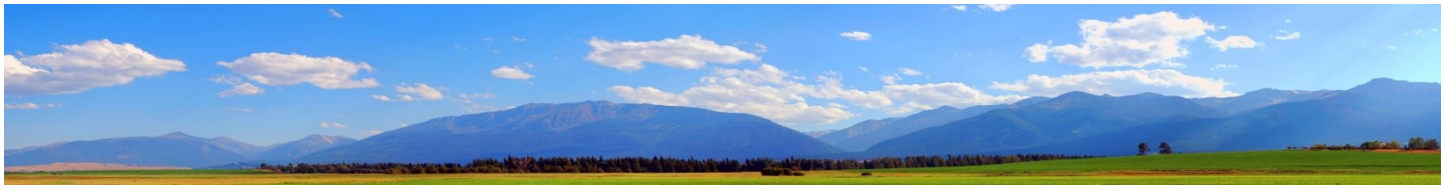


MultiValue Bookstore

Sample Application

General Guide

Brian Leach Consulting Limited

http://www.brianleach.co.uk

Contents

Welcome to the Book Store

Welcome to the MultiValue Bookstore, a small boutique outlet for popular books and audio publications with a friendly and knowledgeable staff to help you choose your reading.

The Bookstore has been envisaged as a showcase for the Rocket MultiValue platforms, a central repository of examples, explanatory materials and resources through which developers can explore the potential of the MultiValue data model and can view working samples of some of the many techniques available for building real-world applications leveraging MultiValue technologies.

The Bookstore will, over time, come to hold a wide range of applications and solutions, all of which are founded on a single database and that represent a single fictional company, “U2 Books”.

By using a single database, this frees up contributors from having to worry about creating their own demonstration data and should simplify the learning curve for developers wishing to explore and to compare amongst the range of different technologies on offer.

Exploring the Bookstore

The MultiValue technologies are very wide ranging, encompassing different data models, middleware APIs, target languages, data access techniques and application generation tools - and so can be daunting to developers who are encountering these for the first time. Even developers seasoned in other products that share the same data model, can find the sheer weight of the available features overwhelming.

For most, the problem is not that these technologies are difficult in themselves: the difficulty arises in simply knowing which of these approaches to use for a given situation.

The Bookstore is therefore intended to give a flavour of these various options, by presenting a range of different applications all sharing the same database. By comparing these applications, and by examining how they are built, new and existing developers can assess the various strengths and weaknesses and discover techniques that they can adapt.

The Bookstore therefore contains:

- A core database representing the fictional Bookstore.
- Additional databases holding the server-side components for each sample application.
- Client side applications showing different data access or application development techniques.

Each of these applications is accompanied by documentation explaining how the application has been put together and pointing out any techniques of special interest.

In addition, this General Guide gives information on the Bookstore as a whole, and background information on the MultiValue data model.

Who Should Read This Guide

This Guide has been prepared with three different audiences in mind:

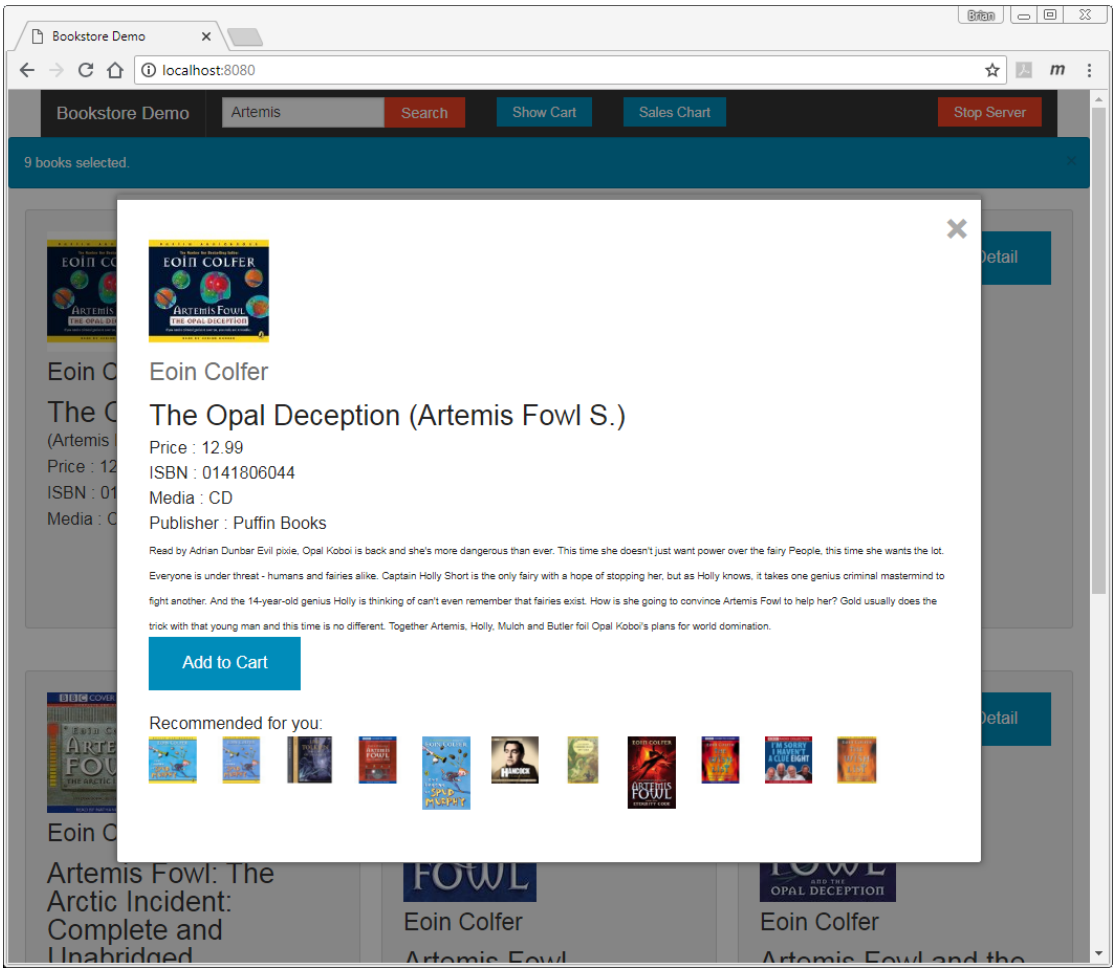
- Developers who are experiencing the MultiValue technologies for the first time, and who are unfamiliar with the underlying MultiValue Database (MVBDMS) data model.
- Developers who are new to the Rocket technologies but who have previous exposure to other MVDBMS platforms.
- Developers who are familiar with the MultiValue technologies but who wish to extend their technical knowledge into new areas.

The General Guide has therefore been divided into the following sections:

- General Introduction
  This explains the thinking behind the Bookstore. You have already read this!
- Installing the Bookstore
  This explains where to find the bookstore application and how to get up and running. Recommended for all readers.
- Fundamental Concepts
  This explains the fundamental concepts that underlie the MultiValue Technologies in particular and the MultiValue Database model (MVDBMS) in general. Developers who have worked with MVDBMS platforms can skip this section.
- The MultiValue Bookstore Database
  This explains the layout of the demonstration database and the history of the fictional company, “U2 Books”. This is recommended reading for anyone looking at the samples.

Installing the MultiValue Bookstore

This chapter provides instructions on downloading and installing UniVerse Personal Edition, the Client Tools required and the MultiValue Bookstore sample applications.



Bookstore Application (Python web server sample).

Downloading UniVerse Personal Edition

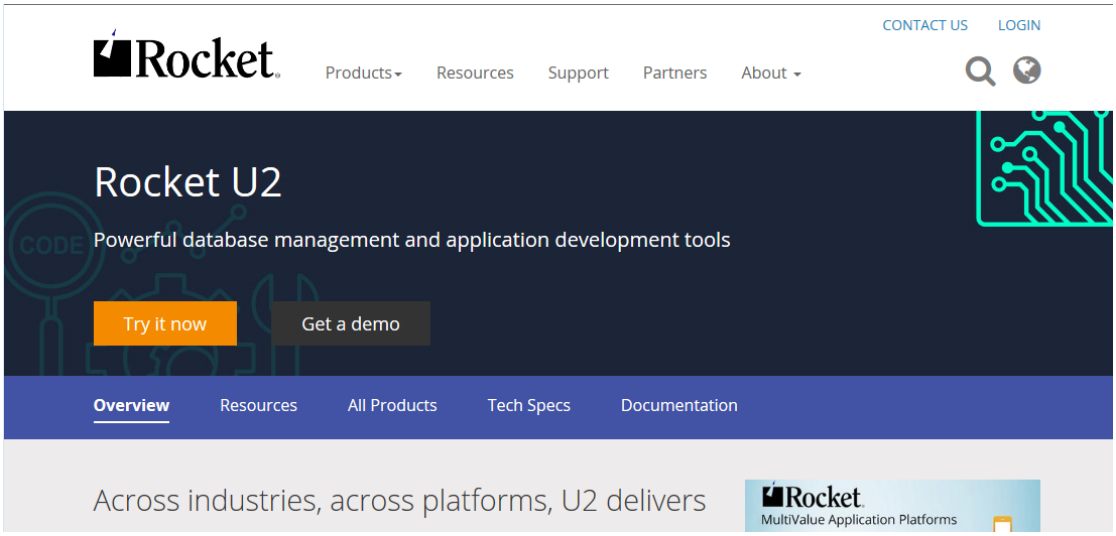
The MultiValue Bookstore database is primarily targeted at UniVerse and designed to work within the limitations of the Personal Edition on Windows.

Both UniVerse and UniData are supported on a variety of different operating systems, but the MultiValue Bookstore database and the Bookstore are designed for a Microsoft Windows platform. If you already have a licensed copy of UniVerse or UniData on Windows, you can use that version to run the MultiValue Bookstore database. If not, you can download the free Personal Editions from the Rocket website.

The Personal Edition of UniVerse is a fully functional product that is restricted to two concurrent users and that may only be used for non-commercial purposes, including training and demonstrations.

To download a copy of the UniVerse Personal Edition, navigate to the Rocket Software MultiValue page at: www.rocketsoftware.com/U2

Select the Try it Now button for a list of the tools and editions that can be downloaded.



You should download **both** the ***UniVerse Personal Edition for Windows*** and the accompanying ***UniVerse Clients for Windows*.**

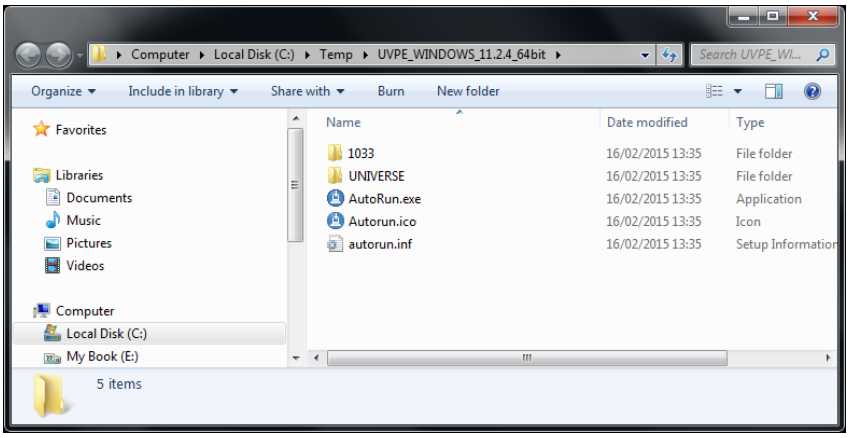
NOTE
We know that websites and locations can change, so for the latest information on how to obtain and download UniVerse Personal Edition and Tools, please see:

http://www.mvdeveloper.com/info/install_uvpe.htm

Installing the UniVerse Database Server

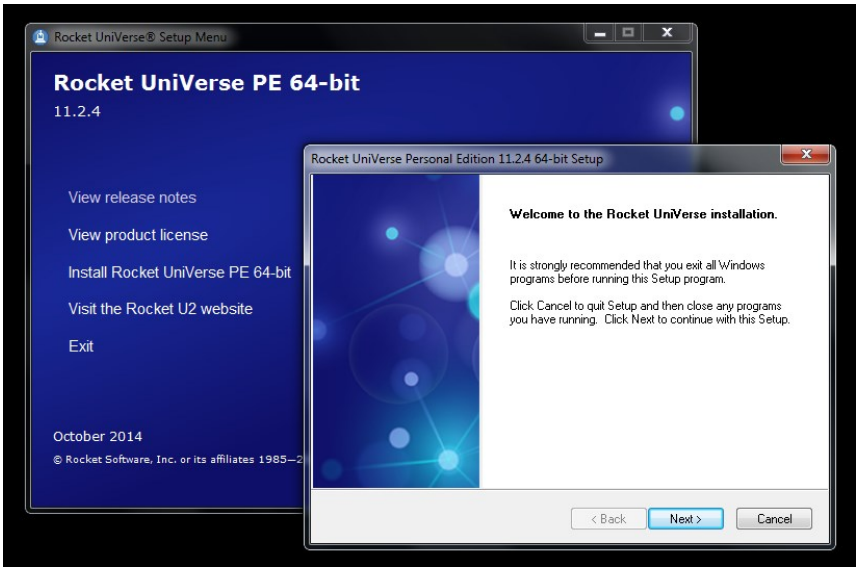
Once you have downloaded the two packages, you can proceed to install the UniVerse server.

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, unzip the UniVerse Personal Edition package to a local folder (e.g. c:\temp) and find the file named autorun.exe.



Right click this and select **Run As Administrator**.

This will display an installation menu like the one below from which you can install UniVerse PE 64 bit:



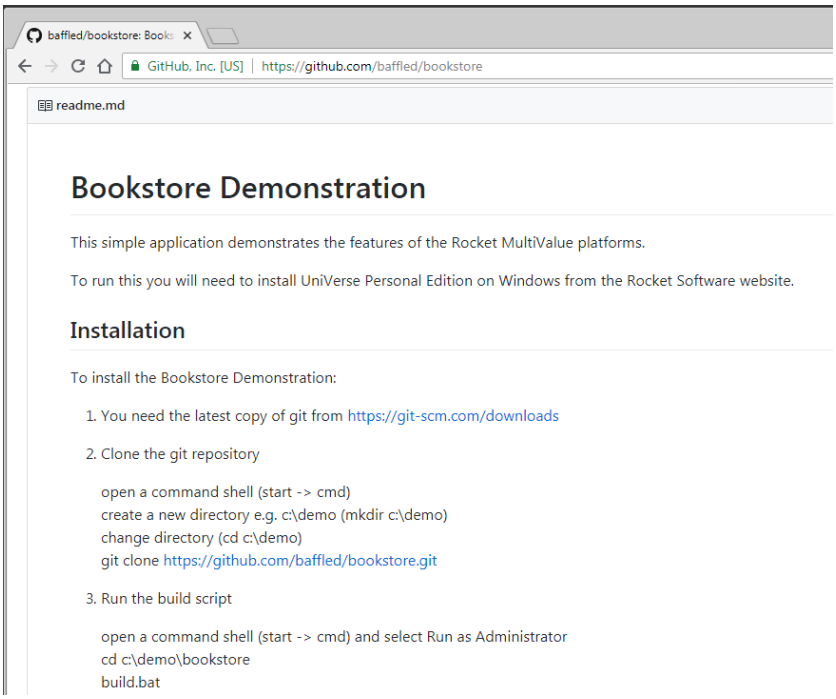
Do not select the NLS (National Language Support) or UVNET options.

Installing the MultiValue Bookstore Database

The Bookstore can be found on github at:

https://github.com/baffled/bookstore.git

You should clone or download this following the instructions in the README.MD file on the bookstore site:



Once you have run the build command, you will have a working UniVerse application installed in a UniVerse\mv_books folder next to wherever you downloaded and ran the build script.

From here you can explore the application using the interactive administration shell by typing the command **start_uv**, or using a simple web interface by typing the command **start_webserver**.

The UniVerse Command Shell

The UniVerse Command Shell provides a text based interface to a UniVerse 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 UniVerse 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 (>), known as the TCL (Terminal Control Language) prompt, ready for you to enter a command. The good news - most UniVerse 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. UniVerse will answer with a response similar to the one below:

```
>WHO
1292 mv_books From administrator
```

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 (mv_books) tells you in which account (or workspace) 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 (administrator) 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.

You may find that your keyboard operates in a different case when connected to UniVerse. This is an idiosyncratic feature left behind by the original UniVerse developers: most UniVerse commands are in upper case and most UNIX commands in lower case, so 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:

PTERM CASE NOINVERT

Disconnecting Safely

When you have finished with UniVerse, it is important that you disconnect safely. This will ensure that any resources you have used are released and that UniVerse can 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 because UniVerse believes you are still connected.

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

Type the **OFF** command to finish your session.

Fundamental Concepts

This chapter will familiarize you with the fundamental architecture of a UniVerse database application, and will introduce the ways in which UniVerse stores 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.

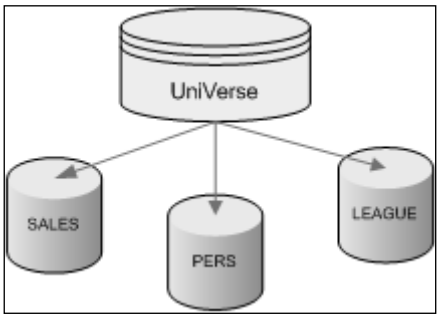
If you are already familiar with the MultiValue Database Model (MVDBMS) you may skip this section.

UniVerse Accounts

UniVerse has a deceptively simple architecture.

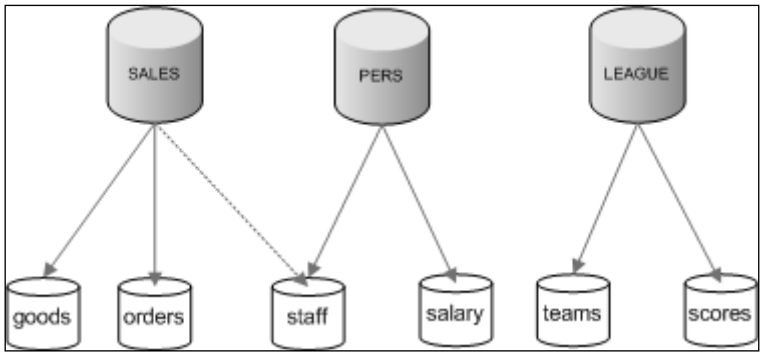
At the top level, a UniVerse system is divided into a number of separate areas known as Accounts. Each Account is a kind of work space, that provides access to a set of information and the commands needed to work with that information. An account might contain an application, a set of data, development code or just a subset used by one department. An account may also contain a database schema.

The rules for how a system should be divided into accounts are, like many things where the UniVerse 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 of 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 separate directory in the underlying file system. Logically, the arrangement is more flexible. Accounts can freely reference information held in other accounts, making the organization more of an administrative feature.

An order processing application, for example, 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 the other personnel information such as workers' salaries or directors' expenses.



Account Flavors

UniVerse may be a thoroughly modern business platform, but there are times when its long history catches up with it. One of those concerns the Account Flavor.

UniVerse began its life as an emulator. Its original purpose was to enable people to migrate applications from a variety of proprietary and closed MultiValue database platforms into a single open and UNIX based environment. Just like the various SQL dialects today, different MultiValue database manufacturers had introduced differences into their syntax, which meant UniVerse needed a way to support these variations. Fortunately, these generally fell into a number of 'families', and so each account emulates the syntax used by one of these families.

The impact of this today is that each account on a UniVerse system has a particular 'flavor', which sets the various syntax options. So an account determines not only the information that you can see and the applications that you can run, but also defines how some of the commands that you use may be interpreted.

Fortunately most of the standard commands and most of the data retrieval languages operate transparently across all account flavours: but some of the other commands you will meet may differ slightly when you come to use them on your live systems.

You can view the account definition entry for your account by using a special form of the statement stack **.L** command:

```
.L RELLEVEL
```

This displays the release level of UniVerse and the account flavor.

UniVerse Files

UniVerse stores data and programs in structures called Files.

> **Files and Tables**
>
> As a developer you are probably already familiar with the concept of database tables. UniVerse files and tables may look similar, but beware - they are not the same thing!
>
> UniVerse implements a SQL data model over the top of its native MultiValue data model: the SQL model adds schema and constraint information to files to create SQL tables. However, the use of SQL is very rare amongst UniVerse developers.

You can think of a UniVerse file as being very similar to a filing cabinet in an office. A filing cabinet typically contains many records - sheets of information often organized into formal structures. Each set of cabinets may be dedicated to holding just one type of record: customer details, invoices, despatch notes or similar.

UniVerse 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 (but not always) contain similar information.

A single UniVerse 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 record instantly.

Listing the Files in your Account

Because UniVerse holds all its data in files, the first stage when looking for information is to discover the files that 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 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 may be held in other accounts.

Navigating through Listings

The LISTF command is displayed one page at a time, using the number of lines UniVerse expects your terminal emulator to support. This is true of many TCL commands when run through the command shell.

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

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

For most commands, this prompt gives you 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.

There are various means to look at the data held in a file, some of which can be generally more helpful than others. In a perfect world (or a demonstration application), it should be possible for you to 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.

For example, type the following command to see the books in stock:

```
LIST U2_BOOKS
```

Records and Fields

Each UniVerse record contains several pieces of information. To distinguish between each of these pieces of information, each record is divided into fields. A record may have no fields, one field or any number of fields.

The LIST command creates a columnar listing of various selected fields from the records.

Files and Dictionaries

The default listing of a file (using LIST filename) may or may not show you anything useful: you may get lucky and see the information you need included in the display in just the way you want to see it, but in most cases you will need to discover what other information is being held in the file.

This introduces a defining aspect of the UniVerse database: the File Dictionary.

MultiValue databases have a number of pronounced features that separate them from the rest of the database market, one of the most important of which is the use of File Dictionaries. Each UniVerse 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 the Metadata that describes the content of the file. This Metadata is the key to understanding the enquiry languages and other structures used by UniVerse.

UniVerse divides records into fields. Each field generally holds one piece of information: the U2_BOOKS file, for example, contains records defining each title in the bookstore catalogue. Each title record contains fields holding the short title of the book, the author, the ISBN number, genre, department, price and stock level amongst others.

Whenever the LIST command encounters a file, it looks at the file dictionary to find the information for each of the fields it needs to display.

Listing a File Dictionary

The file dictionary is always referenced using the name:

```
DICT filename
```

You can list 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
```

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. These define the real fields that occupy set positions in each record: the position number is given in the next column.

You can use the familiar LIST command to display individual columns, following the syntax:

```
LIST filename field
```

If you 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 variant of the MultiValue enquiry language called RetrieVe and a variant of SQL. Both use the same dictionaries and can be used largely interchangeably. Most UniVerse users prefer RetrieVe which is more English-like and has a friendlier syntax.

Records and Keys

When you list the content of a file using 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 file must have a unique identifier, or record key. UniVerse uses the key when storing and retrieving records, so if two records are supplied with the same key the second will overwrite the first.

Internally, UniVerse applies a hashing algorithm to the key to determine the storage location. In a well-maintained system, accessing specific records by key is instantaneous, regardless of the size of the file and number of records held. UniVerse applications can work directly with record keys to give a very fast and light response. For this reason, MVDBMS developers generally choose meaningful keys: a customer account number, a short descriptive word, a date.

Dictionary Definitions

UniVerse dictionaries go further than simply defining the file content. The real power of the dictionary lies in the fact that it can contain other content than pure field definitions. In particular, Dictionaries contain definitions of virtual fields: fields that 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 building derived information.

Look further down a dictionary listing and you will find another series of definitions, this time identified with an I (Interpreted) in the Type column. These are the calculated fields.

A calculated field may, for example, report difference between the number of units in stock and the number that have been ordered; define pricing calculations, tax, line totals, order balances and similar financial operations.

Virtual fields are not restricted to numeric calculations. They can also be used perform complex functions, to access remote data, to combine data for selection or to reformat data for display and selection.

Because they are held in the dictionary they are available for any enquiry commands without the need to reproduce the column expressions that make up their calculation separately for each enquiry statement.

Related Fields

One particularly powerful feature of virtual fields, is that the data they present can be sourced from other files – similar to using a lookup in a spreadsheet. A virtual field can present data as if it were part of the file, when in reality it is held somewhere else. This makes database enquiries very much simpler than using join clauses!

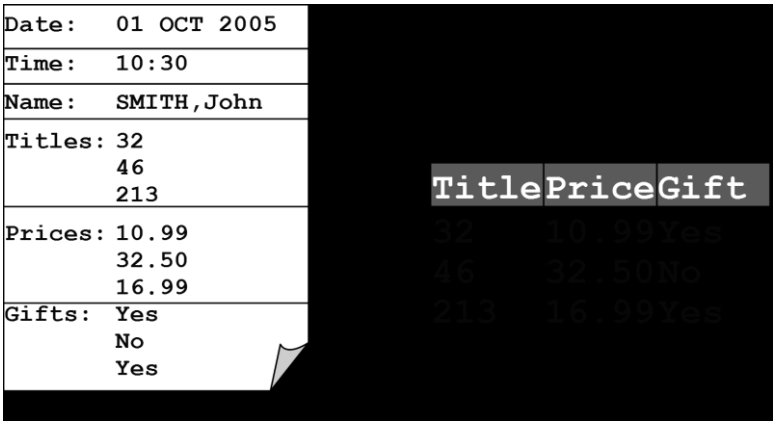
MultiValued Data

Most databases use flat representations of data: information is stored in two dimensional tables of columns and rows, like a spreadsheet. 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 item line. This is known as a 'Master-Detail' or 'Parent-Child' relation.

The problem with this format is that in most cases you have relate these all of these rows back together again whenever you need to reassemble that sales order for display. This need to relate information together is the heart of the 'relational database' model. It is also 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 sidestep many of these issues by allowing each field to hold more than one value: an arrangement that more closely matches the way information is handled in real life.

A MultiValued Database does not need to use two separate files to hold master/detail records, such as a sales order. In such a 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 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.

Historically some MultiValued databases have been poor at handling sub-values for reporting, and thus the convention has arisen that sub-values are normally reserved for programmers only and are rarely used for enquiry.

XML Listings

Because UniVerse supports repeating values, a listing in UniVerse can behave more like an XML document with nested entities, than a traditional table. So it is not surprising that RetrieVe and UniVerse SQL can both 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.

The **TOXML** keyword converts a LIST or SELECT listing into an XML format.

```
LIST filename TOXML ELEMENTS
```

UniVerse Programming

Any database platform of substance supports some way to define stored procedures that localize complex data operations to run within the database itself. UniVerse and UniData go further than this – they are fully equipped with two substantial programming languages that can be used to build entire applications, and contains features for database operations, device control, printing, exporting, XML navigation and generation, data encryption, socket support, external data access… the list goes on.

Business Language

UniVerse Business Language 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.

Python

For programmers coming from other disciplines, or to take advantage of the huge wealth of features and packages available, UniVerse has in-built support for Python. You can run Python programs to interact with the Universe platform, including calling Business Language subroutines: similarly, you can call Python functions and instantiate classes from within the Business Language to interact the other way.

The MultiValue Bookstore

This chapter provides an overview of the MultiValue Bookstore database used for all the sample applications, and provides some assistance with navigating and exploring the database.

The MultiValue Bookstore and MultiValue Books Company

"MultiValue Books" is a fictional book store with a physical and on-line presence.

The store has been in existence for three years, with a steady growth in sales driven by a combination of new clients – most of whom enrol through the website – and repeat business from existing clients who appreciate their personal service.

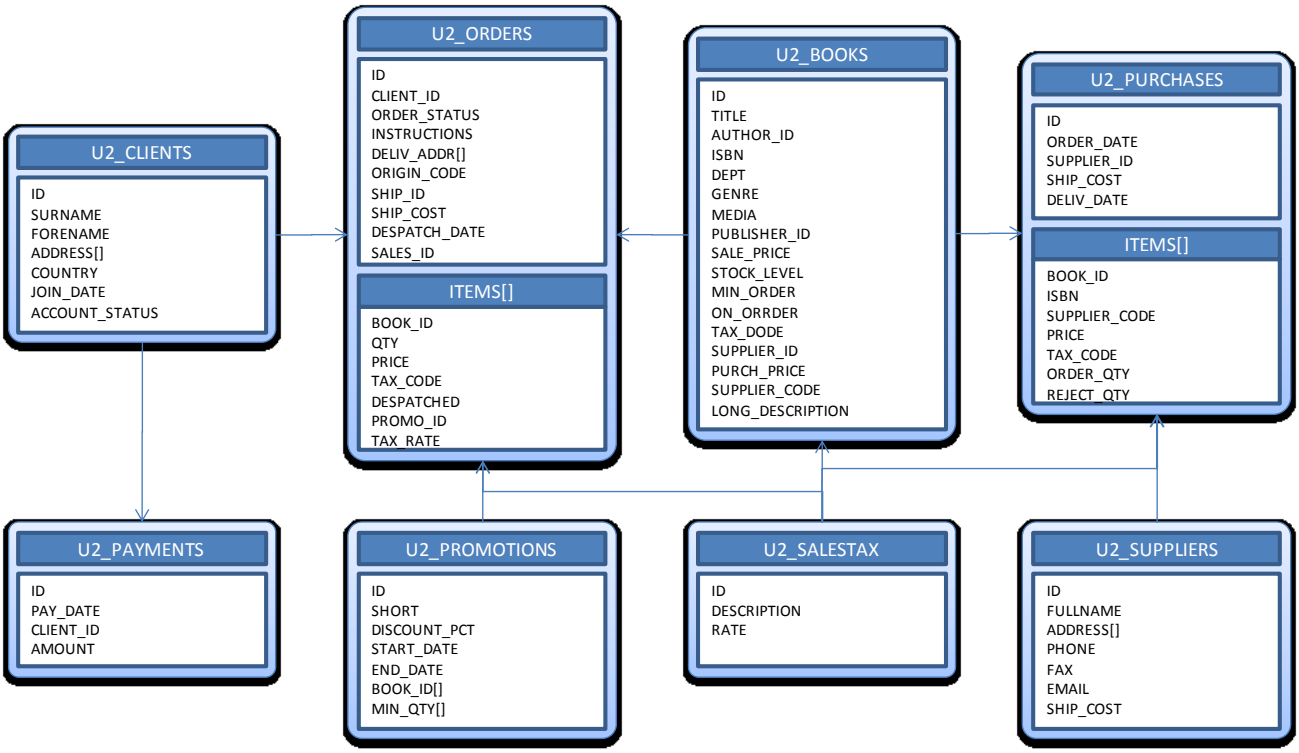
The store maintains a small catalogue of books, including many audio books.

Customers can place sales orders over the phone or web. As a start-up company they need to keep costs down, and so they keep their stock levels to a minimum, generating purchase orders for new stock from daily batch routines that examine the outstanding sales requirements and current stock levels. The books are sourced from a select group of suppliers. Even so, occasionally books are rejected on receipt.

The store has experimented in the past with promotional offers, and continues to look to new technology to help them build their business.

MultiValue Bookstore Database Layout

The following diagram displays the main layout of the MultiValue Bookstore:



Book Details

The Book Store is based on a catalogue of audio and traditional books on stock, held in the U2_BOOKS file. This divides the stock into ADULT and JUNIOR sections, and for audio books also defines the media. Each book has a current stock level and a minimum order quantity set by the supplier: this is used in the automatic purchase order generation.

Regular books exempt from sales tax in the UK, however this is the not the case for audio books. The tax code on the U2_BOOKs entry is linked to the U2_SALESTAX file that holds the rate for each code type.

Authors are identified by their AUTHOR_ID in the U2_BOOKS file. This is a foreign key to the U2_AUTHORS file, where you will find the author name and also a multivalued list of their titles for lookups. The same is true of the publishers, held in the U2_PUBLISHERS file.

Clients and Sales Orders

The sales orders are held in the U2_ORDERS file. For ease, these have been given composite keys: each sales order is identified by the date and time of the sale, followed by a suffix to disambiguate sales orders placed at the same moment. This prevents the need to maintain an accumulator or auto-numbering scheme for new sales orders.

Each sales order is associated with a client, with the client details being held in the U2_CLIENTS file. The client details record the client joining date and contact details. An index of the client orders is maintained manually by the system, using the U2_CLIENT_ORDERS file.

The sales orders contain single valued headed information, including the client identifier, status, delivery details and shipping code. The shipping costs and descriptions are read from the associated U2_SHIPPING file. An origin code records whether the order was placed over the web, by phone or via email.

The second part of the sales order is the association that makes up the order lines. Each sales order may have any number of order lines, which are held as a set of multivalued fields. These record the multivalued BOOK_IDS, relating to the U2_BOOKS file, the current sale price when the order was filed, quantity requested and tax code. The order total is not stored, but calculated on the fly taking into account the order line totals, tax lines, shipping costs and any promotions in force.

Promotions are separately defined in the U2_PROMOTIONS file. These offer discounts that come into effect when the client has ordered a given quantity of books from a specified list. For example, readers of Terry Pratchett will receive a 50% discount on his works, where for Stephanie Meyer there is a 100% discount on the second book ordered from the list (i.e. buy one get one free).

Payments are also recorded for clients, though by design these are not directly associated with an order: a client may part-pay an order or a payment may cover multiple orders. Unlike the manual index between the clients and orders, the payments in the U2_PAYMENTS file hold a system index on the CLIENT_ID.

Purchases

As books are ordered, so the company must refresh its stock. It does so by raising purchase orders on a daily basis, though for this demonstration the routines must be run manually – in a live environment these would be constantly running background processes.

Each book is identified with a preferred supplier, using the SUPPLIER_ID field. This is a foreign key to the U2_SUPPLIERS table, where the supplier details and standard shipping costs are stored.

The purchase order creation checks all orders received for the day against the current stock and raises purchase orders to the suppliers: one purchase order per supplier per day for the required stock. These are stored on the U2_PURCHASES file.

Occasionally the wrong books may be sent by the supplier, or the books may not be in a suitable condition. In these cases the books can be rejected, and this also feeds into the current stock level calculations.

This describes the main parts of the MultiValue Bookstore Database. For more details please see the individual samples.