands and Selections

You have now completed a simple maintenance form that allows you to read and modify details held in a UniVerse or UniData file. This still suffers from one drawback: you need to know the key to the record. This is not always possible.

In the earlier chapter you discovered how to run a LIST command to display the records in a file. The LIST command can also be used to select specific records based on various criteria, and can be used to generate XML.

You can leverage this knowledge to add some browsing capabilities to your form.

The UniCommand Object

The UniCommand Object, as the name suggests, allows you to run a command, just as if you were entering it into the command shell. The UniCommand Object, like the UniFile Object, is bound to a specific connection, and so must be created directly from the UniSession object:

```
Dim Cmd As UniCommand

Try
    Cmd = Sess.CreateUniCommand()
Catch ex As Exception
    MsgBox("Cannot create UniCommand object")
Exit Sub
End Try
```

You can run practically any U2 command through the UniCommand Object by completing the following steps:

- Assign the text of the command to the Command property.
- Call the Execute method to run that command.
- View the results through the Response property.

Add the following function to add the result of the WHO command (stripped of any blank lines from the text) into the Maintenance Form title bar.

```
Private Sub ShowUser()  
    Dim Cmd As UniCommand  
  
    Try  
        Cmd = Sess.CreateUniCommand()  
    Catch ex As Exception  
        MsgBox("Cannot create UniCommand object")  
        Exit Sub  
    End Try  
  
    Cmd.Command = "WHO"  
    Cmd.Execute()  
    Text = Text & " " & Cmd.Response.Replace(vbCrLf, "  
")  
  
End Sub
```

The UniXML Object

Getting text in this way from a command is useful, but the format of that text is not always well designed for parsing. If you wanted to fetch data from a command that can be used in the client, it may be better to format it into JSON or XML.

In the previous chapter you learned how to use the LIST command to produce XML. You could pass this to a UniCommand Object and capture the results, as in the example above. But there is an easier way.

The UniXML Object provides an additional wrapper around an XML listing that makes it even easier to add to your application. In particular, the UniXML Object can return XML results directly in the form of a .NET DataSet.

To do this, you use two methods of the UniXML class:

```
UniXML.GenerateXML  
UniXML.GetDataSet
```

For example

```
Cmd.GenerateXML("SORT BOOK_TITLES SHORT_TITLE AUTHOR_NAME")  
DS = Cmd.GetDataSet
```

If you are not familiar with DataSets, don't worry: we will be calling on this to create a browse list in the next example.

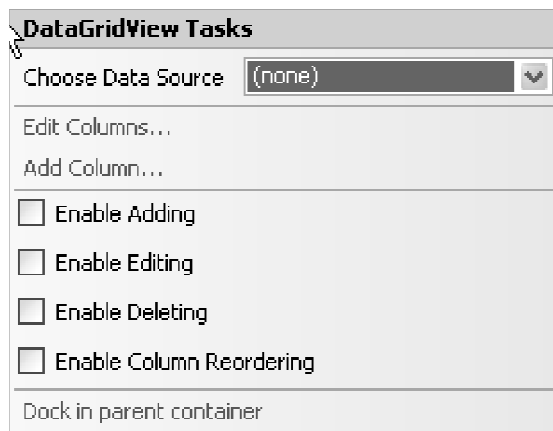
Adding a Browse List

Your maintenance form relies on your knowing the record key for the title to display or to maintain. This is a clear weakness in the design, so you will now add a browse list to the application. This will list all of the existing titles, allowing you to select a title and populate the maintenance form with the title details.

The browse list should present a list of book titles as a grid, and there is a ready-made grid component in the shape of the **DataGridView**, available from the Data section of the Toolbox.

Enlarge the maintenance form to give yourself some room and draw a DataGridView control onto the left hand side. You will need to move the other controls across.

As you do so it will pop up a small tasks window next to the grid: this provides a short cut to the most used features.



The DataGridView component is designed for data binding. You will be using this facility to present the browse list of titles.

The data source will be assigned at run time, so select **None** for the data source and uncheck each of the **Enable** options as above.

The browse list will retrieve a list of title records, by creating a UniXML Object and running the following Retrieve/UniQuery command:

```
SORT BOOK_TITLES SHORT_TITLE AUTHOR_NAME
```

DataGridView components bound to a DataSet will automatically display the content of that DataSet; and the DataSet can in turn be populated with XML data using the UniXML Object.

So putting all of this together provides an easy way to bind the results of a UniVerse or UniData command:

```
Public Function GetTitles() As Boolean
    Dim Cmd As UniXML
    Dim DS As New DataSet

    Try
        Cmd = Sess.CreateUniXML
        Cmd.GenerateXML("SORT BOOK_TITLES SHORT_TITLE AUTHOR_NAME")
        DS = Cmd.GetDataSet
        DataGridView1.DataSource = DS
        DataGridView1.DataMember = "BOOK_TITLES"
        DataGridView1.Refresh()
    Catch ex As Exception
        MsgBox(ex.Message)
        Return False
    End Try

    Return True
End Function
```

This list should display as soon as the connection to the server has been made.

Selecting the Title from the Grid

Finally you can complete the wiring to display the maintenance details for a title when the grid view line is double clicked. Use the Events view of the Properties panel for the DataGridView to populate the CellDoubleClick event as below:

```
Private Sub DataGridView1_CellDoubleClick(ByVal sender As System.Object,
    ByVal e As System.Windows.Forms.DataGridViewCellEventArgs) Handles
    DataGridView1.CellDoubleClick
        Dim TitleId As String

        TitleId = DataGridView1.Rows(e.RowIndex).Cells(0).Value.ToString
        txtID.Text = TitleId
        ReadTitle(TitleId)
    End Sub
```

Run your project and select a title from the list of titles as shown below.

_ID	SHORT_TITLE	AUTHOR_NAME
1	Just William: No. 6 (BBC Radio Collection)	Richmal Crompton
2	Just So Stories	Rudyard Kipling
3	The Duchess of Malli (abridged version)	John Webster
4	Great Expectations	Charles Dickens
5	The Importance of Being Earnest	Oscar Wilde
6	The 7 Habits of Highly Effective People	Stephen R. Covey
7	The American Boy	Andrew Taylor
8	James and the Giant Peach (Puffin Audio...)	Roald Dahl
9	Hogfather	Terry Pratchett
10	Hancock a Comedy Genius (BBC Radio C...	Cast
11	I'm Sorry I Haven't a Clue: Vol 8 (BBC Rad...	Cast
12	Friends, Lovers, Chocolate	Alexander McCall
13	The Legend of Spud Murphy	Eoin Colfer
14	Farmer Giles of Ham and Other Stories	J. R. R. Tolkien
15	The Lord of the Rings: Complete & Unabrid...	J. R. R. Tolkien
16	The Navy Lark: Taking Some Liberties v. ...	Cast
17	The Worst Witch: Complete and Unabridg...	Jill Murphy
18	Friends, Lovers, Chocolate	Alexander McCall
19	Harry Potter and the Chamber of Secrets [...]	J.K. Rowling
20	Grass Roots Management	Guy Browning

Title Id: 13

Short Title: The Legend of Spud Murphy

Author: 12

Media type: CD

ISBN: 0141805293

Units in Stock: 5

Department: JUNIOR

Price: 5.99

Genre: FANTASY

Update

That's the first version of your maintenance form completed.

In the next chapter you will learn how to improve your application by creating and calling server based routines.

Chapter 4: Server Coding

Content:

- Server Coding
- Server Programming
- Reading Data
- Using Dynamic Arrays
- Using Constants
- Creating a Subroutine
- Calling Subroutines

Objectives

This chapter introduces the UniBasic server language. You will create your first programs and subroutines and learn how to read and format data directly on the server.

UniVerse Server Coding

Reading and writing data directly using the UniFile Object is fast and easy, but is not the recommended way to build applications other than in the most trivial cases. Direct reads and writes scatter data access throughout your code, especially if you start to develop more than one interface to the same U2 files – for example, a Windows Forms client, a set of web pages, XML gateways and the like. As with any development, scattering your data access leads to difficulties when the time comes to make changes.

There are other problems involved with client/server reading and writing. Performing network operations imposes overheads that most applications would rather avoid. And it bypasses the use of the U2 model as an embedded business platform.

The recommended way is to call server code to perform your business operations whilst remaining within the database application engine, keeping the logic close to the data whilst preserving your freedom to design your interfaces in your technologies of choice.

Server Side Programming

Both U2 databases are equipped with a substantial programming language that, like many modern languages, is modelled after a variant of Basic. Just as Visual Basic .NET has added features appropriate to a graphical, object oriented environment, so UniBasic has added features appropriate to a business application and in particular to working with the file model.

UniBasic code is written as programs and subroutines (and external functions, but we will ignore these for this guide). Programs are standalone routines used for performing fixed tasks within the database environment: generally running utilities or reports. Subroutines are external blocks of code that can be called from both inside and outside the UniVerse environment, similarly to the stored procedures in some other traditional databases. Subroutines are the building blocks of applications and the way to expose your business logic to the outside world.

Once again this guide will only introduce the fundamentals of UniBasic programming – that is a large subject in its own right. But by the end of this chapter you will have created subroutines to read and write title information in a more complete manner.

Creating and Running Programs

UniBasic programs are written inside the database environment, and reside in database files. Each program or subroutine is contained in separate record in a file. However, and this will serve to make your life considerably easier, source files are always directory files.

UniVerse:

To create any new file in a UniVerse account, you need to connect into UniVerse using the Command Shell and run the CREATE.FILE command:

```
CREATE.FILE filename [parameters]
```

The parameters define the type of file to be created and, for certain types of file, the initial space to be reserved on disk to optimise storage.

To create a new directory file to hold your UniBasic source code called PROGS, you need to ask UniVerse to create a type 19 file by issuing the following command:

```
CREATE.FILE PROGS 1,1 1,1,19
```

Don't worry about the syntax for now. You can read more about the CREATE.FILE command in the UniVerse Administration guide.

UniData:

To create any new file in a UniData account, you need to connect into UniData using the Command Shell and run the CREATE.FILE command:

```
CREATE.FILE [DIR] filename [parameters]
```

To create a new directory file to hold your UniBasic source code, the equivalent UniData command is:

```
CREATE.FILE DIR PROGS
```

If you open up Windows Explorer and look inside the c:\wychbooks\wychbooks_pe folder, you will see a new directory called PROGS. This is your PROGS file – and the database will see each file you place into there from Windows as a separate record in that file.

Directory files should only be used for programs and for sharing information with other processes. They should not be used for holding data.

Your First Server Program

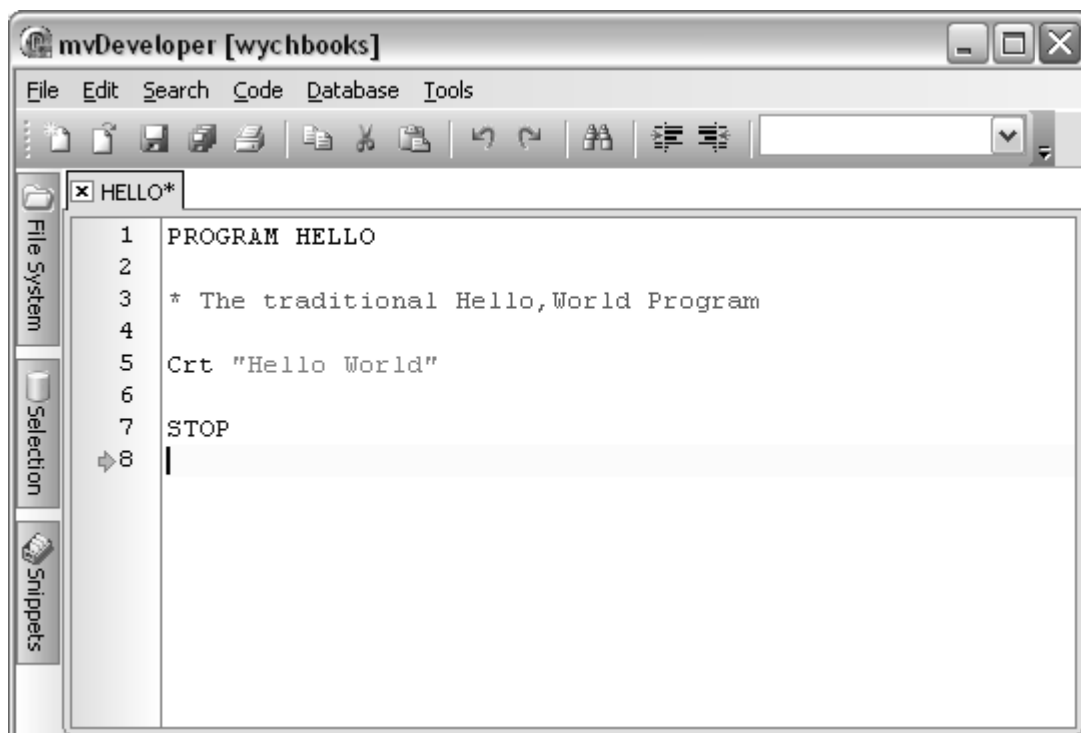
It is traditional to begin any introduction to programming with the ubiquitous "Hello, World" program and this guide will be no exception.

To create a new program you need to edit the program text into a new program record in your source file using an appropriate text editor. There are various editors available for U2 developers, and out of all of these the system editors provided inside UniVerse and UniData are by far the most horrible and generally (and rightfully) derided.

If you have chosen to download the mvDeveloper in the first chapter, this provides a simple-to-use Windows based editor with support for syntax highlighting and the usual programmer features. If not, you can simply open Notepad and save your text document directly into your program directory from Windows.

Open Notepad, or connect mvDeveloper to your server and select the New button.

Type the following text:



Save the resulting program into the PROGS file under the name HELLO. If you use Notepad you will need to remove the .txt extension.

Compiling your Program

Before a Server program can be used, it must first be compiled. This is done from within the Command Shell through the BASIC command:

```
BASIC filename programname (options)
```

UniVerse:

Compile your new HELLO program using the command:

```
BASIC PROGS HELLO
```

This will generate a new directory file named PROGS.O that will hold the object code for each program in the PROGS file.

```
>BASIC PROGS HELLO
Creating file "PROGS.O" as Type 19.
Creating file "D_PROGS.O" as Type 3, Modulo 1, Separation 2.
Added "@ID", the default record for Retrieve, to "D_PROGS.O".
```

```
Compiling: Source = 'PROGS/HELLO', Object = 'PROGS.O/HELLO'
*
```

Compilation Complete.

UniData:

Compile you new HELLO program using the command:

```
BASIC PROGS HELLO -i
```

UniData basic requires the `-i` argument to tell the compiler that the source code is case insensitive. The compiler creates the object code as an item in the same directory, named `_HELLO`.

The UniData compiler accepts two types of syntax: the 'Pick' and the 'UniData' styles. For this guide we are sticking to the (nicer) Pick style, so you can force this by issuing the `BASICTYPE` command as below:

```
:BASICTYPE "P"
:BASIC PROGS HELLO -i
```

```
Compiling Unibasic: PROGS\HELLO in mode 'p'.
Compilation finished
```

Running your Program

You can test the program by using the RUN command thus:

```
RUN PROGS HELLO  
Hello World
```

Finally you can publish this program so that it can be used as a command in its own right or can be called from external sources such as a UniObjects client. Before any program or subroutine can be used externally it should be published using the CATALOG command:

```
CATALOG PROGS HELLO  
HELLO  
Hello World
```

Examining the Syntax

Now you are a *bona fide* server programmer, let's take a look at what you just created.

Line 1 contains the item header. This is the word PROGRAM or SUBROUTINE followed by the name of the item. If the subroutine has a parameter list, this follows the subroutine name.

Line 2 contains whitespace. You can use blanks to separate out your code into legible blocks.

Line 3 contains a comment. Comments begin with an asterisk or an exclamation mark and continue until the end of the line.

Line 5 contains a statement. A UniBasic program consists of a series of statements that are normally executed in order. In this case, the statement consists of a command (Crt) and a literal "Hello World".

Line 7 terminates the program with another statement, STOP.

NOTE

In UniVerse, statements and key words are not case sensitive. In UniData programs they are, unless you use the `-i` option.

Reading Data

If this were a proper UniBASIC publication, we would now spend the next 50 to 100 pages examining the various syntax structures, operators, variables, compiler instructions and so forth that make up the UniBASIC language. You can learn about these from the self-paced guides on the Author's website. This guide just wants to get you up and running, so we will forego most of this, and move on directly to reading data.

You have already seen the model for working with U2 data, when you created your first client application using UniObjects.NET in the previous chapter. Using UniObjects, a file is accessed through a UniFile Object, records are read from the file by supplying the key, the data is modified and then written back to the file using the same record key. This is the standard data access model for UniVerse and for UniData, and working with data in UniBASIC is no different in concept, though of course the syntax is necessarily different.

Just as UniObjects provides the UniFile Object as a means of accessing a UniVerse or UniData file, so UniBASIC requires that files are accessed using a special 'file' variable. This is created by using an OPEN statement thus:

```
Open FileName To FileVariable {Then|Else}
```

For example:

```
Open "BOOK_TITLES" To BOOKTITLES Else
  Crt "Cannot open the BOOK_TITLES File"
  STOP
End
```

This introduces a number of pieces of syntax for you to consider.

The OPEN statement associates a file name ("BOOK_TITLES") with a variable (BOOKTITLES). All variables in UniBASIC are defined by context: there is no need to declare variables before they are used. Unlike commands and keywords, variables are case sensitive.

The second thing to notice is the ELSE clause. UniBASIC does not raise exceptions when there is an error. Instead, certain commands take an explicit Else branch that can be used to handle errors gracefully. The Else clause will run to a block of statements on following lines. The end of the Else block is signalled with an END statement.

A very similar syntax is used when reading a record. This uses the READ statement as below:

```
READ Record From FileVariable, Key {Then|Else}
```

For example

```
TitleId = 3
Read TitleRec From BOOKTITLES, TitleId Else
  Crt "Cannot read this title"
  STOP
End
```

The Read statement is supplied with the record id and returns the record as a dynamic array, just as the UniFile Object did in your Visual Basic.Net application. Notice that again, there is no need to declare the TitleId or TitleRec variables before they are used.

UniBASIC does not require that you declare data types. If you think this is a bad thing, consider that neither do the world's most popular languages – PHP and javascript! UniBASIC uses full data typing internally and coerces variables into the correct data type at runtime.

Putting this new found knowledge to the test, you will now create a program called ShowRecord.

This will open the BOOK_TITLES file, and read the title record with the record id 3. You can show the content of the record using the statement:

Crt TitleRec

Your program will look similar to the one below:

```
PROGRAM ShowRecord

Open "BOOK_TITLES" To BOOKTITLES Else
  Crt "Cannot open the BOOK_TITLES File"
  STOP
End

TitleId = 3
Read TitleRec From BOOKTITLES, TitleId Else
  Crt "Cannot read this title"
  STOP
End

Crt TitleRec

STOP
```

You can compile and run this example, but beware that the results may not look right! We will deal with that by introducing the concept of Dynamic Arrays.

Using Dynamic Arrays

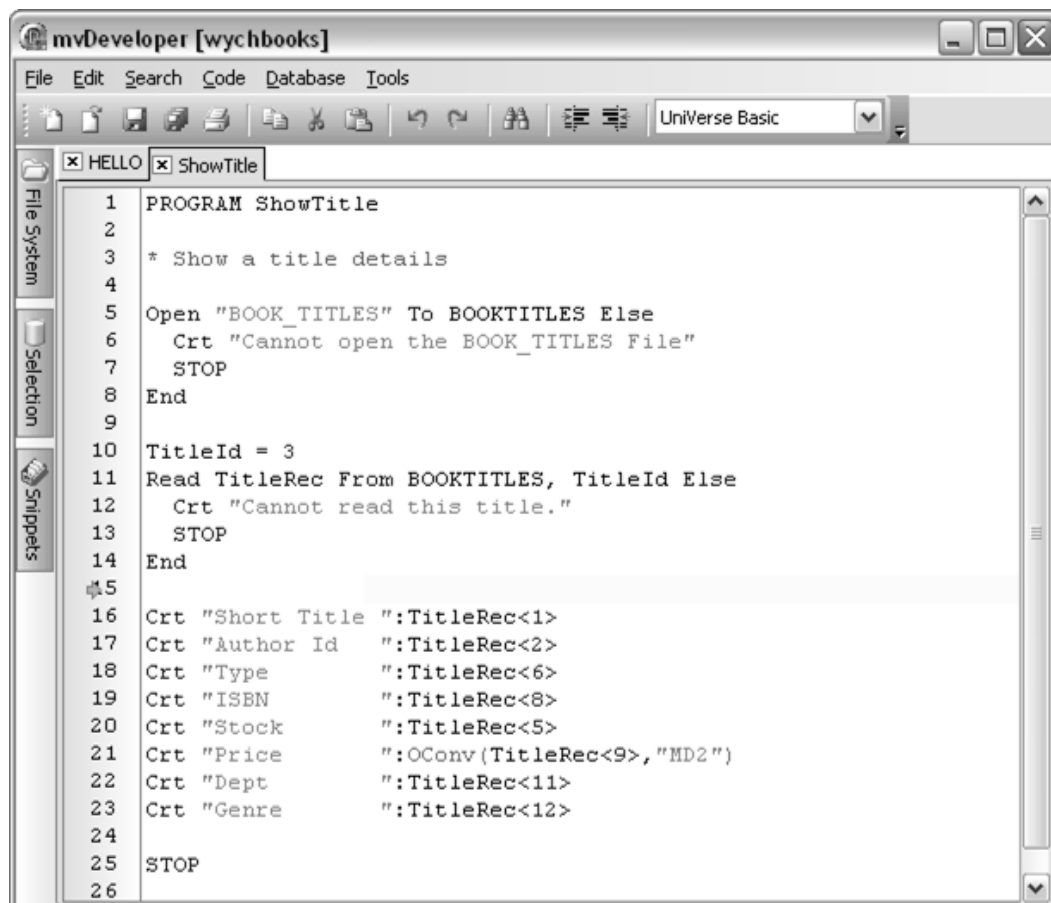
In your UniObjects.NET application, you used the UniDynArray Object to hold a record and to expose methods for extracting and replacing array elements. In UniBASIC all variables can be treated as dynamic arrays, and the syntax for extracting elements is very much simpler:

```
MyVariable = MyArray<FieldNumber>
MyVariable = MyArray<FieldNumber, ValueNumber>
```

And when replacing elements:

```
MyArray<FieldNumber> = MyValue
MyArray<FieldNumber, ValueNumber> = MyValue
```

You can now create an improved program that will read and display the details from a BOOK_TITLES record, field by field. Remember that you find the meanings of each field by listing the dictionary. An example is shown below:



Notice that although there are syntax differences, this is almost identical to the ReadTitle() function in your VB.Net application. Compile and run it as below:

```

>BASIC PROGS ShowTitle
Compiling: Source = 'PROGS/ShowTitle', Object = 'PROGS.O/ShowTitle'
**

Compilation Complete.
>RUN PROGS ShowTitle
Short Title The Duchess of Malfi (abridged version)
Author Id   3
Type        TAPE
ISBN        0001050478
Stock       8
Price       10.99
Dept        ADULT
Genre       DRAMA

```


Using Constants

Referring to fields by number is all very well, but it makes for code that is difficult to read and awkward to search – and makes it easy to type the wrong field numbers. Good practice dictates that you define the field numbers using constants.

Constants are defined by the EQUATE (or EQU) statement:

EQU constant TO expression

For example:

```
EQU First To 1
EQU Second To 2
EQU BIG To 99999
```

You should get into the habit straight away of using EQUATED constants, so change your program to name the various fields in the BookTitle array, following the conventions of the snippet below:

```
EQU TITLE.TITLE To 1
EQU TITLE.AUTHOR.ID To 2
EQU TITLE.TYPE To 6

Crt "Short Title :" : TitleRec<TITLE.TITLE>
Crt "Author Id   :" : TitleRec<TITLE.AUTHOR.ID>
Crt "Type       :" : TitleRec<TITLE.TYPE>
```

Constants are handled by compiler token substitution, so there is no overhead in using them.

Creating a UniBASIC Subroutine

You have now seen how to read a record in a server program. So far, the code has been functionally identical to the code you wrote in the previous chapter. So why bother to do this on the server?

If that data changes, or if we want to supply more data than can simply be found in a single file, or when you want to validate the information that is written back, the balance begins to move towards the server processing. The problem with doing validation or processing on the client is that you have to repeat the same validation for every separate client you write - and if the rules or the structure change, you need to roll out those same changes everywhere. At the very least, you have to re-test them for each client. So to prevent this you should **always** place your logic inside UniBASIC subroutines.

A UniBASIC subroutine, just like a Sub procedure in Visual Basic or a void function in C#, is a block of code designed to perform a specific operation. Usually this operation is performed against information passed into and out of the subroutine. In Visual Basic.NET or C#, these are compiled into the main assembly. In UniVerse and UniData, these are held as separate code libraries and dynamically loaded at run time. This means that you can add new subroutines at any time without recompiling your entire application. Moreover, you can call them from practically any interfacing environment including UniObjects.NET.

A UniBASIC subroutine begins with the keyword **SUBROUTINE** followed by the (optional) subroutine name and a list of arguments that will be accepted from the calling routine:

```
SUBROUTINE mySubroutine( arg1, arg2 )
```

Internally, the subroutine is the same as the program you typed earlier, except that it finishes with a **RETURN** statement not a **STOP**. When it returns, any changes to the arguments are also returned to the calling program.

```
SUBROUTINE Greet( AName, Language, Greeting )

    Greeting = "HELLO ":AName

Begin Case
    Case Language = "FRENCH"
        Greeting = "Bonjour, ": AName
    Case Language = "ITALIAN"
        Greeting = "Ciao ": AName
    Case Language = "GERMAN"
        Greeting = "Guten Tag ":AName
End Case

RETURN
```

So where would you use such a subroutine? Here is a very simple example.

Your client application currently reads a single BOOK_TITLE record. This displays details of the title, but it does not include the author name. As you learned in Chapter 2, the author name is held on the BOOK_AUTHORS file using the AUTHOR_ID as a foreign key.

If you wanted to show the full title details in your .NET application, you would first need to be aware of this fact, and you would then need to read the author details from the BOOK_AUTHORS file through a second UniFile Object. This is not difficult, but it means creating a new object and performing another network operation. If you did the same in another environment, you would have to repeat the process. It also means that you – the client developer – need to know the structure of the database.

A better solution is to create a subroutine that will return all the title details in a single array. This still has to open the BOOK_AUTHORS file and to read the author details, but this happens much more quickly at the server and once it has been completed, the same routine can then be called from any number of places without the client having to know where the data has come from.

Here is a subroutine to offer an improved view of the title details. Note that I have added EQUATEs to make the code more legible.

You will need to copy and paste this to create a new subroutine in your PROGS file.

```

SUBROUTINE GetTitle(TitleId, Details, Error )

* Get Title details

EQU DETAIL.DESC          TO 1
EQU DETAIL.AUTHOR        TO 2
EQU DETAIL.AUTHORNAME    To 3
EQU DETAIL.TYPE          To 4
EQU DETAIL.ISBN          To 5
EQU DETAIL.PRICE         To 6
EQU DETAIL.STOCK         To 7
EQU DETAIL.DEPT          To 8
EQU DETAIL.GENRE         To 9

EQU TITLE.TITLE          To 1
EQU TITLE.AUTHOR.ID      To 2
EQU TITLE.STOCK          To 5
EQU TITLE.TYPE           To 6
EQU TITLE.ISBN           To 8
EQU TITLE.PRICE          To 9
EQU TITLE.DEPT           To 11
EQU TITLE.GENRE          To 12

Open "BOOK_TITLES" To BOOKTITLES Else
    Error = "Cannot open the BOOK_TITLES File"
    RETURN
End
Open "BOOK_AUTHORS" To BOOKAUTHORS Else
    Error = "Cannot open the BOOK_AUTHORS File"
    RETURN
End

Read TitleRec From BOOKTITLES, TitleId Else
    Error = "Cannot read title."
    RETURN
End
Details = ""

Details<DETAIL.DESC> = TitleRec<TITLE.TITLE>
Details<DETAIL.AUTHOR> = TitleRec<TITLE.AUTHOR.ID>
Read AuthorRec From BOOKAUTHORS, TitleRec<TITLE.AUTHOR.ID> Else
    AuthorRec = ""
End
Details<DETAIL.AUTHORNAME> = AuthorRec<1>

Details<DETAIL.TYPE> = TitleRec<TITLE.TYPE>
Details<DETAIL.ISBN> = TitleRec<TITLE.ISBN>
Details<DETAIL.STOCK> = TitleRec<TITLE.STOCK>
Details<DETAIL.PRICE> = OConv(TitleRec<TITLE.PRICE>,"MD2")
Details<DETAIL.DEPT> = TitleRec<TITLE.DEPT>
Details<DETAIL.GENRE> = TitleRec<TITLE.GENRE>

RETURN

```

Calling a Subroutine from the Server

You can call a subroutine from inside a server program or from another subroutine, using the Call statement thus:

```
Call SubroutineName( arguments )
```

For example

```
Call GetTitle( 3, Details, Error )
```

The subroutine must be published using the CATALOG command before it can be called in this way, unless the calling routine resides in the same source file.

For example, you can now create a new simple calling routine to call the new subroutine and display the details that it returns:

```
PROGRAM ShowTitle2

* Get Title details

EQU DETAIL.DESC          TO 1
EQU DETAIL.AUTHOR        TO 2
EQU DETAIL.AUTHORNAME    To 3
EQU DETAIL.TYPE          To 4
EQU DETAIL.ISBN          To 5
EQU DETAIL.PRICE         To 6
EQU DETAIL.STOCK         To 7
EQU DETAIL.DEPT          To 8
EQU DETAIL.GENRE         To 9

Details = ""
Error = ""
Call GetTitle( 3, Details, Error )

If Error <> "" Then
  Crt "Error ": Error
  STOP
End

Crt "Description : ": Details<DETAIL.DESC>
Crt "Author      : ": Details<DETAIL.AUTHOR>
Crt "Author Name : ": Details<DETAIL.AUTHORNAME>
Crt "Type       : ": Details<DETAIL.TYPE>
Crt "ISBN       : ": Details<DETAIL.ISBN>
Crt "Stock      : ": Details<DETAIL.STOCK>
Crt "Price      : ": Details<DETAIL.PRICE>
Crt "Dept       : ": Details<DETAIL.DEPT>
Crt "Genre      : ": Details<DETAIL.GENRE>

STOP
```

```
>ShowTitle2
Description : The Duchess of Malfi (abridged version)
Author      : 3
Author Name : John Webster
Type        : TAPE
ISBN        : 0001050478
Stock       : 8
Price       : 10.99
Dept        : ADULT
Genre       : DRAMA
```

This lesson has hardly scratched the very surface of UniBASIC. But once you can create subroutines you are well on the way to unleashing the power of the language.

In the final chapter, you will add the facility to call your subroutines from the client.

Chapter 5: Integrating Server Code

Contents:

- Calling subroutines
- Server side updates
- Server side validation
- Calling Internal Subroutines
- Locking
- Variable Scope and Persistence

Objectives

In this chapter you will complete your UniVerse or UniData client application by replacing the local functionality with calls to business logic on the server. You will learn how to update the database, and how to perform simple validation.

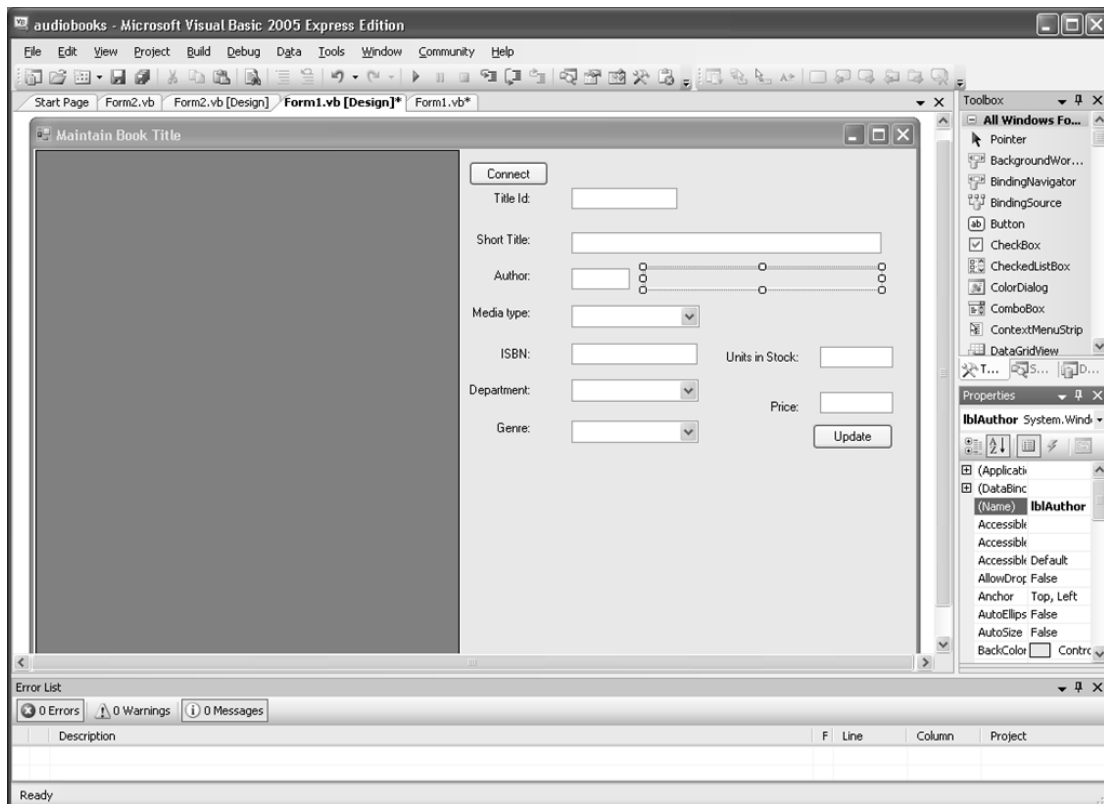
Calling Subroutines from the Client

It is always a good idea to test your subroutines in the Command Shell, wherever you will be calling them from: any errors are more easily spotted and corrected.

Once your subroutine is working, you can call it directly from your UniObjects.NET application by using a UniSubroutine Object.

In this final chapter, you will replace the UniFile Object with a call to your subroutine to get the title details - including the author name.

To prepare for this, reopen your audiobooks project and add a label beside the author that will display the author name. Call this label **IblAuthor**.



The UniSubroutine Object

UniObjects encapsulates a call to a server side UniBASIC subroutine through the **UniSubroutine** Object. This encapsulates the same details as the Call statement you used in the last chapter, allowing you to set the arguments, call the subroutine and examine the return values.

The UniSubroutine Object, like all the other UniObjects classes, is instantiated through the session to which it belongs. The **CreateUniSubroutine** method requires both the name of the published subroutine, and the number of arguments it expects:

```
Try
    Subr = Sess.CreateUniSubroutine("GetTitle", 3)
Catch ex As Exception
    MsgBox("Cannot find subroutine")
    Return False
End Try
```

Each incoming subroutine argument must be supplied to the UniSubroutine call by using the **SetArg** method. This sets the arguments by ordinal position, though following standard .NET conventions, this time the arguments are numbered from zero:

```
Subr.SetArg(0, TitleId)
```

The call to the server subroutine is made through the aptly-named **Call** method. If the subroutine cannot be found, or there is an error whilst running the subroutine, the UniSubroutine Object will raise an exception.

Assuming all went well, the resulting changes to the subroutine arguments can be extracted using either of two methods: **GetArg** or **GetArgDynArray**. GetArg returns the argument as a string and GetArgDynArray returns the argument as a UniDynArray Object, as below:

```
Dim Details As UniDynArray

Subr.Call()
Details = Subr.GetArgDynArray(1)
```

Generally you would use GetArgDynArray only where you know that the argument passed back will contain multiple fields and/or values.

Adding the Subroutine Call to your Code

In the last chapter you created the `GetTitle` subroutine to return the details of a book title, and tested that using a server based calling routine. To use the same subroutine call to return the title details to your form in place of the `UniFile` Object, open your maintenance form in code view and add a new function named `ReadDetails()` to your form as below:

```
Function ReadDetails(ByVal TitleId As String) As Boolean

    Dim Subr As UniSubroutine
    Dim Details As UniDynArray

    Try
        Subr = Sess.CreateUniSubroutine("GetTitle", 3)
    Catch ex As Exception
        MsgBox("Cannot find subroutine")
        Return False
    End Try

    Subr.SetArg(0, TitleId)
    Subr.Call()

    Details = Subr.GetArgDynArray(1)

    txtTitle.Text = Details.Extract(1).ToString
    txtAuthorId.Text = Details.Extract(2).ToString
    lblAuthor.Text = Details.Extract(3).ToString

    cboType.Text = Details.Extract(4).ToString
    txtISBN.Text = Details.Extract(5).ToString
    txtPrice.Text = Details.Extract(6).ToString
    txtStock.Text = Details.Extract(7).ToString
    cboDept.Text = Details.Extract(8).ToString
    cboGenre.Text = Details.Extract(9).ToString

    Return True

End Function
```

Notice that this function mirrors the `ReadTitle()` function currently used to populate the form.

Next, change the **LostFocus** method on the **txtID** textbox to call the new `ReadDetails` function in place of the existing `ReadTitle` function.

```
Private Sub txtID_LostFocus(ByVal sender As Object, ByVal e As
System.EventArgs) Handles txtID.LostFocus
    If txtID.Text <> "" Then
        If ReadDetails(txtID.Text) = False Then
            txtID.Text = ""
            txtID.Focus()
        End If
    End If
End Sub
```

Change the browse list also to use the new function also, and you can now test your form and ensure that it returns the author name alongside the other information from the GetTitle subroutine:

Maintain Book Title -2433 wychbooks From brian

_ID	SHORT_TITLE	AUTHOR_NAME
1	Just William: No. 6 (BBC Radio Col...	Richmal Crompton
2	Just So Stories	Rudyard Kipling
3	The Duchess of Malfi (abridged ve...	John Webster
4	Great Expectations	Charles Dickens
5	The Importance of Being Earnest	Oscar Wilde
6	The 7 Habits of Highly Effective P...	Stephen R. Covey
7	The American Boy	Andrew Taylor
8	James and the Giant Peach (Puffin...	Roald Dahl
9	Hogfather	Terry Pratchett
10	Hancock a Comedy Genius (BBC ...	Cast
11	I'm Sorry I Haven't a Clue: Vol 8 (B...	Cast
12	Friends, Lovers, Chocolate	Alexander McCall Sm...
13	The Legend of Spud Murphy	Eoin Colfer
14	Farmer Giles of Ham and Other Sto...	J. R. R. Tolkien
15	The Lord of the Rings: Complete &...	J. R. R. Tolkien
16	The Navy Lark: Taking Some Libe...	Cast
17	The Worst Witch: Complete and U...	Jill Murphy
18	Friends, Lovers, Chocolate	Alexander McCall Sm...
19	Harry Potter and the Chamber of S...	J.K. Rowling
20	Grass Roots Management	Guy Browning
21	Just A Classic Minute: v. 2	Just a Minute Cast

Title Id:

Short Title:

Author:

Media type:

ISBN: Units in Stock:

Department:

Genre: Price:

Server side updates

In the previous chapter you learned how to read a record from a file using a BASIC subroutine. To update the database, you can perform the opposite action by using a WRITE statement:

Write record on FileVariable, Id

Using either mvDeveloper or Notepad, connect to the server and create a new UniBASIC subroutine that will update the database as follows:

```
SUBROUTINE WriteTitle(TitleId, Details, Error )
* Write back title details

    EQU DETAIL.DESC      TO 1
    EQU DETAIL.AUTHOR    TO 2
    EQU DETAIL.AUTHORNAME To 3
    EQU DETAIL.TYPE      To 4
    EQU DETAIL.ISBN      To 5
    EQU DETAIL.PRICE     To 6
    EQU DETAIL.STOCK     To 7
    EQU DETAIL.DEPT      To 8
    EQU DETAIL.GENRE     To 9

    Error = ""

    Open "BOOK_TITLES" To BOOKTITLES Else
        Error = "Cannot open the BOOK_TITLES File"
        RETURN
    End

    Read TitleRec From BOOKTITLES, TitleId Else
        Error = "Cannot read title."
        RETURN
    End

    TitleRec<1> = Details<DETAIL.DESC>
    TitleRec<2> = Details<DETAIL.AUTHOR>
    TitleRec<6> = Details<DETAIL.TYPE>
    TitleRec<8> = Details<DETAIL.ISBN>
    TitleRec<5> = Details<DETAIL.STOCK>
    TitleRec<9> = IConv(Details<DETAIL.PRICE>, "MD2")
    TitleRec<11> = Details<DETAIL.DEPT>
    TitleRec<12> = Details<DETAIL.GENRE>

    Write TitleRec On BOOKTITLES, TitleId

    RETURN
```

Compile the subroutine and remember that you need to use the CATALOG command to publish the subroutine before it can be called.

You can now change your client to call the WriteTitle routine when performing the update. Because you will not be using the Record property of the BOOKTITLES UniFile to hold the changes to the record, you need to create a new UniDynArray to build the details.

```
Private Function WriteTitle() As Boolean
    Dim Subr As UniSubroutine
    Dim Details As UniDynArray
    Dim Err As String = ""

    Details = Sess.CreateUniDynArray("")
    Subr = Sess.CreateUniSubroutine("WriteTitle", 3)

    Details.Replace(1, txtTitle.Text)
    Details.Replace(2, txtAuthorId.Text)
    Details.Replace(4, cboType.Text)
    Details.Replace(5, txtISBN.Text)
    Details.Replace(6, txtPrice.Text)
    Details.Replace(7, txtStock.Text)
    Details.Replace(8, cboDept.Text)
    Details.Replace(9, cboGenre.Text)

    Subr.SetArg(0, txtID.Text)
    Subr.SetArg(1, Details.StringValue)
    Subr.Call()

    Err = Subr.GetArg(2)
    If Err <> "" Then
        MsgBox(Err)
        Return False
    End If

    Return True
End Function
```

Test this by making some changes to one of the book titles and ensure that they are written back correctly.

Server Side Validation

Before you perform any updates to the database, it goes without saying that you should validate the content before it is written. Some validation should be performed on the client in the first instance, but it always pays to double check.

Most validation is performed using some variation of the BASIC IF statement. This performs a test and branches if the test is met. Notice that the IF statement follows the same branching structure as the Read and Open statements you have already met.

Using this, we can add some simple validation to a write subroutine to test that some of the values are correctly numeric.

- The Num() function returns true if the argument is numeric.
- The Len() function returns the length of an argument.
- The Match operator tests against a simple pattern match.
- The Not() function returns the Boolean opposite of the test.

```
If Not( Num( Details<DETAIL.STOCK>)) Then
    Error = "Stock level must be a number."
    Return
End
If Details<DETAIL.STOCK> < 0 Then
    Error = "Cannot have negative stock."
    Return
End
If Len(Details<DETAIL.ISBN>) <> 10 Then
    Error = "ISBN must be 10 characters long."
    Return
End
If Not((Details<DETAIL.ISBN> Match "9N'X'") Or (Details<DETAIL.ISBN>
Match "10N")) Then
    Error = "ISBN must be 10 numbers or 9 numbers followed by X."
    Return
End
```

In a moment you will add the validation to your write subroutine. But first we need a quick word about structure.

Calling Internal Subroutines

UniVerse and UniData routines benefit from a structured approach just as much as any other language. As routines grow, partitioning them into small discreet chunks can make them easy to follow and to maintain - however large they may eventually become.

UniBASIC partitions routines into blocks identified by labels. A label is simply a unique name that is entered as the first token on a line, and terminated using a colon. Labels are case sensitive.

UniBASIC code branches internally to a label by using the GoSub statement.

```
GoSub DoThis
GoSub DoThat

RETURN

DoThis:
    * do some work
Return
```

Each internal subroutine completes with a Return statement just like an external subroutine call. To differentiate between these, a useful convention is to use a mixed case Return from an internal subroutine, and an upper case RETURN for the main return from an external subroutine.

So now, you can write a slightly more structured routine to update the title details from your form, first performing some simple validation.

```

SUBROUTINE WriteTitle(TitleId, Details, Error )

* Write back title details

    EQU DETAIL.DESC          TO 1
    EQU DETAIL.AUTHOR        TO 2
    EQU DETAIL.AUTHORNAME    To 3
    EQU DETAIL.TYPE          To 4
    EQU DETAIL.ISBN          To 5
    EQU DETAIL.PRICE         To 6
    EQU DETAIL.STOCK         To 7
    EQU DETAIL.DEPT         To 8
    EQU DETAIL.GENRE         To 9

    Error = ""

    Open "BOOK_TITLES" To BOOKTITLES Else
        Error = "Cannot open the BOOK_TITLES File"
        RETURN
    End

    GoSub Validate
    If Error <> "" Then
        RETURN
    End

    Read TitleRec From BOOKTITLES, TitleId Else
        Error = "Cannot read title."
        RETURN
    End

    TitleRec<1> = Details<DETAIL.DESC>
    TitleRec<2> = Details<DETAIL.AUTHOR>
    TitleRec<6> = Details<DETAIL.TYPE>
    TitleRec<8> = Details<DETAIL.ISBN>
    TitleRec<5> = Details<DETAIL.STOCK>
    TitleRec<9> = IConv(Details<DETAIL.PRICE>, "MD2")
    TitleRec<11> = Details<DETAIL.DEPT>
    TitleRec<12> = Details<DETAIL.GENRE>
    Write TitleRec On BOOKTITLES, TitleId

    RETURN

*-----
* Validate
*-----
Validate:

    If Not( Num( Details<DETAIL.STOCK>)) Then
        Error = "Stock level must be a number."
        Return
    End
    If Details<DETAIL.STOCK> < 0 Then
        Error = "Cannot have negative stock."
        Return
    End
    If Len(Details<DETAIL.ISBN>) <> 10 Then
        Error = "ISBN must be 10 characters long."
        Return
    End
    If Not((Details<DETAIL.ISBN> Match "9N'X'") Or (Details<DETAIL.ISBN> Match
"10N")) Then
        Error = "ISBN must be 10 numbers or 9 numbers followed by X"
        Return

```



```
End
If Not(Details<DETAIL.PRICE> Match "1N0N'.'2N") Then
    Error = "Price must be a decimal number."
    Return
End
If Not(Details<DETAIL.DEPT> = "ADULT" Or Details<DETAIL.DEPT> = "JUNIOR") Then
    Error = "Dept must be ADULT or JUNIOR"
    Return
End

Return
```

Recompile the routine and test it by entering a non-numeric value into the stock field. The update should be rejected.

You now have a slightly more watertight means of updating the database.

Adding Locking

There is one final stage before your application is ready for the big time.

Switching the application away from using the UniFile Object to calling subroutines has brought benefits in assembling the details and in building additional validation. But one important final element has been lost – contention management. Your subroutines are not locking the record!

Using a subroutine to perform the update gives you an opportunity to add *optimistic* locking. Under this scheme you would store the original copy of the record when you call the GetTitle subroutine, and send it back to the WriteTitle routine so that you could compare it with the copy on disk and merge any changes that have happened in the interim. This takes work, but is no different from the merge processing needed on any mainstream database³.

In a stateful environment, UniVerse and UniData prefer *pessimistic* locking, as you added to your first version of the maintenance screen by setting the locking strategy on the UniFile object. In this scheme, an operation requests an exclusive update lock when reading a record, and releases that lock after writing the changes to that record. Whilst the lock is held, no other program can lock that record, and no other program can make updates (though they can continue to read the record).

UniBASIC does not use an overarching locking strategy in the manner of UniObjects, but uses a more fine-grained approach. To add pessimistic locking to your application will require just a couple of small changes.

Locks are requested when a record is read. There are two statements in UniBASIC that can be used for reading a record: a READ statement that reads the record without locking and a READU (Read for Update) statement that attempts to apply a lock. If the lock is already held by another session, a READU will either wait for it to become available or branch using an optional LOCKED clause.

The syntax for the locking read is as follows:

```
ReadU Record From FileVariable, Id Locked
    Crt "Record is locked"
End Else
    Crt "Record not found"
    Release FileVariable, Id
End
```

Notice the use of a Release statement to release a record lock.

³ One of the complaints frequently heard from mature SQL developers is that the growth in Wizard-based tools for accessing data has led to a whole generation of so-called SQL 'developers' who do not understand contention management and merge processing.

Even if a ReadU fails because the named record does not exist (yet), the lock is still set on the non-existent key. This may be necessary if, for example, you are writing a new record based on a counter. The RELEASE statement releases the lock without writing.

There is a similar extension when you come to write a record. The WRITE statement will automatically release any held lock, on the basis that most routines read, modify, write and forget about that record. In a persistent environment like a Windows form, this means you must either clear the form as soon as the record has been updated or otherwise protect the lock. A WRITEU statement (Write for Update) will retain any existing lock until it is explicitly cleared with a Release statement.

```
WriteU Rec On FileVariable, Id
```

Using this, you should change the Read in your GetTitle routine to create a lock request when reading the title record:

```
ReadU TitleRec From BOOKTITLES, TitleId Locked
  Error = "This record is locked."
  RETURN
End Else
  Error = "Cannot read title."
  Release BOOKTITLES, TitleId
  RETURN
End
```

So you are now protected against accidental updates, right? Not quite: there is just one last task to perform.

Variable Scope and Persistence

Locks are held against open files. As soon as the file is closed, any remaining locks are automatically released. This was an original feature of the MultiValue mode to protect the database against lazy programmers - but frankly it is a bad practice, and it raises a problem in this instance.

A file is closed when the file variable goes out of scope. If the file variable is owned by an external subroutine, the variable scope is the lifetime of the subroutine call: when the subroutine returns the variable is destroyed, the file is closed and the lock disappears with it!

If you want to retain the lock you need a way to keep the file open. And the easiest way to do that is to use a shared variable to hold the file reference.

All regular variables in UniBASIC have item scope. This means that they are visible within the program or external subroutine in which they are used, and are destroyed when the routine ends.

There is an exception to this rule: shared or COMMON variables. COMMON variables have global scope: they belong to the login session, not to an individual routine. A file opened to a COMMON variable will remain open until the session ends (or you explicitly close the file). This means any locks will not be automatically released until you disconnect.

Creating a Common Block

Unlike regular variables, COMMON shared variables are declared as part of a named block. You can set up any number of common blocks by using the COMMON statement in your code:

```
COMMON /name/ variable[,variable]
```

For example:

```
COMMON /MyCommon/ FIRST, SECOND, THIRD
```

It is conventional to name common variables in upper case, to prevent them being confused with local variables.

On UniData the name of the common block cannot be more than 7 characters. You can declare a longer name but it will be truncated.

Placing File Variables in Common

Variables held in common persist and can be shared between programs and subroutines. By placing the file variable into a named common block in the GetTitle routine, it remains in memory for the duration of the session so that any locks set are not automatically released when the subroutine completed.

```
SUBROUTINE GetTitle(TitleId, Details, Error )

* Get Title details

EQU DETAIL.DESC          TO 1
EQU DETAIL.AUTHOR        TO 2
EQU DETAIL.AUTHORNAME    To 3
EQU DETAIL.TYPE          To 4
EQU DETAIL.ISBN          To 5
EQU DETAIL.PRICE         To 6
EQU DETAIL.STOCK         To 7
EQU DETAIL.DEPT          To 8
EQU DETAIL.GENRE         To 9

COMMON /TITLEDemo/ BOOKTITLES

Open "BOOK_TITLES" To BOOKTITLES Else
  Error = "Cannot open the BOOK_TITLES File"
  RETURN
End
...
```

The same common block can also be used by the WriteTitle routine. This also means it does not have to reopen the file:

```
SUBROUTINE WriteTitle(TitleId, Details, Error )

* Write back title details

      EQU DETAIL.DESC          TO 1
      EQU DETAIL.AUTHOR        TO 2
      EQU DETAIL.TYPE          To 3
      EQU DETAIL.ISBN          To 4
      EQU DETAIL.PRICE         To 5
      EQU DETAIL.STOCK         To 6
      EQU DETAIL.DEPT          To 7
      EQU DETAIL.GENRE         To 8

      Error = ""
      COMMON /TITLEDemo/ BOOKTITLES

* No need to open the file .. it is already there

      GoSub Validate
      If Error <> "" Then
        RETURN
      End
...
```

Now when you run your maintenance form, the lock remains and your records are protected. You can use the LIST.READU EVERY command to display the READU locks in the lock table:

```
01 LIST.READU EVERY
```

Active Group Locks:

Device....	Inode.....	Netnode	Userno	Lmode	G-Address.	Record Locks	Group ...RD	Group ...SH	Group ...EX
1075829046	255068497	0	64259	9 IN	D000	1	0	0	0

Active Record Locks:

Device....	Inode.....	Netnode	Userno	Lmode	Pid	Item-ID.....
...						
1075829046	255068497	0	64259	9 RU	1276	3

Just remember to release the lock when you no longer require it!

Conclusion

Congratulations, you have reached the end of this guide!

If you followed the steps carefully you should now have a working and robust client/server application for your U2 database. More important, along the way you have discovered the UniVerse and UniData file systems and dictionaries, learned some UniObjects coding and written your first server based routines. And hopefully begun to think of some of the advantages the MultiValue platform can present.

That's a lot of ground to cover!

Next Steps

This guide was intended to give you the first baby steps to creating your own UniVerse or UniData applications. From here you can start to explore some of the resources available for developers and add to the knowledge of what you have learned so far.

The Incubator Project takes this a stage further by presenting demonstration applications using U2 technologies, and can also be found on the U2UG website.

From the Author

Thank you for taking the time to complete this guide. Please tell us what you thought of this: we would really like to know!

You can send feedback to the author at:

info@brianleach.co.uk.

Second, you can start to explore some other sources:

- You can find self paced materials and training courses, along with free utilities and example code on the author's website at: www.brianleach.co.uk
- You can visit the Welcome forum on the U2UG website.
- You can find a Welcome forum, knowledge base articles, tutorials, wiki, technical forums, incubator projects and other assistance on the U2UG website at www.u2ug.org.
- You can find white papers and case studies on the Rocket web site at www.rocketsoftware.com/u2
- You can find help and assistance through the U2 User email lists: joining details are on the U2UG website.
- You can contact your local U2 or MultiValue group.

And once you feel confident, you can start to offer assistance to others.

Thank you for your attention, and may I wish you all the best with your UniVerse and UniData projects.

