First Steps in Administering UniVerse

U2 User Group Learner Pack

Written for the International U2 User Group
www.u2ug.org

Brian Leach
http://www.brianleach.co.uk
First Steps in UniVerse Administration

Thank you for downloading this guide. Please distribute it freely.

This guide is aimed at students who have downloaded the personal edition of UniVerse or professionals who are starting out with a new or existing UniVerse system and now need to undertake some simple administration tasks. The guide will cover such basic topics as logging on, creating files or administering the file system.

This guide will assist you in taking your first steps as a UniVerse DBA, and to provide a brief overview of what UniVerse administration may entail. Once you have read this guide, you can find more detailed information on each of the subjects highlighted in the UniVerse Administration Guide, part of the UniVerse documentation set. The full UniVerse library is available for (free) download from the IBM web site.

Please note that if you are serious about UniVerse administration, these guides should not be read as a replacement for undertaking full training. Training in UniVerse administration is available both from IBM and from independent third party trainers.

This guide has been written on behalf the International U2 Users Group (U2UG), a volunteer user group promoting the UniVerse and UniData ("U2") product lines. This guide is part of their ongoing commitment to assist new users, and forms part of the U2UG Learner Pack.

You can find additional learning materials on the U2UG web site, located at: www.u2ug.org.

About the Author

Brian Leach is an independent UniVerse consultant and trainer and author of several books on UniVerse. He was named in 2008 as an IBM Information Champion and is a former President of the U2 User Group. Find out more at: www.brianleach.co.uk.

Thank You

I am indebted to my fellow board member Baker Hughes for his diligence in proof reading and for his insightful comments regarding this guide.
# Contents

First Steps in UniVerse Administration .............................................................................................................. 2  
Notes ........................................................................................................................................................................... 4  
Starting Out with UniVerse ....................................................................................................................................... 5  
What is UniVerse? ....................................................................................................................................................... 6  
  Installing the UniVerse Database Server .................................................................................................................... 8  
Starting and Stopping UniVerse ................................................................................................................................. 8  
Connecting to UniVerse ............................................................................................................................................... 9  
Testing your Connection ........................................................................................................................................... 9  
Using the UniVerse Command Shell ........................................................................................................................ 10  
  Leaving the Command Shell .................................................................................................................................. 11  
Terminal Types .......................................................................................................................................................... 12  
UniVerse User Credentials ......................................................................................................................................... 14  
  User Connection Policy ......................................................................................................................................... 15  
Identifying UniVerse Processes (Windows) .................................................................................................................. 17  
Identifying User Processes (UniVerse) ....................................................................................................................... 18  
Creating an Account .................................................................................................................................................. 20  
Navigating Accounts ............................................................................................................................................... 22  
The VOC File ............................................................................................................................................................. 24  
Working with Files .................................................................................................................................................... 25  
Understanding UniVerse File Sizing .......................................................................................................................... 26  
  Administering Files Sizing .................................................................................................................................... 27  
How UniVerse Data Files Work ............................................................................................................................... 28  
File Overflow ............................................................................................................................................................ 29  
File Resizing ............................................................................................................................................................. 30  
Dynamic Files .......................................................................................................................................................... 31  
  How Dynamic Files Work ...................................................................................................................................... 32  
Dynamic File Impacts ............................................................................................................................................... 33  
Simple Analysis and Resizing .................................................................................................................................. 34  
File Contention and the Lock Table ........................................................................................................................... 36  
  Read and Write Locks ............................................................................................................................................... 36  
Lock Table and Performance ...................................................................................................................................... 37  
READU and File Locks ............................................................................................................................................. 38  
Listing Record Locks .................................................................................................................................................. 38  
Printing from UniVerse ........................................................................................................................................... 39  
Changing Terminal Definitions .................................................................................................................................. 41  
Backing Up Your Applications .................................................................................................................................... 43  
Finding More Information ............................................................................................................................................. 44
This page has been left blank for your notes.
Starting Out with UniVerse

You have downloaded your copy of UniVerse Personal Edition, or you have been given the job of understanding how to use a UniVerse system at work. Right now you're probably asking yourself three questions:

- What is this?
- What do I need to know about it?
- Where do I start?

This guide is intended to answer those three questions and more.
What is UniVerse?

UniVerse is a database, and also a platform for powerful and feature rich business applications. It is probably quite different from the databases you have used up until now, especially if you are accustomed to working with mainstream relational databases.

UniVerse is a style of data management system known as a MultiValue Database, usually shortened to MVDBMS. These are information and logic platforms that make it possible to write highly scalable, feature rich business applications for client/server, Internet, service orientated architecture (SOA) and many other situations. MultiValue databases are frequently found as embedded data sources behind leading business, financial, engineering and government systems. They are robust, scalable, low maintenance, low cost and facilitate rapid and changing development requirements, making them essential for end users and less profitable for consultants.

UniVerse is one of the leading MultiValue databases, and together with its sister database, UniData, it has been branded by IBM with the less than searchable moniker of "IBM U2". To add to the confusion, these same databases are also known as 'nested relational' databases, or 'post-relational' databases.

The key distinguishing elements behind UniVerse are:

**Integrated Business Language**

UniVerse features an embedded business language suitable for creating complete applications or for encapsulating complex business rules in client/server or internet based solutions. This removes the need for the three tier logic required for most modern development platforms.

**Nested ("MultiValued") data structures.**

MultiValued data structures support nested entities removing the need to normalize information. These provide highly efficient storage of complex data types, and allow for a more natural representation of real world entities such as sales orders or manufacture assemblies.

**Fast Transactional Middleware**

UniVerse supports its own brand of middleware that embodies the MultiValue data model and exposes the features of the business language. You will also find the regular, relational protocols represented here in the form of ODBC, JDBC, OleDB and ADO.Net: however these require translations between the native MultiValue model used by UniVerse and the relational models they offer. They are only recommended for situations in which the native middleware cannot be employed.
Virtual Metadata

UniVerse, in common with all MultiValue platforms, enforces a strict separation between the data that it stores and the definition used to define it as meaningful information. Whilst it is entirely possible to create SQL tables within an UniVerse system, in most circumstances UniVerse applications will use structures known as File Dictionaries to define the content of a data file.

In addition to defining fields within their associated files, file Dictionaries can also contain synonyms, phrases that act as short cuts to frequently used combinations of fields or columns, and virtual fields whose values are derived at runtime using a simple expression language.

Files

UniVerse stores data in structures known as Files. These are similar to tables in other databases, but need not be homogenous. UniVerse also supports SQL tables, which are stored as UniVerse files but with additional restrictions placed upon their use. File Dictionaries also remove the need for developers to create views on the underlying files and tables.

Unlike many other MVDBMS platforms, UniVerse and UniData both support a range of different file types and models. These range from different storage allocation patterns to optimize record distribution, through to different file structures such as B-Tree, Hashed, Linear Hash and Directory.

You do not need to learn everything about file types before you begin to work with UniVerse. More details about data structures are covered in accompanying Learner Pack article, "Writing your First UniVerse Application".
Installing the UniVerse Database Server

A personal edition of UniVerse is available for download. This is currently available for Windows and Linux platforms, and is a fully functional server available for evaluation and training. There are generally two packages available for the Windows platform:

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

You should download both of these packages.

To install UniVerse you will need to log into your PC using the credentials of a user who has local administrator rights

Starting and Stopping UniVerse

On Windows, UniVerse is supported by a number of services that must be running before you can connect to the database. These are normally not started by the installation, so you will need to start them manually if you want to connect directly following setup.

You can do this from the services list, or from a Control Panel applet. This also sets the authorisation code if you are running a full (i.e. not the personal) edition.

On Windows there are three main services that should be started:

- UniVerse Resource Service
- UniVerse Telnet Service
- UniRPC Service

Older versions also include a fourth service, UniVerse REXEC service, used to support the ODBC and similar protocols. This has been discontinued and rolled into the UniRPC service on current versions because of the greater security provided by UniRPC.

On Linux and UNIX platforms, the installation creates a start-up and shutdown script, named `S99uv.rc` that will be located wherever your particular distribution places its initialization files (typically `/etc/init.d` but your mileage may vary). To start and stop UniVerse manually on these systems you will need to type the command:

```
{path_to}/S99uv.rc start
{path_to}/S99uv.rc stop
```
Connecting to UniVerse

For most administration tasks, and for creating your first databases, you will need to connect to UniVerse using its command shell, also known as TCL (Terminal Control Language). Whilst most UniVerse applications are GUI based, it is easier to perform administration functions using the command environment.

UniVerse is supplied with a terminal service that supports both the regular TELNET and SSH protocols. This can be used with practically any terminal emulator for remote connections. A terminal emulator by the name of Dynamic Connect is provided free with the UniVerse Clients. A more powerful version with a number of helpful tools and more advanced emulations is available as part of the U2 product suite under the name wIntegrate. There are other versions of TELNET clients on the web that can be used, some of which include toolkits for working with MultiValued data.

If neither of these is available, you can use the telnet client that is built into Windows by clicking on the Start button and choosing Run -> Telnet.

Testing your Connection

If UniVerse is loaded onto your local PC or laptop, you can connect using the internal network name localhost. If not, you will need to know the host name or IP address of the server hosting your UniVerse.

To test your connection you can use the regular Windows Telnet client. Click on the Start button, select Run and enter the command:

telnet localhost

or

telnet hostname

where hostname is the name of your server.

You should be presented with a window similar to the one below:
If Telnet reports that it cannot connect, you may need to check your host name and verify that there is no firewall interfering with your connection. TELNET connections are normally made over port 23.

**NOTE:** you can change the port number used by the UniVerse TELNET service to prevent clashes if you are running another TELNET server.

**NOTE:** if you need to ensure secure access to your system, please consult the UniVerse documentation on setting up SSH (secure socket shell) as an alternative to TELNET.

To connect to UniVerse, enter a local Windows user name and password with local administration privileges and when prompted, specify uv as the account name. User credentials, including those for non-local and non-administration users, are discussed below.

**Using the UniVerse Command Shell**

The UniVerse command shell is a text based environment for running commands, stored procedures, batch operations and text based legacy applications.

The command shell will prompt you by presenting a chevron prompt (>).

Commands typed are case sensitive except for the initial word, and are usually entered in upper case. You can check the details on any UniVerse command by calling the UniVerse HELP:

```
HELP command
```

For example,

```
HELP HELP
```

If this does not display correctly, refer to the section on terminal types below.
Leaving the Command Shell

To leave the command shell, type the command **OFF** or **QUIT**.

<table>
<thead>
<tr>
<th>Important</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do not simply close the connection by exiting your terminal window: this may not release your connection from the licensed user count.</td>
</tr>
<tr>
<td>For more information on the command shell and on entering commands, please refer to the accompanying Learner Pack document, &quot;Writing your First UniVerse Application&quot;.</td>
</tr>
</tbody>
</table>
Terminal Types

When you connect to the UniVerse command shell, you do so using a terminal emulator. This enables you to make use of immediate administration commands, much like the CMD shell on Windows: however it also requires that your terminal emulation software understands the control sequences used for such simple operations as clearing the screen or setting the cursor to a given location.

The various term types look back to a day before desktop computing when warehouses and sales offices connected to mainframes through dedicated terminals. The communication protocols used by those proprietary terminals have been broadly enshrined as de-facto standards still used today. The protocols are identified by their historical namesake (see description in table below). UniVerse supports a wide range of terminal types and emulations.

For most uses, the standard vt100 terminal type works best: this is as close to an industry standard default as you will find for terminal handling though typically there are also differences between vt100 emulations. If you are running a legacy application, the chances are that you will need to select a different terminal type as required by your system.

Terminal types are identified by a short, case-sensitive name, for example, vt100 or wyse60. Setting the terminal type affects only your current session.
Setting your Terminal Type

There are two commands that can be used to set your current terminal type: TERM and SET.TERM.TYPE.

The SET.TERM.TYPE command is more helpful in that it gives the option to list all the supported types:

```
>SET.TERM.TYPE

Enter Terminal name (or '?' to list possible choices)?

Available Terminal Types

Name........ Description............................
97801-UV       SIEMENS terminal
a210-adm5     Ampex 210 (emulating Lear Sigler adm5)
a210-hz1410    Ampex 210 (emulating Hazletine 1410)
a210-hz1500    Ampex 210 (emulating Hazletine 1500)
a210-gt102     Ampex 210 (emulating Qume 102)
a210-regent25  Ampex 210 (emulating Adds Regent 25)
a210-tvi910    Ampex 210 (emulating Televideo 910)
a210-tvi910+   Ampex 210 (emulating Televideo 910+)
a210-tvi920    Ampex 210 (emulating Televideo 920)
a210-tvi925    Ampex 210 (emulating Televideo 925)
a210-vp        Ampex 210 (emulating Adds viewpoint)
a210           Ampex 210
aixterm-m      IBM 5151 display (Aixterm Emulator Monochrome)
aixterm       IBM Aixterm Terminal Emulator
```

The TERM command lets you set either the terminal type or the viewport size. The latter controls the page width and depth expected by your terminal, and used by UniVerse when displaying listings or other commands that span multiple screens.

If the TERM command is followed by a number, UniVerse interprets this as the number of characters per line, optionally followed by the page depth:

```
TERM 132,42
```

If the TERM command is followed by a terminal type name, UniVerse interprets this as the terminal type to set (identically to the SET.TERM.TYPE command)

```
TERM wyse60
```

It may well be that the terminal settings do not entirely match what your terminal emulator or application expects. This is quite normal: many terminal libraries have slight differences depending on the revision of the terminal type or the specific emulation. Sometimes you may need to choose a terminal type that is itself an emulation of an emulation: any of the terminal types that in the format name-name reflect an emulation (e.g. a Wyse60 terminal running in vt100 emulation may be slightly different from the original vt100 emulation).

Changing terminal settings will be covered later in this guide.
UniVerse User Credentials

Each connection to a UniVerse system requires authentication in the form of a user name and password.

UniVerse does not provide its own user credentials. UniVerse will use the authentication provided by the operating system on which it runs. However, it does require certain additional information, especially when running on Windows.

UniVerse on Windows will accept any of the following:

- A locally defined user and password, where that user has local administration rights (i.e. is a member of the local administrators group).

- An Active Directory or Domain User with local administration rights and who has been explicitly provided with Log On Locally permission on the UniVerse server. You must specify a fully qualified user name in the form: domain\username.

- A locally defined non-administration user who has been granted Log On Locally rights and where you have entered specific location information or selected a policy that allows access to any database account (see below).

- An Active Directory or Domain User without local administration rights who has been granted Log On Locally rights and where you have entered specific location information or selected a policy that allows access to any database account (see below). You must specify a fully qualified user name in the form: domain\username.

Note:

For many administration tasks, including installing UniVerse, you must have local administration privileges.
User Connection Policy

A UniVerse system is divided into a number of work areas, known for historic reasons as 'accounts'. These are similar to (but not exactly the same as) separate database on other DBMS platforms. Generally, each application resides in a separate account. Accounts and account creation are discussed below.

Before UniVerse can permit access to a user based on their credentials, it examines a connection policy to determine which account (if any) should be the initial connection point for that user.

Note:

All UniVerse systems have a home account known as the uv account. Until you have created more accounts, you will need to login to the uv account to perform administration tasks.

Many administration commands are only found in the uv account.

If the user has local administration rights, the UniVerse telnet service will prompt for the location or name of the account to which the user wishes to connect. A local administrator can therefore connect to any account, subject to additional directory or SQL restrictions.

UniVerse runs as a regular Windows application, so it will be subject to Windows directory security. You can only connect to accounts that reside in directories to which Windows will grant your user access. However, please note that the UniVerse temporary directory defaults to the UVTEMP directory beneath the uv account, and all UniVerse users must have writeable access to that directory.

If the user does not have local administration rights, the login policy is checked to determine which options are available. This can be set using the UniAdmin tool supplied on the client CD, or for those familiar with the REGEDT32 tool under Windows, by setting a DWORD value in the registry of the machine hosting UniVerse under the path:

HKEY_LOCAL_MACHINE\Software\IBM\UniVerse\CurrentVersion\UserPolicy

The following policy options can be used, depending on the balance required between security and ease of access:

**Home Directory (default)**

By default, UniVerse will permit a non-administration user to only connect to a UniVerse account located in their home directory. If their home directory does not contain a UniVerse account, they are denied access to the system. This is primarily intended for locally defined users, and is of limited use for active directory or domain users whose home directories may be on different servers.
Location specified in the UV.LOGIN file

For active directory or domain users, administrators can maintain a list of user names (including domain prefix) and the location of the account to which they should connect, in a special file in the home (uv) account name UV.LOGIN.

You can define these using the UniAdmin tool or, if you have a background in some other MultiValue platforms, by using the ED system editor.

If you do not define the home directory for a user, and you are running the default policy, any attempts to connect as a non-administration user will be effectively rejected:

Welcome to the IBM UniVerse Telnet Server.
Enter user name: demo
Enter password:

Unable to find user account information in the UV.LOGINS file
IBM UniVerse telnet session terminating...
[Connection aborted]

Any Account (registry setting 0x2)

This policy will cause the UniVerse TELNET service to prompt the user for the name of an account to which they need to connect. This must be an account registered in the UV.ACCOUNT file in the home (uv) account.

Any Directory (registry setting 0x3)

This policy will cause the UniVerse TELNET service to prompt the user for the name or directory path of an account to which they need to connect. This enables users to attach to any directory that contains a UniVerse account. This is the normal mode for administration users, but is not recommended for regular users.
Identifying UniVerse Processes (Windows)

UniVerse is a multiuser, multi-process application. Each connecting user is given a new process. You can view the running UniVerse processes using the Windows Task Manager:

![Windows Task Manager](image)

Depending on the edition of Universe and the middleware you may be running, the following UniVerse processes may be visible:

<table>
<thead>
<tr>
<th>Process</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tl_server</td>
<td>UniVerse TELNET service</td>
</tr>
<tr>
<td>unirpcd</td>
<td>UniVerse RPC daemon, used for middleware connections.</td>
</tr>
<tr>
<td>uvservice.exe</td>
<td>UniVerse service, used to maintain configuration and resources.</td>
</tr>
<tr>
<td>uvdlockd.exe</td>
<td>Deadlock daemon, used to resolve deadlocks and clean up processes.</td>
</tr>
<tr>
<td>uniapi_server</td>
<td>UniObjects API server</td>
</tr>
<tr>
<td>uvsh</td>
<td>Console or background (phantom) user.</td>
</tr>
<tr>
<td>uvapi_slave</td>
<td>Middleware background user</td>
</tr>
<tr>
<td>tl_server</td>
<td>TELNET user process.</td>
</tr>
</tbody>
</table>
IMPORTANT

You should not "End" any UniVerse process using the Windows task manager or kill a UniVerse process that is running. This may leave your database in an inconsistent state and may result in file corruption. See instructions below for safe termination.

Identifying User Processes (UniVerse)

The UniVerse shell contains a number of commands for identifying and closing user processes. UniVerse supports both foreground and background processes, the latter being initiated using the PHANTOM command:

PHANTOM some_command

Phantom processes are useful for long running or load spreading operations, and do not consume a user licence unless they involve themselves in interactive work such as socket handling.

The following commands are useful in identifying who is using the system and what they are doing, and in terminating processes that have gone awry:

LISTU or LISTUSERS

This displays a list of the UniVerse processes running:

<table>
<thead>
<tr>
<th>uid</th>
<th>User No</th>
<th>User Name</th>
<th>Terminal No</th>
<th>Login Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2580</td>
<td>brian</td>
<td>telnet:2580</td>
<td>Feb 27 09:52</td>
</tr>
<tr>
<td>0</td>
<td>2036</td>
<td>system</td>
<td>uvdlock:2036</td>
<td>Feb 27 09:49</td>
</tr>
<tr>
<td>0</td>
<td>2640</td>
<td>brian</td>
<td>uvcs:2640</td>
<td>Feb 27 11:28</td>
</tr>
<tr>
<td>0</td>
<td>2660</td>
<td>brian</td>
<td>uvcs:2660</td>
<td>Feb 27 11:28</td>
</tr>
<tr>
<td>*</td>
<td>2212</td>
<td>Administrator</td>
<td>telnet:2212</td>
<td>Feb 27 14:49</td>
</tr>
</tbody>
</table>

These are the UniVerse users presently sharing the system.

There are currently 5 users logged on the system.

PORT. STATUS

This shows detailed information on user activity including the last command run. This is normally only available to run in the uv account and requires administration privilege. PORT.STATUS is a powerful command, with advanced options enabling administrators to check on the process stack, files opened and lock history: please see the online help for full details.

PORT.STATUS requires processes to report back their activities: background processes and client/server backend processes that are busy may not be able to report their activities, and will be shown as Unavailable. The reporting places a heavy load on a system, and should only be undertaken when required.
LO (LOGOFF)

This command can be used to log off a background process or phantom. The process is identified by the "user no" column in the LISTU command above, or by the process number in the task manager. For historic reasons, this is prefixed with a minus (actually a dash):

For example:

LO -1628

MASTER OFF

This command can be used to log off a foreground (terminal) user. This is normally only available from the uv account, and may take several minutes to complete.

MASTER OFF 1628

TIP: Remember that you can use the HELP command to get more information on any of the commands in this guide.
Creating an Account

Each UniVerse system is divided into a number of separate work areas, known (for historical reasons) as Accounts. Each account may contain a separate database, or part of a database, or an application, or a customized view of the data, or any combination of the above. If you use the SQL model, each schema is held in a separate account.

When UniVerse is first installed, it contains a number of accounts:

<table>
<thead>
<tr>
<th>Account</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uv account</td>
<td>UniVerse Home Directory used for administration tasks</td>
</tr>
<tr>
<td>HS.ADMIN</td>
<td>Administration account for ODBC and related protocols.</td>
</tr>
<tr>
<td>HS.SALES</td>
<td>Demonstration account for ODBC and related protocols.</td>
</tr>
</tbody>
</table>

It is not recommended that you use any of these accounts for your own work. To explore UniVerse you should create one or more accounts. Creating new accounts does not create an overhead since UniVerse is based on separate user processes, not account based listeners.

You can create an account in a number of ways:

**Copying an existing account**

Each account consists of a number of files held in the same directory, and including certain required system files such as the VOC and VOCLIB files. You could create a new account by copying the entire content of an existing account into a new directory. This is only recommended for certain limited situations, such as installing the wychbooks demonstration database used to provide example data for the U2UG Learner Pack and Knowledge Base articles.

However there are two very important restrictions on this.

**You MUST NOT copy files that are in use, especially dynamic files (see below).**

**You MUST NOT copy ANY file that has an index.**

UniVerse files hold the absolute path to their index in the file header, so any copy of a file will use the same index as the original file. This is a pretty neat way to screw up your live application big time.

To check for the presence of an index, use the LIST.INDEX command and also check for any directories in the account prefixed with ",, for example, I_CUSTOMER.
Creating a new Account (Windows Shell)

You can create a new UniVerse account in any existing directory on the server hosting UniVerse. To do so:

- open a command shell from Windows (Start-> Run -> cmd) or by logging into Universe and issuing the DOS command.
- Once in a Windows shell, navigate to the directory in which you wish to create your account.
- type the command (changing the path if UniVerse is not loaded into its default location):

```
C:\IBM\uv\bin\mkaccount PICK
```

This creates a new, PICK flavoured account in the directory.

Account Flavor

UniVerse supports a number of different account styles or flavours, each of which contains certain emulations of older MultiValued systems. For new users, the PICK flavour is recommended because of its relative simplicity. If your background includes one of the other supported flavours, you can naturally substitute one of those from the following list:

<table>
<thead>
<tr>
<th>Account Flavor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFORMATION</td>
<td>Compatible with PRIME INFORMATION syntax.</td>
</tr>
<tr>
<td>REALITY</td>
<td>Compatible with MicroData REALITY and McDonnell Douglas syntax.</td>
</tr>
<tr>
<td>IN2</td>
<td>Compatible with IN2 (Group Intertechnique) syntax except in English.</td>
</tr>
<tr>
<td>PIOPEN</td>
<td>Compatible with PRIME INFORMATION PI/OPEN. Treat with caution.</td>
</tr>
<tr>
<td>IDEAL</td>
<td>An ugly combination that nobody except the original engineers respects.</td>
</tr>
</tbody>
</table>
Creating a New Account (Login)

If you are connecting as an administration user, the UniVerse TELNET service will prompt for the name of an account or directory to which to attach.

If you enter the name of an existing directory path that is not a UniVerse account, the service will ask whether you wish to enable it for UniVerse: type "Y" to create a new account in that directory. You are given the same choice of account flavour.

Welcome to the IBM UniVerse Telnet Server.
Enter user name: brian
Enter password: 
Account name or path\(\text{:c:\myaccounts\newaccount}\)
This directory is not set up for uniVerse.
Would you like to set it up (Y/N)?Y
0. Ideal UniVerse compatibility
1. IN2 compatibility
2. Prime Information compatibility
3. PICK compatibility
4. PI/open compatibility
5. Microdata Reality compatibility

Which way do you wish to configure your VOC ? 3
Your VOC is configured for PICK compatibility
Creating file "VOC" as Type 3, Modulo 23, Separation 4.
Creating file "D_VOC" as Type 3, Modulo 2, Separation 1.
Loading your "VOC" file. (Each "*" = 10 records.)
******************************************************************************
Loading your "D_VOC" file. (Each "*" = 10 records.)
* 
Creating file "&SAVEDLISTS&" as Type 1.
Creating file "D_&SAVEDLISTS&" as Type 3, Modulo 1, Separation 2.
Added "@ID", the default record for RetriVe, to "D_&SAVEDLISTS&".
Creating file "VOCLIB" as Type 2, Modulo 7, Separation 4.
Creating file "D_VOCLIB" as Type 3, Modulo 1, Separation 2.
Added "@ID", the default record for RetriVe, to "D_VOCLIB".
UniVerse Command Language 10.3
(c) Copyright IBM Corporation 2009. All rights reserved.
newaccount logged on: Mon Sep 21 13:09:31 2009
>

Navigating Accounts

Once an account has been created, you need a way to navigate to and from it. As you have already seen, you can specify an account name or path at the TELNET prompt if you have sufficient rights. You can also move between accounts during your command shell session. When you start to develop or deploy applications, the various middleware APIs each specify an account as a connection point.

Accounts can be located in one of two ways:

- By pathname to the directory holding the account.
- By logical account name.
Using logical account names makes it easier for users to connect and also insulates against possible future changes to your layouts e.g. if you find you need to expand your storage and move some accounts out onto different disks.

When you create, copy, import or otherwise load on accounts, UniVerse does not automatically keep track of their locations. If you want UniVerse to know where to find an account by name, rather than needing to enter the pathname each time, you can record the account in a file called the UV.ACCOUNT file. This, like most of the administration files and commands, resides in the installation uv account.

<table>
<thead>
<tr>
<th>Account...</th>
<th>UID</th>
<th>GID</th>
<th>Owner name...</th>
<th>NT pathname............</th>
</tr>
</thead>
<tbody>
<tr>
<td>UV</td>
<td></td>
<td></td>
<td>C:\IBM\UV</td>
<td></td>
</tr>
<tr>
<td>uv</td>
<td></td>
<td></td>
<td>C:\IBM\UV</td>
<td></td>
</tr>
<tr>
<td>HS.SALES</td>
<td></td>
<td></td>
<td>C:\IBM\UV\HS.SALES</td>
<td></td>
</tr>
<tr>
<td>HS.SERVICE</td>
<td></td>
<td></td>
<td>C:\IBM\UV\HS.SERVICE</td>
<td></td>
</tr>
<tr>
<td>HS.ADMIN</td>
<td></td>
<td></td>
<td>C:\IBM\UV\HS.ADMIN</td>
<td></td>
</tr>
<tr>
<td>uvdev</td>
<td></td>
<td></td>
<td>c:\work\universe\project</td>
<td></td>
</tr>
<tr>
<td>rstudio</td>
<td></td>
<td></td>
<td>c:\work\universe\project</td>
<td></td>
</tr>
<tr>
<td>wychbooks</td>
<td></td>
<td></td>
<td>c:\wychbooks</td>
<td></td>
</tr>
</tbody>
</table>

To add a new entry to the UV.ACCOUNT file you need only specify the name for the account and the account location. You can set this using a SQL INSERT command:

```
INSERT INTO UV.ACCOUNT(@ID,PATH)
VALUES('MyAccount','c:\myaccounts\myaccount');
```

Once UniVerse knows where an account resides, the account name can be used with the LOGTO command to move to that account:

```
LOGTO MyAccount
```

If you prefer not to update the UV.ACCOUNT file, you can still move between accounts using the full account path:

```
LOGTO c:\myaccounts\myaccount
```

```
>LOGTO c:\myaccounts\myaccount
>WHO
4832 myaccount From DEMO\brian
```
The VOC File

Each account is identified by a special UniVerse file named **VOC**, an abbreviation for VOCABULARY. The VOC file determines what actions you can perform in that account, and also what data is visible to you. The VOC has the same purpose as the MD (Master Dictionary) on other MVDBMS platforms, except that it is not a single level file.

The VOC file contains records that define and name the entities in an account. Each record in the VOC file is identified by a type code in the first field (column) that identifies the entity being defined. These are the most common type codes and what they define:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Data File. The record holds the path to the underlying file or directory.</td>
</tr>
<tr>
<td>Q</td>
<td>Data File. The record holds a logical pointer to a file held in another account.</td>
</tr>
<tr>
<td>V</td>
<td>Verb. The record holds the definition of a command that can be run.</td>
</tr>
<tr>
<td>S</td>
<td>Sentence. The record holds a stored sentence, a reusable command.</td>
</tr>
<tr>
<td>PA</td>
<td>Paragraph. The record holds a series of commands executed as a batch.</td>
</tr>
<tr>
<td>PQ[N]</td>
<td>PROC. The record holds an ugly series of job control commands.</td>
</tr>
</tbody>
</table>

The VOC file can be used to service a number of useful commands for seeing what information is available to you in this account. These include:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LISTF</td>
<td>Lists all the files you can access (literally all the F definitions).</td>
</tr>
<tr>
<td>LISTV</td>
<td>Lists all the verb (commands) you can run.</td>
</tr>
<tr>
<td>LISTS</td>
<td>Lists all the prestored sentences you can run.</td>
</tr>
<tr>
<td>LISTPA</td>
<td>Lists all the batch commands you can run.</td>
</tr>
</tbody>
</table>

Details on the content of the VOC file and creating stored commands and sentences can be found in the *UniVerse System Description Manual*.

Caution

Most entries in the VOC file are there for a reason. Many verb pointers, for example, are created by the system. Do not tamper with these if you do not know what you are doing: you may compromise your applications.
Working with Files

UniVerse stores its data in containers known as Files. These are similar in concept to the Tables found in other databases, though the term File in UniVerse connotes something more wide ranging. When using the SQL model, UniVerse SQL Tables are physically held as UniVerse Files.

A UniVerse file is a collection of data, which may or may not be defined using a special metadata layer known as the File Dictionary. When you create a file, you have the option to create both the file and its dictionary. You also have a number of choices about how that file will be physically structured and held in the operating system.

A file usually holds a set of related information, as in the case of a CUSTOMER file holding customer names and addresses, or a SALESORDER file holding sales orders. UniVerse programs also reside in files, either in the form of source code or as compiled object, as does anything that can be easily addressed in a UniVerse database.

For details on the UniVerse files and dictionaries, and on how UniVerse defines its content, please see the Learner Pack Guide to Writing your First UniVerse Application.

Creating a file does not specify what the content of that file will be. It only specifies how the file will be structured and, in some cases, what the initial size of the file should be for optimizing performance. As with most industrial strength database management systems, there are ways to tune the file system get the best in terms of performance and efficiency.

Files are created using the CREATE.FILE command. This, annoyingly enough, has different syntax options depending on what flavour of account you are running. In a PICK flavoured account, the command lets you define the structure and size of the file dictionary as well as the structure and size of the file data.

CREATE.FILE  filename  dict_mod,  dict_sep,  dict_type  data_mod,  data_sep,  data_type

The file parameters are divided into a modulus, separation and type. Understanding these is the most critical aspect for any UniVerse administrator, so this part will be treated in rather more detail.
Understanding UniVerse File Sizing

UniVerse is a MultiValue database (MVDBMS). This means that it adheres to the MultiValue database model, one in which complex data is stored in three or four dimensional records to better represent real world data. A Sales Order - that in a relational database would be split into an order header, order detail and an index to bind them together - can be stored as a single data record in UniVerse.

This makes for a more efficient storage model, since UniVerse can spend less time on the regular database operation of breaking apart complex entities only to join them together again when it needs to process them as a whole (which is most of the time). It simplifies data modelling, enquiry, and contention management, and can offer improved performance with lower overheads.

However, those improvements come at a price. In a relational system, it is usually fairly easy to predict the storage requirements for a single row of data. With individual detail lines being stored individually, the variation in actual storage requirements to hold each row is pretty small. You may need to hold one or many rows, but each one can be a relatively predictable size: which makes for relatively simple storage.

<table>
<thead>
<tr>
<th>Sale Id</th>
<th>Date</th>
<th>Time</th>
<th>Cust</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0001</td>
<td>01/01/09</td>
<td>12:00</td>
<td>ABC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sale Id</th>
<th>Part</th>
<th>Price</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0001</td>
<td>35</td>
<td>17.99</td>
<td>3</td>
</tr>
<tr>
<td>A0001</td>
<td>10</td>
<td>12.50</td>
<td>1</td>
</tr>
<tr>
<td>A0001</td>
<td>78</td>
<td>10.99</td>
<td>2</td>
</tr>
<tr>
<td>A0001</td>
<td>102</td>
<td>12.50</td>
<td>1</td>
</tr>
</tbody>
</table>

Relational Storage: lots of rows, but all predictably sized.

In the MVDBMS world, storage is more complex. If you take the Sales Order record holding one detail line, and put it next to a sales order record holding a hundred detail lines, that's potentially a lot of difference in size. Since these rows are not split up, it means the database needs an efficient way to hold – and quickly find again – records that can vary widely in size.

<table>
<thead>
<tr>
<th>Sale Id</th>
<th>Date</th>
<th>Time</th>
<th>Cust</th>
<th>Part</th>
<th>Price</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0001</td>
<td>01/01/09</td>
<td>12:00</td>
<td>ABC</td>
<td>35</td>
<td>17.99</td>
<td>3</td>
</tr>
<tr>
<td>A0001</td>
<td>01/01/09</td>
<td>12:00</td>
<td>ABC</td>
<td>10</td>
<td>12.50</td>
<td>1</td>
</tr>
<tr>
<td>A0001</td>
<td>01/01/09</td>
<td>12:00</td>
<td>ABC</td>
<td>78</td>
<td>10.99</td>
<td>2</td>
</tr>
<tr>
<td>A0001</td>
<td>01/01/09</td>
<td>12:00</td>
<td>ABC</td>
<td>102</td>
<td>12.50</td>
<td>1</td>
</tr>
</tbody>
</table>
MultiValued Storage: single records but difficult to predict the size.

The other big difference is how records are accessed. The relational world is optimized for query based access. MVDBMS, on the other hand, is optimized for reading and writing individual records based on the record key. An application will typically read a record, modify it, and write it back again as a single unit. Fast access to specific records is therefore the key to file performance.

Administering Files Sizing

Administering the file system is the single most important aspect of maintaining a well performing and safe UniVerse system.

The impact of file administration extends into three related areas:

- File performance
- Process Contention
- Risk

UniVerse supports a range of different file structures, designed to offer a choice to fit different application circumstances. These are:

- Static hashed files
- Dynamic hashed files
- Directory files
- B-Tree files
- Distributed files

Static Hashed Files

The majority of UniVerse data files are Static Hashed Files. These offer the best performance with the lowest overheads for both reading and writing, and the potential for the lowest contentions. However, it is important to recognize that Static Hashed Files require regular maintenance, and may not suit all data profiles.

Dynamic (Linear) Hashed Files

Dynamic or Linear Hashed Files accommodate changes automatically in such as way as to reduce, though not to totally remove, the overheads of administration. These are typically favoured for small applications where the knowledge or availability of database administration is not present, and where the individual records are relatively predictable in size. Dynamic files present higher overheads than well-maintained static files, and are not suited for situations in which large records need to be stored.
B-Tree Files

B-Tree files are used by UniVerse internally for indexed fields, and can be used programmatically as another form of data file. However, these are slow to update and should be reserved for situations in which data needs to be traversed.

Directory Files

As the name suggests, directory files are simply operating system directories. Each record in the Directory file is mapped to a separate file in the underlying operating system folder (with certain name substitutions to prevent illegal file naming). These are preferred for situations in which data needs to be shared with external processes, or for holding relatively small numbers of large records. Directory files do not suffer from record size restrictions, but do suffer from higher levels of contention.

Distributed Files

Not a specific file structure, but a Distributed File is a logical arrangement of separate UniVerse files that accommodates a single data set and that can be manipulated as if they were a single file. Distributed files are most useful where the data cannot be accommodated easily in a single file, or where subsets can be derived that can have individual meaning: for example, sales for different regions may be held as separate files but also be visible as a single logical file depending on context.

How UniVerse Data Files Work

Static files are the most common files, and best suited to storing and retrieving data. A static file is so called because the initial organization of the file is set up by the file creator, and remains static until such time as it is explicitly changed.

A static file consists of a number of separate storage sections, known as buffers. The size and number of these buffers is specified when the file is created, and space is reserved for these on disk. This is known as the 'primary space' of the file.

As records are added to the file, UniVerse distributes the records across this primary space using a scattering formula based on the record keys. This formula produces the address of the buffer in which the record should be placed. Different UniVerse file types use different calculations to give better spreads for specific arrangements of keys. Matching the right formula to the right key pattern can ensure optimum performance, but selecting the wrong pattern can cause huge system bottlenecks. If you are uncertain, stick to the default file type until you have become fully aware of the dangers.

When UniVerse needs to locate a record, by replaying this formula, it can quickly determine in exactly which buffer in the file a record should be found, irrespective of the size of the file or the number of records held. For this reason, static files provide near-instantaneous access to data, and are the most efficient - so long as the records can fit into the primary space.
File Overflow

If a buffer in a file becomes full, and another record is added to the file that is addressed to the same buffer, the file is said to be 'overflowed'.

It cannot use a different buffer to hold the record, since this would break the allocation scheme. Instead it will extend the file by chaining an additional buffer to the end of the first buffer – and so on, the more records are added to the file:
In this way, the file can grow beyond the primary space, but at the risk of degrading performance. The chain of buffers attached to a specific buffer in the primary space, is known as a 'group'. A file can grow until it hits its maximum size: this is a mere 2GB for a 32 bit file though 64 bit files are also supported.

The more buffers are chained onto the end of the primary space, the more space UniVerse must traverse in order to find a requested record, and the more buffers it must check when writing a record to try and find free space in the chain.

Since overflow buffers are added to different groups at different times, this may involve seeking forward and backward around the file content. For a large file, this means excessive disk I/O and paging activity. It also increases the risks of failure as UniVerse is spending longer performing disk operations. The longer a single file operation takes, so the greater risk of collisions on the lock table: so the bottlenecks are not just felt for the overflowed file but for all files in the system.

**Oversized Records**

A special case is that of oversized records. These are too large to fit into a single buffer, and so these are shipped into out-of-line storage. This also adds buffers to the end of the file to hold the records, but with a direct pointer to their location from the primary space so Universe does not need to hunt for the record.

Out of line storage does not slow down all read and write operations, but it has an impact on enquiry processes that have to jump around the file to read the out of line items as they are needed.

**File Resizing**

The solution to overflow problems is usually to resize the file.

Resizing a file creates a new copy of the file (it is possible to resize a UniVerse file in place, but that is NOT recommended) with a different user-specified allocation of primary space. Wherever possible when resizing a file the DBA will attempt to place all the records into the primary space. In reality unless the distribution calculation is a perfect match to the keys, this is not always possible, but the resize should bring the overflow under control.

By redistributing the records, a resize can ensure that whilst the number of records in the file remains the same, the addressing of those records can be brought back into line. Resizing a file to the correct size redistributes the records so that they can be accessed instantly once more.
Dynamic Files

Dynamic files are similar to hashed files, in that they are designed for the storage of medium sized data items. Unlike static files, they are capable of changing their primary space to improve the addressing: thus a dynamic file spreads the overheads of administration over the lifetime of the file. In order to accommodate this dynamic files are more complex internally and the automated administration offers its own, sometimes substantial, overheads.

A Dynamic File, like a static file, is divided into a number of buffers that are used to store data, using an addressing scheme based on the record key. Here, though, the allocation is performed using linear hashing and the file is physically built from two operating system files: a DATA.30 holding the primary space and an OVER.30 holding the overflow. This allows the primary space to resize itself without the overflow getting in the way.

As records are placed into a dynamic file, the file maintains an overall load counter, written to the file header and also held in shared memory for faster access. As the load reaches a defined threshold this triggers a process known as a file split, in which the primary space of the file is increased. In theory, as file splits occur the overflow data will gradually be reorganized so that eventually it all moves into the primary space once more, ready for the file to grow again as more records are added and the load increases again.

This reorganization of records takes its toll, and a process adding a record that tips the load balance to the point where the file is split, should not have to wait whilst the whole file is reorganized. So the way a dynamic file actually works is little more surprising: you do not need to read this in detail, but you may need to understand the impact.
How Dynamic Files Work

When a dynamic file is created, it contains a series of buffers. These make up the real size of the file, and records will be added to these buffers based on their keys.

As the file grows and the load increases, the point will come when the file needs to split. Here two things occur:

- The effective size of the file doubles. This does not increase the size of the file on disk, but only the size used for calculating the record positions.
- One (just one) new buffer is added to the file, and the first group in the file is split so that some of its records are written to the new buffer.

This may or may not reduce the overall load (if the first group was empty, for example). But it starts the resize process.

On the next write, the load may increase so that a new split is required. The effective size of the file remains the same, but again a buffer is added to the end and now it is the second buffer that splits.

This process repeats itself until each of the original buffers has been split once. Now the real size of the file and the effective size of the file are one and the same, since the file has really doubled in size.

Now the file is back to where it started, but with twice the volume. So the next time the load increases, the whole process begins again: the effective size of the file doubles, and the first buffer is split. Each time the process completes, the file has doubled in size.
When looking for a record, if the record address is after the last group to be split Universe knows it will still be in that group. If not, it could be in one of two places: the original buffer or the split buffer. So it may need to look in a maximum of two places to find it. In reality the ratio is about 1.2 reads for any record.

### Dynamic File Impacts

Dynamic files are a useful choice for situations in which file maintenance is impossible (24x7x365 sites) or if the expected size of a file is genuinely unknown – such as a work file used for temporary data or to hold and post-process the results of an enquiry command. However they are not a panacea for resolving file system performance: indeed, using these in the wrong situations may impact performance.

Dynamic files have more costly overheads: there is the overhead of the split operations as they grow, and the overhead of maintaining two physical files for each individual file, which may affect the rotating file pool (used to maintain the maximum number of files open) for a user process. On UNIX especially, opening files means big overheads.

When a dynamic file splits, UniVerse must also write the new sizes to the file header, which means that for safety it will perform a sync() call. This is also a heavy operation on UNIX.

To prevent all processes that are reading or scanning a dynamic file from having to constantly read the file header (to check that another process hasn't caused a split in the meantime), Universe keeps a copy of the sizing details in shared memory for each and every dynamic file open on the system. But that shared memory block is protected by a single semaphore, so that only one process can read from the memory block at one time.

Dynamic files, like static files, place large records into overflow. However, dynamic files cannot have large buffer sizes to absorb these, and the large record threshold is automatically set to 1628 bytes unless explicitly overridden when creating the file – something that a DBA or UniVerse developer will rarely remember. Consequently, large records and dynamic files do not mix.

A common mistake is to create a dynamic file without specifying an initial size. This will create a dynamic file with a single buffer, forcing splits immediately as data is added. A dynamic file should always be created with a sensible starting size.

In summary, a dynamic file will never be as efficient as a well sized static file. A dynamic file will rarely be as inefficient as a badly sized static file. Wherever possible, sites should use static files with an appropriate sizing and monitoring policy to get the best database performance.
Simple Analysis and Resizing

UniVerse has a collection of simple file analysis and resizing commands. More complex commands and tools are available from third parties that specialize in file management.

The three most required commands are:

**ANALYZE.FILE**

The ANALYZE.FILE command performs an analysis of the loading of a static file. This shows in a simple form the overall totals of data and storage space, and illustrates how well the data fits into the primary space.

```
>ANALYZE.FILE BOOK_SALES
File name       = BOOK_SALES
File type       = 18
Number of groups in file (modulo)       = 509
Separation      = 2
Number of records = 5002
Number of physical bytes              = 1010688
Number of data bytes                    = 728512

Average number of records per group     = 9.8271
Average number of bytes per group       = 1431.2613
Minimum number of records in a group    = 3
Maximum number of records in a group    = 18

Average number of bytes per record      = 145.6441
Minimum number of bytes in a record     = 72
Maximum number of bytes in a record     = 212

Average number of fields per record     = 7.0108
Minimum number of fields per record     = 6
Maximum number of fields per record     = 8

Groups   25%    50%    75%   100%   125%   150%   175%   200%  full
0       2     14     56    114    137    92     94
```

The ANALYZE.FILE command only operates with hashed files. You cannot use this on other file types.
The HASH.HELP command provides a rough estimate of an appropriate file sizing. This can usually be improved upon by knowledge and better tools, but is a good starting point. With static files, different file types may change the distribution model: only select a file type that is not the generic distribution (type 18) once you really understand what you are doing.

**HASH.HELP BOOK_SALES**

<table>
<thead>
<tr>
<th>File BOOK_SALES</th>
<th>Type= 18</th>
<th>Modulo= 509</th>
<th>Sep= 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Of the 5002 total keys in this file:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 keys were wholly numeric (digits 0 thru 9)</td>
<td>(Use File Type 2, 6, 10 or 14 for wholly numeric keys)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5002 keys were numeric with separators (as reproduced below)</td>
<td>(Use File Type 3, 7, 11 or 15 for numeric keys with separators)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 keys were from the 64-character ASCII set reproduced below</td>
<td>(Use File Type 4, 8, 12 or 16 for 64-character ASCII keys)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The keys in this file are more unique in their entirety.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The smallest modulo you should consider for this file is 873.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The smallest separation you should consider for this file is 2.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The best type to choose for this file is probably type 18.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**RESIZE**

The RESIZE command creates a new copy of the file with new sizing parameters.

```bash
>RESIZE BOOK_SALES 18 873 2
>ANALYZE.FILE BOOK_SALES
File name = BOOK_SALES
File type = 18
Number of groups in file (modulo) = 873
Separation = 2
```
UniVerse is a true multi-user, multi-process application. Many user processes may need to access the same UniVerse file at the same time. The group structure used by static and dynamic hashed files, in conjunction with the lock table, makes this possible.

The Lock table holds a number of different lock types, used to coordinate activity between different UniVerse processes. These are:

- Read and Write Locks.
- READU (Update) Locks.
- File Locks.

**Read and Write Locks**

When a process needs to read or write a record to or from a file, UniVerse first works out which group should hold that record if it is can be found in the primary space by applying the relevant hashing algorithm to the record key. This tells it which portion of the file will hold the record if it is present, or into which portion the record should be written.

Because the group structure can hold more than one record, UniVerse needs to ensure that only one process at a time can read from, or update an individual group. This prevents one process trying to scan through a group whilst another process is busy reorganizing it. So it then proceeds to place a lock entry for that group into the lock table.

Because the lock specifies both the file and the group within the file, this does not prevent another UniVerse process from accessing a different group in the same file at the same time. So different processes can still read and write a single file safely.
Group locking: user 2 is temporarily blocked from accessing the same group as user 1, but user 3 can continue to read from other sections in the file.

Lock Table and Performance

If a file is well sized and records are distributed across the primary space, it should be very rare that two processes will need access to the same part of the file at the same time, and so there will be few collisions within the group lock table.

If a file is badly sized and records are consigned to long overflow chains of buffers belonging to the same group, the chances of more than one process needing access to the same group are much higher – and so the collisions are much higher. And the longer the chains, the longer the time that each lock needs to be held.

All of which means that more processes will spend longer queuing up and having the wait for the lock to become available.

What about files that are not group based, such as directory files?

These have an even larger impact, since UniVerse has to lock the entire file every time a record is read from, or written to, the file. Effectively, UniVerse sees the whole file as a single group (group zero) and locks accordingly. For this reason, directory files should not be used to hold application data, other than in exceptional circumstances.

The lock table is a limited resource, which means that it may not have enough slots for each unique lock that needs to be taken, which will cause further contention and waiting times. Access to the lock table is controlled through a series of semaphores, each of which grants access to a set of slots: another potential bottleneck.
Whilst the lock table can be increased in size, this is treating the symptom and not the cause. In general, a lock table should only be increased where there are large numbers of concurrent users or where there are high transaction volumes.

An easy way to establish if contention is a problem is to repeatedly list the lock table (LIST.READU EVERY). If successive listings show the same read (RD) and write (WR) locks present, you have a contention issue. Read and write locks should be transient and last a few milliseconds.

**READU and File Locks**

In addition to the read and write locks, the lock table is used to hold application level locks. These grant access to a process to modify a record, and are part of the pessimistic locking model normally used for UniVerse applications.

These are not directly affected by the read and write group locks, but still require access to the lock table to check on their existence: so any bottlenecks caused by excessive read and write times will impact these checks also.

**Listing Record Locks**

The LIST.READU EVERY command displays the current state of the lock table, including both group read/write locks and record locks.

Record locks are identified by two entries: a Group Lock with a mode of IN (INFORMATION) and one or more individual record locks. The Group Lock speeds up the process of searching, since UniVerse only has to check for record locks where there is a group lock present:

<table>
<thead>
<tr>
<th>Device.... Inode..... Netnode Userno</th>
<th>Lmode</th>
<th>Group Group Group</th>
<th>G-Address.</th>
<th>Locks ...RD ...SH ...EX</th>
</tr>
</thead>
<tbody>
<tr>
<td>323281969 831484131 0 4284 89 IN</td>
<td>400</td>
<td>1 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device.... Inode..... Netnode Userno</th>
<th>Lmode</th>
<th>Pid Item-ID.............</th>
</tr>
</thead>
<tbody>
<tr>
<td>323281969 831484131 0 4284 89 RU</td>
<td>4284 13660<em>37800</em>1</td>
<td></td>
</tr>
</tbody>
</table>

If a process terminates unexpectedly, or a badly design program fills up the lock table, you may need to release locks manually. This should only be reserved for emergency situations, since releasing a group lock may lead to file corruption.

The UNLOCK command releases a range of locks, depending on the options supplied.
Printing from UniVerse

When running on UNIX based systems UniVerse is equipped with its own print spooler. On Windows, it takes direct advantage of the Windows spooler, reducing considerably the administration required on the UniVerse side to enable printing.

The primary command for printer control on UniVerse is SETPTR:

```plaintext
SETPTR fixed_options,other_options
```

This fixed_options consists of six, comma delimited entries that must be present, although the individual entries can be blank. These hold:

<table>
<thead>
<tr>
<th>Channel</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>You can define up to 10 separate printer configurations at a time, each identified by a numeric print channel. Print channels are numbered up to 256, but only ten can be active at any one time.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular line length in characters.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular page length in lines</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Top</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top margin in lines</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bottom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottom margin in lines</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print mode.</td>
<td></td>
</tr>
</tbody>
</table>

The other_options provide the control over printer output, including the name of the destination printer. These are each set by a specific keyword, and can appear in any order. The most useful of these are:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT printername</td>
<td>Sets the name or share name of the destination printer.</td>
</tr>
<tr>
<td>BANNER text</td>
<td>Sets the name of the print job as it appears in Print Manager.</td>
</tr>
<tr>
<td>COPIES n</td>
<td>Sets the number of copies to produce.</td>
</tr>
<tr>
<td>FONTNAME name</td>
<td>For GDI printing sets the font name.</td>
</tr>
<tr>
<td>FONTSIZE n</td>
<td>For GDI printing sets the font size in points.</td>
</tr>
<tr>
<td>GDI</td>
<td>Enables GDI based printing</td>
</tr>
<tr>
<td>NHEAD</td>
<td>Suppresses the initial banner page.</td>
</tr>
<tr>
<td>NOFMT</td>
<td>Enables application control over pagination.</td>
</tr>
</tbody>
</table>

For example, this sets the printing to a share named HP on a remote server called MYSERVER, using GDI mode. Notice the commas separating the initial fixed options:

```plaintext
SETPTR ,,,,,,AT \MYSERVER\HP,GDI
```

Most printing from UniVerse is text based, with some applications using mark-up like PCL (Printer Control Language) to add effects, cursor positioning and simple graphics. Where such mark-up is used, it is important that the printers remain defined as 'generic/text only' to ensure that the mark-up sequences flow directly to the printer, and are not interpreted by the Windows graphical device interface (GDI). You can normally set up more than one printer entry under Windows to access the same printer, offering a choice of text only or GDI printing.
It is possible to print to Windows printers using graphics mode, by adding the GDI flag to the SETPTR command. This may be necessary for some printers that do not understand plain text mode, or if you want to select fonts, though note that UniVerse does not naturally adjust for proportional fonts. Third party printing tools, such as mvPDF and PrintWizard, are available that offer support for fonts, barcodes and graphics.

The UVDEFAULT Printer

If a command is issued that requires printer output, such as a RetriVe listing or SQL SELECT with the LPTR keyword, this will use the currently active printer assigned using a prior SETPTR command. If no printer has been assigned, UniVerse will fall back onto a printer named UVDEFAULT.

If the UVDEFAULT printer does not exist, this will cause a fatal error that can terminate the process issuing the PRINT command, so it is a good idea to create this as a catch-all even if it should not normally be used.

The easiest option is to create a local printer under Windows named 'UVDEFAULT' and set it to print to file and paused or directed to a real printer:
Changing Terminal Definitions

In very rare cases you may need to adjust an existing emulation or to import a new emulation. UniVerse uses a terminal library named terminfo, which is a reflection of one of the two standard terminal libraries found on UNIX based systems (the other being termcap). UniVerse supplies its own terminfo library, and on UNIX systems will fall back on the UNIX terminfo for any entries that it does not have in its own library. On Windows, there is only the one, UniVerse terminfo.

Since Windows does not understand terminfo, and provides no tools for manipulating it, UniVerse comes supplied with two simple commands: uvtidc and uvtic. Both can be found in the UniVerse bin directory underneath the installation account.

The uvtidc command stands for the 'UniVerse TermInfo DeComplier'. The terminfo definitions are held in a compiled, internal format for speed: to change any of these you must first decompile the definition. Running the uvtidc command will decompile a specified terminal type to screen: more usefully you can direct the result into a text file that can be manipulated using a program like NOTEPAD:

```
uvtidc vt100 > c:\vt100.txt
NOTEPAD c:\vt100.txt
```
All of this may look like confusing garbage, but the terminfo settings are easily found on the internet. The top line holds a list of the terminal type names followed by a description. The following lines are divided into two sections: those that are generic and those that are UniVerse specific.

The UniVerse specific settings are found in the lines beginning with #!uv. On UNIX this is interpreted as a comment, so that the native terminfo compiler will ignore these lines. The UniVerse terminfo compiler, on the other hand, will compile them.

The UniVerse specific entries make up the @(-n) series in Basic (or -(n) series in PROC) such that the at4 setting means @(-4), at14 means @(-14) and so forth.

To alter a definition, change the decompiled source (take a backup first!) and then recompile using the uvtic command (UniVerse TermInfo Compiler). The –v flag (verbose) must be supplied if you want to see what it is doing:

```
uvtic -v c:\vt100.txt
```

If you are adding a new terminal type name, this needs to be added to the terminfo index. For this, there is a final command named tiindex (TermInfo Index) that rebuilds the entire index:

```
tiindex
```

Note: Any changes to your current terminal type are not visible until you log out and into UniVerse again.
Backing Up Your Applications

Taking regular backups of your applications is highly recommended, but they must be done right. Backups should be tested regularly to ensure that you can restore your systems effectively in time of need. A corrupted or incomplete backup coupled with a spurious peace of mind that all is well, is probably worse than no backup at all!

Whilst the UniVerse database is running, two things are occurring: files are changing structurally and changes are being made in memory. If you use dynamic files, in particular, taking a snapshot of your database at the operating system level using normal backup tools may not be sufficient to guarantee the resilience of your backup.

At the very least, you must ensure that UniVerse is quiescent at the point of backing up. You can use the handy SUSPEND.FILES command to pause and to resume the database: whilst the database is suspended any updates will wait. Nota that any processes that attempt to write may appear to freeze until the database is resumed.

SUSPEND.FILES ON
SUSPEND.FILES OFF

UniVerse provides its own backup and restore commands, known as uvbackup and uvrestore respectively. These create an image of a database whilst respecting the UniVerse contention management and file update procedures, so that the image should be safe to restore.

The uvbackup command can be used for full, daily or weekly incremental backups. The command is given a list of pathnames, representing the files and directories that require backing up. The following commands can be used to back up one or more accounts under the same c:\myaccounts directory:

CD c:\myaccounts
C:\ibm\uv\bin\uvbackup -f -v -nodrive -walk . > c:\mybackup.ubk

where:

- 

- Full backup i.e. not incremental
- Verbose, show each files backed up
- strip off the drive letter (useful for test restoring the backup elsewhere).
- recurse through directories
- current directory as start path

To restore your backup, you can use the companion uvrestore command:

C:\ibm\uv\bin\uvrestore -v c:\mybackup.ubk

Note that these commands can both take a wide set of options to control the scope and format of their operations. Please refer to the UniVerse Administration Guide for details on using these commands.
Finding More Information

This guide has hopefully put you on the right tracks to administering a UniVerse system. From here you may wish to learn about the regular administration commands for daily operations in more depth, to discover how to administer the various APIs, how to tune your system, and how to use the SQL model to extend the security of your system. Please note that you can download the entire UniVerse documentation set, which is available in Adobe PDF format.

The following guides are particularly useful for administrators:

- UniVerse Administration Guide
- UniVerse System Description Manual
- Administrative Supplement for Client APIs
- Using UniAdmin
- UniVerse SQL Administration for DBAs
- UniVerse Transaction Logging and Recovery
- UniVerse Security Features

You can find more information relating to UniVerse on the U2 User Group website.

Here you will find technical and welcome forums, learner pack and knowledge base, a wiki and a thriving email list community ready to answer your questions. You can also find details of local MultiValue user groups.

Please visit http://www.u2ug.org.