



Rocket UniVerse

Guide for Pick Users

Version 11.2.3

April 2014

UNV-1123-PICK-1

Notices

Edition

Publication date: April 2014

Book number: UNV-1123-PICK-1

Product version: Rocket UniVerse V11.2.3

Copyright

© Rocket Software, Inc. or its affiliates 1985-2014. All Rights Reserved.

Trademarks

Rocket is a registered trademark of Rocket Software, Inc. For a list of Rocket registered trademarks go to: www.rocketsoftware.com/about/legal. All other products or services mentioned in this document may be covered by the trademarks, service marks, or product names of their respective owners.

Examples

This information might contain examples of data and reports. The examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

License agreement

This software and the associated documentation are proprietary and confidential to Rocket Software, Inc., are furnished under license, and may be used and copied only in accordance with the terms of such license.

Note: This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when exporting this product.

Contact information

Website: www.rocketsoftware.com

Rocket Software, Inc. Headquarters

77 4th Avenue, Suite 100

Waltham, MA 02451-1468

USA

Tel: +1 781 577 4321

Fax: +1 617 630 7100

Contacting Global Technical Support

If you have current support and maintenance agreements with Rocket Software, you can access the Rocket Customer Portal to report and track a problem, to submit an enhancement request or question, or to find answers in the U2 Knowledgebase. The Rocket Customer Portal is the primary method of obtaining support.

To log in to the Rocket Customer Portal, go to:

www.rocketsoftware.com/support

If you do not already have a Rocket Customer Portal account, you can request one by clicking **Need an account?** on the Rocket Customer Portal login page.

Alternatively, you can contact Global Technical Support by email or by telephone:

Email: u2support@rocketsoftware.com

Telephone:

North America	+1 800 729 3553
United Kingdom/France	+44 (0) 800 773 771 or +44 (0) 20 8867 3691
Europe/Africa	+44 (0) 20 8867 3692
Australia	+1 800 707 703 or +61 (0) 29412 5450
New Zealand	+0800 505 515

Table of Contents

Chapter 1

Chapter 1: Getting started

The UniVerse environment	1-3
UniVerse command processor (TCL)	1-4
The VOC file (master dictionary)	1-4
RETRIEVE (ACCESS)	1-5
BASIC programming language	1-5
REVISE	1-5
Editors.	1-5
PROVERB (PROC)	1-6
Menus	1-6
Relationship of UNIX and UniVerse	1-7
The UniVerse file system	1-8
UniVerse files	1-8
Data files and file dictionaries.	1-9
Differences between UniVerse and Pick	1-10

Chapter 2

Chapter 2: Understanding UniVerse accounts

UNIX vs. UniVerse accounts	2-3
UNIX file and directory structure	2-5
A UNIX login account	2-7
The standard UNIX shell environment.	2-7
A UniVerse account	2-10
UniVerse account flavors	2-10
Creating or updating an account	2-12
VOC File templates (NEWACC files)	2-12
Converting to a UniVerse account	2-14
Convert master dictionary items to entries in a UniVerse VOC file	2-14
Convert Pick file dictionaries to UniVerse file dictionaries	2-16
Converting REALITY procs	2-16
Converting dictionaries with the DC command	2-17

Converting UniVerse BASIC program files	2-18
Recompiling and cataloging BASIC programs	2-19
Where to go from here	2-20

Chapter 3

Chapter 3: The VOC file

The UniVerse command processor (TCL)	3-3
Special character interpretation	3-3
PHANTOM processes	3-4
The VOC file (master dictionary)	3-6
VOC file record format	3-6
VOC file entry types	3-9
The VOCLIB file	3-16
Listing VOC file entries	3-16
The command processor sentence stack	3-17

Chapter 4

Chapter 4: UniVerse file structure

The UniVerse file system	4-3
Hashed and nonhashed files	4-5
Records in a hashed file	4-5
Static hashed file structure	4-7
Dynamic hashed file structure	4-8
Nonhashed file structure	4-10
B-Tree files and secondary indices	4-12
Secondary indices	4-13
Multiple data files	4-14
File pointers	4-16
SETFILE command	4-16
The SET.FILE command	4-17
File protection	4-19
Explicit locks.	4-19
Implicit locks	4-20
UNIX file protection	4-21
Maintaining files	4-24
GROUP.STAT and HASH.TEST commands	4-24
More file statistics	4-25
Maintaining dynamic files	4-26
Resizing files.	4-26
Cleaning up an account	4-28

Chapter 5

Chapter 5: UniVerse file dictionaries

UniVerse dictionary entries	5-3
UniVerse dictionary fields.	5-4
Pick attribute conversion.	5-7
More about dictionary records.	5-9
I-descriptors	5-11
Defining I-descriptors	5-11
I-descriptor expressions	5-12
Compound expressions	5-15
Correlative and conversion codes.	5-17
Multivalued data conversions	5-17
Phrases.	5-21
Special phrases	5-21

Chapter 6

Chapter 6: Retrieve

An overview of Retrieve verbs.	6-3
Retrieve syntax	6-6
Specifying records	6-7
Using the WITHIN keyword.	6-7
Selection expressions	6-8
Print limiting	6-10
Report and output keywords	6-11
Parenthetical options	6-12
REFORMAT and SREFORMAT	6-14
Creating and processing select lists	6-15
NSELECT	6-15
Using active select lists.	6-16
Retrieve command parsing	6-18

Chapter 7

Chapter 7: Printer, terminal, and tape commands

Using logical print channels	7-3
Specifying printing forms.	7-4
Printers and the UniVerse spooler	7-5
Printer configuration and status	7-5
Directing spooler output	7-6
Checking spool queues.	7-8
Managing print jobs.	7-8
Managing hold files.	7-9
Spooling print jobs to and from tape	7-10
Terminals	7-11

Magnetic tapes	7-13
--------------------------	------

Chapter 8

Chapter 8: UniVerse BASIC

The UniVerse BASIC command	8-3
Options to the UniVerse BASIC command	8-3
Compiler directives	8-6
Successful compilation	8-12
The RUN command	8-14
Debugging your UniVerse BASIC program	8-16
Cataloging a UniVerse BASIC program	8-17
Redimensionable arrays	8-18
Pick-style COMMON.	8-18
Pick-style DIM statement	8-19
Executing UniVerse commands	8-20
Sequential I/O access	8-21
Vector functions	8-28
The MATPARSE and MATBUILD statements	8-31
Using MATPARSE.	8-31
Using MATBUILD.	8-32
REMOVE statement	8-33

Chapter 9

Chapter 9: Application development tools

The ProVerb processor (PROC)	9-3
How ProVerb works	9-3
Using sentences and paragraphs	9-5
Using menus	9-8

Appendix A

Appendix A: User exit codes

Writing user exits.	A-2
Writing user exits to be called from UniVerse BASIC programs	A-2
Writing user exits to be called from procs	A-3
Cataloging user exits	A-6
Cataloging programs in PICK accounts	A-6
Executing user exits from UniVerse BASIC programs	A-7
Executing user exits from procs	A-8
User exits called from ProVerb	A-9
User exits called from UniVerse BASIC programs	A-10
User exits called from dictionaries	A-10
Alphanumeric list of user exits	A-11
U0190	A-11
U0192	A-12

U0196.	A-13
U01A2	A-13
U01A6	A-14
U01AD	A-14
U01BE	A-15
U029E	A-15
U035A	A-15
U1114.	A-16
U11A2	A-16
U11BE	A-16
U201E	A-17
U20E0	A-17
U2196.	A-17
U21A2	A-18
U307A	A-18
U30E0	A-18
U31AD	A-19
U407A	A-19
U41AD	A-20
U508E	A-20
U50BB	A-20
U5114.	A-21
U60E0	A-21
U61A2	A-22
U70E0	A-22
U7201.	A-22
U80E0	A-22
U81F5	A-23
UA1A2	A-23

Appendix B

Appendix B: Flavor-dependent commands

ACCOUNT.RESTORE	B-3
CHECK.SUM	B-8
CONVERT.ACCOUNT	B-9
CONVERT.VOC.	B-11
COPY	B-13
COPY.LIST	B-17
CREATE.BFILE	B-19
CREATE.FILE	B-21
DC	B-22

DECATALOG	B-25
GET.LIST	B-27
LOGOFF.	B-29
PRINT.ERR.	B-30
QSELECT	B-31
REFORMAT	B-32
RUN	B-33
RAID	B-34
SELECT	B-35
SET.FILE	B-36
SREFORMAT	B-37
SSELECT	B-37

Chapter 1: Getting started

The UniVerse environment	1-3
UniVerse command processor (TCL)	1-4
The VOC file (master dictionary)	1-4
RETRIEVE (ACCESS)	1-5
BASIC programming language	1-5
REVISE	1-5
Editors	1-5
PROVERB (PROC)	1-6
Menus	1-6
Relationship of UNIX and UniVerse.	1-7
The UniVerse file system	1-8
UniVerse files	1-8
Data files and file dictionaries	1-9
Differences between UniVerse and Pick	1-10

UniVerse has all the features familiar to users of other Pick systems, including

- variable structure files
- dictionaries that define data stored in separate data files
- a BASIC compiler
- a run machine

This chapter provides a brief overview of UniVerse processors and utilities, noting differences and describing unfamiliar components of the system. Much of this material is explained in greater detail in *UniVerse System Description*.

The UniVerse environment

UniVerse allows database designs that reflect the application environment without awkward system design and excessive system overhead. UniVerse uses UNIX and Windows directories and files. You can access files throughout the system in any UNIX or Windows directory. UniVerse provides you with a familiar interface while taking advantage of the operating system. Many of the processors and utilities function no different from those on other Pick systems. However UniVerse contains some processors and utilities that may be new to you.

The UniVerse Command Processor, like the Pick Terminal Control Language (TCL), interprets command lines, performs certain substitutions on the command lines, and passes control to the proper process or utility, as does the UNIX or Windows shell. Other UniVerse processors, such as the Editor, PROVERB (the UniVerse equivalent of PROC), and REVISE (a data entry processor), offer a set of commands tailored to their specific functions, as do UNIX and Windows processors such as the full screen editor and desk calculator processes. UniVerse also supports a procedural language, UniVerse BASIC, that allows you to write a program, compile it, and then execute it.

UniVerse comprises the following:

- The UniVerse Command Processor (TCL)
- The PROVERB (PROC) processor
- RETRIEVE, a database query processor and report writer (ACCESS, and so forth)
- UniVerse BASIC, a procedural language processor
- The BASIC Run Machine, which executes compiled BASIC programs
- The UniVerse File Handler, which manages the UniVerse file systems as a collection of UNIX files and directories
- A set of utilities and processes that the language processors and the Run Machine call on to do the tasks
- The UniVerse Editor
- The REVISE data entry processor

Most of these processors and utilities are described in detail in the *UniVerse System Description*. The BASIC language processor and run machine are described in *UniVerse BASIC*.

UniVerse command processor (TCL)

The UniVerse Command Processor performs the same function as the Terminal Control Language (TCL) in other Pick systems. The UniVerse Command Processor interprets commands entered at the system prompt by the user or directed to it from other processors, such as PROVERB or a BASIC program. The Command Processor either executes the command itself or passes the command to another UniVerse, UNIX, or Windows process. It also includes facilities for logging on and logging out of a UniVerse account.

When you enter the UniVerse environment, the UniVerse Command Processor displays the UniVerse system prompt (>). At this prompt you can enter any UniVerse command.

A UniVerse command line is called a *sentence*. A sentence always begins with a UniVerse *verb* specifying the operation or process to be executed. A verb is often followed by a file name and one or more phrases that modify processing. A *keyword* is part of a sentence that modifies the action of the verb.

An added capability of the UniVerse Command Processor is a list of previous commands called the *sentence stack*. The sentence stack lets you recall, change, and re-execute up to ninety-nine previous commands without retyping. You can save any sentence from the sentence stack for use in a future UniVerse session. You can also save two or more sentences as a *paragraph*.

The VOC file (master dictionary)

The VOC file performs the same functions as the Master Dictionary in other Pick systems. The VOC (or vocabulary) file contains records (items) that identify each verb, sentence, paragraph, file, keyword, phrase, and menu that are available in a UniVerse account. The Command Processor uses the VOC file to decide what action to take when you enter a command. The first field (attribute) in a VOC file record contains a code that defines the type of record. It is similar to the D/CODE.

When you convert an existing Pick account to UniVerse, a conversion program changes items in the Master Dictionary into entries in the VOC file. The account conversion process is described in Chapter 2, [“Chapter 2: Understanding UniVerse accounts.”](#)

RETRIEVE (ACCESS)

RETRIEVE is a dictionary-driven database manager that has the same function as ACCESS, INFO/ACCESS, ENGLISH, RECALL, and INFORM on other Pick systems. Nearly all the verbs and connectives that are familiar to users in other query languages are available in UniVerse.

BASIC programming language

The UniVerse BASIC programming language is a much enhanced version of Pick BASIC. Some of the features that may be new to you are:

- Redimensionable arrays
- Sequential file processing
- List logic functions
- The MATPARSE and MATBUILD statements
- The REMOVE statement

For more information about BASIC statements and functions, see *UniVerse BASIC*.

REVISE

REVISE is analogous to the update processors found on some of the enhanced versions of R83 and on Advanced PICK. REVISE is a dictionary-driven data entry processor. You can use it to add, change, and delete data in UniVerse files. When you invoke REVISE on a data file, it uses field definition items in the file dictionary to provide prompts that are specific to the data file. They can be used to determine input format conversion and input validation. When you invoke REVISE on a file dictionary, the dictionary DICT.DICT, located in the UV account, provides the prompts.

Editors

The UniVerse Editor is a line-oriented editor for adding, changing, and deleting records in a UniVerse file. It can also be used to create or modify VOC file entries, BASIC programs, and procs.

You can also use any UNIX or Windows editor, such as *vi*, to edit records in nonhashed files. In the UniVerse environment you can invoke *vi* in two ways. The UniVerse VI command invokes *vi* for editing nonhashed UniVerse, UNIX, and Windows files. The UV.VI command invokes *vi* for editing records in hashed files.

PROVERB (PROC)

PROVERB is the UniVerse equivalent of PROC on other Pick systems. PROVERB is a command language for defining a procedure, or *proc*. UniVerse PROVERB supports both PQ and PQN procs.

Menus

The menu processor may be new to many Pick users. You can use the UniVerse Menu processor to do many things that can also be done by procs. The Menu processor defines a sequence of commands for execution from menus. Control transfers and prompting for terminal input can be part of menus and menus are a way of providing built-in help. Menus thus make it easy to set up simple applications.

Relationship of UNIX and UniVerse

UniVerse is a group of programs that run under UNIX, but to an end user, UNIX may be invisible. In fact, to some end users, UniVerse *is* the operating environment.

Like Pick, UniVerse has its own Command Processor, with a command vocabulary that includes many UNIX commands as well as a large number of data management commands that cannot be accessed from UNIX. In this sense it is most analogous to the UNIX shell programs *sh* and *csh*. UniVerse also has its own log on procedure and account structure.

It is possible to do almost everything in the UniVerse environment. Because the UNIX operating system forms the basis for the system software, a good understanding of UNIX enhances your effectiveness in UniVerse. We recommend that you become familiar with the tools available in both environments and select the tool appropriate to the job at hand.

Almost all UniVerse processes that perform input or output operations to the terminal make use of an enhanced version of the UNIX *terminfo* facility. This facility allows programmers to construct applications which are hardware-independent: terminal I/O need not be dependent on terminal-specific control sequences. See *UniVerse System Administration* for more details about the relationship between UniVerse and UNIX.

The UniVerse file system

The UniVerse file system differs from the four-level hierarchical Pick file system. This is because the UniVerse file system is implemented using UNIX directories and files. From the user's point of view these differences may not be immediately apparent, but the internal structures are very different.

In UniVerse there is no equivalent to the SYSTEM Dictionary on other Pick systems. Nor is there a single level of the file system on which the UniVerse equivalent of Pick's Master Dictionaries are located: UniVerse VOC files can be located in any UNIX or Windows directory at any level on the system. And in any UniVerse account, the VOC file, all file dictionaries, and all data files are stored at the same level—that is, in the UNIX or Windows directory where the UniVerse account resides. See Chapter 4, "[Chapter 4: UniVerse file structure](#)," for a more extended discussion of the UniVerse file system.

UniVerse files

UniVerse provides several kinds of file organization:

- Nonhashed files
- Hashed files
- B-Tree files
- Dynamic files

Nonhashed files are used to store text, program source code, or other data which is not structured like a Pick data file. Nonhashed data files are implemented as UNIX or Windows directories in which each record is stored as a separate UNIX or Windows file.

Hashed files use a hashing method to distribute records in one or more group buffers on disk. Just as on other Pick systems, a hashing algorithm is applied to the record ID to generate the address of a group buffer where the record is stored. UniVerse provides 17 different hashing algorithms for use, including the one used on most Pick systems.

UniVerse features b-tree files which store records in sorted order. To find a record, its value is compared to the value at the center of the tree. If it is greater, the search continues with the subtree to the left of the center element, if it is less, the search continues with the subtree to the right of the center element. This process continues until the record is found. Thus only a few records need be searched to find any record. Balanced tree files are the most efficient files to use when frequent searches are made for record IDs that are only partially specified. To find such records in a hashed file involves reading the entire file.

UniVerse dynamic files are hashed files that resize themselves automatically as they grow or shrink. A well sized, static hashed file will perform better than a dynamic file. But files of constantly changing size, may perform better as dynamic files.

Data files and file dictionaries

As on all Pick systems, every UniVerse file comprises at least two files: a data file and an associated file dictionary. The data file comprises records (items) that contain data values in fields (attributes). The file dictionary contains records that define the contents of the data file as well as how the data is to be processed and displayed. Most UniVerse files have both a dictionary and at least one data file, although a file dictionary can be created that has no data files associated with it. This relationship is defined by an entry in the VOC file that defines the UniVerse file. Unlike other Pick systems, a data file can be created that is not associated with any file dictionary. A data file can be either nonhashed or hashed, depending on the type of data you want to store in it. All file dictionaries are usually hashed.

Differences between UniVerse and Pick

You can easily convert existing Pick applications to run in UniVerse, and you can develop new UniVerse applications using all the concepts, tools, and languages familiar to a Pick programmer. There are few major differences in functionality. The main unsupported areas are as follows:

- Assembly language subroutines
- SYSTEM Dictionary
- SYSPROG account
- ACC account containing ACC files
- POINTER-FILE, either at the system level or in local accounts
- RUNOFF processor
- The Pick four-level file hierarchy

There are some differences in terminology, as summarized in the following table.

UniVerse Term	Pick Equivalent
Record	Item
Field	Attribute
Record ID	Item ID
VOC file	Master Dictionary
Keyword	Connective, Modifier

UniVerse vs. PICK Terminology

Chapter 2: Understanding UniVerse accounts

UNIX vs. UniVerse accounts	2-3
UNIX file and directory structure	2-5
A UNIX login account	2-7
The standard UNIX shell environment	2-7
A UniVerse account	2-10
UniVerse account flavors	2-10
Creating or updating an account	2-12
VOC File templates (NEWACC files)	2-12
Converting to a UniVerse account	2-14
Convert master dictionary items to entries in a UniVerse VOC file	2-14
Convert Pick file dictionaries to UniVerse file dictionaries. . .	2-16
Converting REALITY procs	2-16
Converting dictionaries with the DC command	2-17
Converting UniVerse BASIC program files	2-18
Recompiling and cataloging BASIC programs.	2-19
Where to go from here	2-20

This chapter describes the differences between UNIX login accounts and UniVerse accounts. It provides an overview of the UNIX file structure, including a section on how to set up a UniVerse account. The last part of the chapter describes how accounts transferred from other Pick systems are converted to UniVerse accounts.

UniVerse users can work in one of two environments: the UNIX programming environment or the UniVerse database management environment. The difference between the two environments has some implications for how user accounts are assigned.

UNIX vs. UniVerse accounts

A UNIX login account and a UniVerse account are different. Login accounts are really more like personal working environments that stay with users no matter what else they may be doing or where they may be working on the system. A login account is defined by a line in the UNIX */etc/passwd* file that defines the user's login name, password, and home directory. Once UNIX users log on to the system, they have access to all directories and files on the system, except those protected by file permissions.

UniVerse accounts, like Pick accounts generally, are more self-contained. The user's working environment in UniVerse is determined primarily by the UniVerse account in which the user is currently working. It is not determined, as it is in UNIX, by the user's login account.

In this manual, whenever the context might be unclear, we will refer to UNIX login accounts as *user accounts* or *login accounts*; we will refer to accounts in the UniVerse environment—which are equivalent to accounts on other Pick systems—as *UniVerse accounts*.

In the UNIX environment, each user is generally given an individual login name, often called a UNIX account, which includes a *home directory* under which the user can create his or her own directory tree of private files. Access to other parts of the system is easy. Users can change their current working directories without changing other aspects of their working environment. Files and commands in other directories can be accessed simply by typing the path that identifies the proper location in the file system's complete directory tree.

In the UniVerse environment, on the other hand, as on other Pick systems, the UniVerse account a user is logged on to and the user's working environment are more or less identical. The VOC file contained in each UniVerse account defines the account environment, including all the files and all the commands that are available to users logged on to the account.

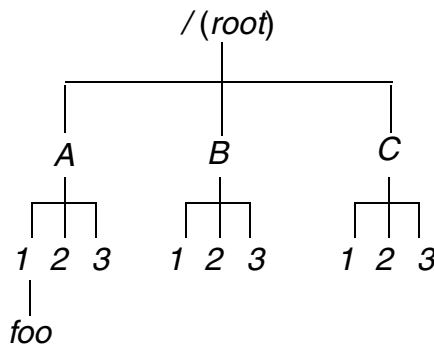
Some sites make both environments available to users and create a separate UniVerse account for each user. At other sites, users may work primarily in the UniVerse environment, and access to data may be limited only to members of a selected group working in one particular UniVerse account. For example, all users of a particular UniVerse BASIC application might share the same UniVerse account, since they all need access to the same commands, data files, and file dictionaries.

A UniVerse account might therefore be defined for SALES rather than for an individual user. Each user of the SALES account might be given his or her own login name at the UNIX level but be assigned the same home directory and, therefore, share the same UniVerse account.

UNIX file and directory structure

The UNIX file system is a hierarchical file structure made up of directory trees. A number of files can be organized together and stored in a special type of file called a *directory*. Directories containing files can be included in a group of other directories and files, which are stored in another directory. A directory that contains other directories and files is called the *parent directory* of the files and subdirectories it contains. Any directory can have its own set of subdirectories and files created below (or within) it. Every file in a UNIX system must be part of this directory tree. The parent directory of the entire system is called the *root* directory and is represented by the character “/”.

You can gain access to any file in the directory tree by specifying the *path* to that file from the root directory. Each directory along the path must be specified in order, separated by the “/” character. If the root directory has three subdirectories, A, B, and C, and each of those three directories in turn has three of its own subdirectories, 1, 2, and 3, the path /A/1/foo refers to the file named “foo” in the directory “1” which is itself in the directory “A” which is in the root directory.



UNIX File and Directory Structure

The root directory comprises a number of subdirectories containing system software. Additional file systems are “mounted” on the root and, once mounted, are available to users as if they were part of a single file system. The names and sizes of the additional file systems on your installation can be displayed using the STATUS DISK command. These additional file systems contains the following:

- Additional system programs and data files.
- UniVerse system programs and files.
- UniVerse accounts.

A UNIX login account

When you add a new user to the UniVerse system, the user is provided with a login name, a password, and a *home directory*. For example, a user named John might have a home directory in */u2* called */u2/john*. (The actual names of the user file systems are system-specific. */usr* and */user* are two more examples.)

A UNIX login account really only gives a user access to the UNIX system and a place to store files. The login account does not limit access to other directories; in fact, it is often necessary to gain access to information stored in other directories, such as the UNIX commands themselves.

The standard UNIX shell environment

The standard UNIX environment provides you with a *shell*, or command interpreter, at login time. The shell interprets commands you type by translating them into a form that can be understood by the computer. Three types of shell are provided on most UNIX systems:

- Bourne shell (*sh*)
- Berkeley C-shell (*csh*)
- Korn K-shell (*ksh*)

Although UniVerse is a very large program, it can be thought of as a UNIX shell. It accepts input from you and performs operations based on that input. Like a UNIX shell, UniVerse has a number of commands which it understands.

Your System Administrator can set up your login account so that you enter a UNIX shell at login time, or your account may be set up so that you log directly on to the UniVerse environment. When you log out of your initial login shell, you exit the system entirely. This means that if you log directly on to UniVerse, you may never interact directly with the underlying UNIX environment.



You access a UNIX shell from UniVerse using the commands SH and CSH. Use the UNIX command *uv* to enter UniVerse from a UNIX shell. When you use the *uv* command while working in a directory that already has a UniVerse account, you enter the UniVerse environment. If the current working directory is not set up for a UniVerse account, you are prompted to set up a new UniVerse account in that directory. If you exit a shell that you entered from another shell, you exit to the calling shell.

***Note:** If you invoke a UNIX shell from the UniVerse environment with the SH or CSH commands, be careful not to return to UniVerse using the UNIX *uv* command. Instead, use either the exit command or CTRL-D.*

Shell environment initialization

In general, whenever a shell is initialized, it executes a sequence of commands stored in a special file in your home directory. The commands in this file can define a set of characteristics that modify the default UNIX or UniVerse environment for a specific account.

The System Administrator usually creates an initial version of this file when an account is set up. Users can modify the contents of this file, although we recommend that they first become familiar with how these environment characteristics are specified.

For example, the special file usually contains commands to set a search path that is used to locate system commands, to set up the default protection that is given to files the user creates, and to define the characteristics of the terminal used to log on to the system. This file can also be used to specify the prompt that is displayed on the terminal screen in the environment.

In the Bourne shell, the special file called *.profile* is read and executed whenever a Bourne shell (*sh*) is executed as the login process.

In the Berkeley *cs*h (referred to as the C-shell) environment, the *.login* file is executed whenever a C-shell (*cs*h) is executed as the login process. The *.cshrc* file is executed whenever a new C-shell is created.

In the UniVerse environment, the System Administrator can set system-wide defaults by placing the appropriate commands in a paragraph named UV.LOGIN in the VOC file of the UV account. The UV.LOGIN paragraph is like a system-wide login proc: it is executed whenever any user logs on to any UniVerse account. You can also create a login entry in the VOC file of any UniVerse account. The login entry can be a paragraph, sentence, proc, menu, or BASIC program; it performs the same function in a UniVerse account as the Pick logon proc does in an account on a Pick system. The login entry in a UniVerse account is executed after the system-wide UV.LOGIN paragraph is executed.

The record ID of a UniVerse account's login entry depends on the flavor of the account. In PICK, IN2, and REALITY flavor accounts, the login entry can be one of the following:

- the name of the UniVerse account
- the name of the user's UNIX login account
- the name LOGIN

The UniVerse account name is read from the UV.ACCOUNT file in the UV account. The UNIX login name is read from the */etc/passwd* file. The system looks for a login entry in the order shown.

In IDEAL and INFORMATION flavor accounts, a login entry with the same name as the account name or the UNIX login name is ignored; login entries in these accounts must be called LOGIN.

A UniVerse account

UniVerse accounts are different from Pick in many ways. UniVerse uses the UNIX login utility. Depending on how the System Administrator sets up your login account, you may log on first to the UNIX environment and then type **uv** to enter a UniVerse account, or you may log on directly to UniVerse.

Unlike Pick, which allocates a certain section of the system for each user, a UniVerse account is defined by a VOC file and by an entry in the UV.ACCOUNT file in the System Administrator's account (the UV account). Although the UV.ACCOUNT file is in some ways analogous to the SYSTEM Dictionary on other Pick systems, it should not be considered identical to it. The UV.ACCOUNT file is not a dictionary in the sense that the SYSTEM file is, nor is it located at the top of the file system hierarchy. It does define the physical location of an account by listing the name of every account on the system along with the full UNIX path of the directory where the account is located. Under most circumstances, a UniVerse account can be referenced both by its name and by the path of the UNIX directory containing the account's VOC file.

The VOC file for an account contains entries (items) defining all the files, commands, keywords, and so forth, that are available to users while they are working in the account. As on other Pick systems, you can set up pointers in the VOC file that reference files in other accounts on the system.

UniVerse accounts can also be defined by function. That is, it may be desirable for all users of a particular application, such as members of accounting or personnel departments, to share an account, since all members of the same department need to use the same set of data files. For example, all members of an accounting department might have the same home directory, `/u2/ACCTG`.

UniVerse account flavors

You can configure your UniVerse account to be compatible with one of the various implementations of the Pick system, such as REALITY or Prime INFORMATION. These different types of account are called UniVerse *flavors*.

The differences between an IDEAL uniVerse account and a PICK, IN2, or REALITY flavor account are minor. The PICK, REALITY, and IN2 flavors accept variations of syntax that are different from the syntax used for the same commands, statements, and functions in an IDEAL UniVerse account. You may not need to be aware of the differences, except to know that applications you want to run in a particular type of Pick environment should be developed in an account whose flavor is compatible with that environment.

Any UniVerse account can be one of the following standard flavors:

- IDEAL UniVerse
- Prime INFORMATION
- PICK
- REALITY
- IN2

The IDEAL flavor represents the native facilities of UniVerse. These facilities are also available in any of the compatibility flavors. The Prime INFORMATION flavor is used to maintain an environment compatible with Prime INFORMATION. The PICK, IN2, and REALITY flavors are compatible with different versions of the Pick system.

Creating or updating an account

The System Administrator creates UniVerse accounts. However, if you use the UNIX *uv* command from a directory not set up for UniVerse, you are notified that the directory is not set up for UniVerse, and then prompted to set up a new UniVerse account. If your System Administrator has not restricted you to a particular flavor, you are also prompted to choose an account flavor. When an application is converted to run in UniVerse, the account conversion process also creates accounts of the appropriate flavor for the application being converted.

The RELLEVEL entry in your VOC file contains the current release level of your account. Each time you log on to the account, this entry is checked to make sure your account is up to date. If your account is not current, you are prompted to update it. To display the RELLEVEL entry, enter the command “.L RELLEVEL” at the TCL prompt.

To update your account to the current release level or to change the flavor of your account, use the UPDATE.ACCOUNT command. While your VOC is being updated, records which are being replaced are moved to the file &TEMP& to prevent them from being destroyed. Any records which are moved to &TEMP& are listed on your screen. See *UniVerse System Description* for information about the &TEMP& file.

VOC File templates (NEWACC files)

The UV account contains a set of templates for VOC files for each account flavor. These templates, called NEWACC files, are stored as multiple data files under the name NEWACC, with a separate data file for each flavor. The name of each data file corresponds to the flavor, as follows:

File Name	Flavor
NEWACC	IDEAL uniVerse
PICK	PICK
NEWACC Files	

File Name	Flavor
REALITY	REALITY
IN2	IN2
INFORMATION	Prime INFORMATION

NEWACC Files

Each NEWACC file is a fully configured VOC template. NEWACC files are analogous to the "NEWAC" file in the SYSPROG account on other Pick systems (note that it is NEWACC on UniVerse, not NEWAC).

When your account is set up, a remote file pointer to the template for your accounts flavor, the NEWACC file, is included in your VOC file. You can gain access to the other NEWACC files from your account by creating a Q-pointer to the multiple data file NEWACC. To create a Q-pointer called UV.NEWACC in the VOC file that points to the NEWACC file in the UV account, type:

```
>SET.FILE UV NEWACC UV.NEWACC
```

To list the contents of one of the NEWACC files, use standard multiple data file syntax. For example:

```
>LIST UV.NEWACC,PICK
```

or:

```
>ED UV.NEWACC,NEWACC record.ID
```

Alternatively, any single NEWACC file can be defined as a single-level data file by using its full path in field 2 of the file pointer. This is how the NEWACC file is defined in standard uniVerse accounts. For example:

```
NEWACC
0001 F
0002 /u1/uv/NEWACC/PICK
0003 /u1/uv/D_NEWACC
```


Converting to a UniVerse account

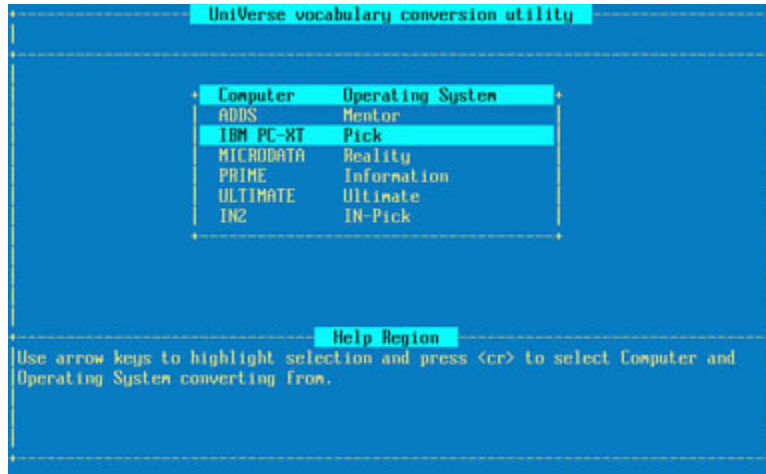
Once a Pick account has been transferred to a UniVerse system, it must be converted from its original Pick format to a format compatible with UniVerse. Use the CONVERT.ACCOUNT command to access the Account Conversion menu. This menu provides two submenus, one for converting dictionaries, the other for converting BASIC programs.

Convert master dictionary items to entries in a UniVerse VOC file

The CONVERT.VOC program converts Master Dictionary items into equivalent entries in the uniVerse VOC file. The Pick Master Dictionary is neither changed nor deleted by the program.

To convert the master dictionary, do the following:

1. Type **CONVERT.ACCOUNT** from TCL.
2. Select **convert md or Voc** from the **convert Dictionary** menu. The following menu appears:



Computer	Operating System
ADDS	Mentor
IBM PC-XT	Pick
MICRODATA	Reality
PRIME	Information
ULTIMATE	Ultimate
IN2	IN-Pick

Help Region
Use arrow keys to highlight selection and press <cr> to select Computer and Operating System converting from.

3. Select the appropriate computer and operating system from the menu. UniVerse prompts whether an error report should be printed that identifies any items that could not be converted and if it should include all conversions.
4. Respond to the prompts to specify the type of reporting you want.

File definition entries in UniVerse are different from their counterparts on a Pick system. The modulo and separation are specified in the file itself and are not part of the entry. When the CONVERT.VOC program converts the files, it allocates space on disk according to the modulo and separation specified by the Pick file definition item. These files are assigned a default File Type of 18. (In some cases, program files are created as type 1 files.) If you need to improve efficiency in UniVerse, you can change the modulo, separation, and file type using the RESIZE command.

CONVERT.VOC converts Pick file definition items (with a D/CODE of "D") to UniVerse file definition entries (with an F1 of "F"). Q-pointers are implemented as they are in Pick. The program copies proc entries from the Pick Master Dictionary to the UniVerse VOC file without conversion. If any unconverted verbs or connectives are used in the proc, you must remove them or specify an equivalent for them to work properly. REALITY users should see the section "Converting REALITY Procs" for special information about converting REALITY procs to UniVerse procs.

Pick Master Dictionary items are passed to the DC utility, and the converted dictionary items are written to the VOC file. Standard Pick Master Dictionary entries that match standard UniVerse VOC file entries (LIST, SORT, and so forth) are compared against a template file corresponding to the specific Pick system on which the account was developed. If the entry does not match the template file entry, it is not converted. This is indicated in the error report that CONVERT.VOC generates. If the entry matches the template file entry, no action is taken, and the standard UniVerse VOC file entry is not affected. CONVERT.VOC does not overwrite any existing records in the VOC file.

Convert Pick file dictionaries to UniVerse file dictionaries

When you move a file from an account on a Pick system to an account on UniVerse, the DC utility places the original Pick dictionary items in a special file, called the Pick dictionary. This Pick dictionary is a UNIX file with the name `P_filename` and can be referenced in a UniVerse command sentence with the keyword `PDICT` (such files are sometimes called `PDICT` files). You can use a special file dictionary, `DICT.PICK`, (which is like `DICT.DICT` for UniVerse dictionaries) to provide a Pick-like display of a dictionary listing.

The DC utility copies all or selected items in a Pick-style file dictionary. The conversion process builds the appropriate `@` phrase and makes minor modifications to some entries. For example, `REALITY`-style dictionary entries are first converted to Pick-style.

After all the records are converted, the DC utility invokes `CD` (the dictionary compiler utility) on the UniVerse file. `CD` compiles all the I-descriptors in the new UniVerse dictionary. (The UniVerse dictionary's name is `D_filename`, which is referenced with the keyword `DICT`.) The original Pick dictionary is retained.

To convert a file dictionary, complete the following steps:

1. Type **CONVERT.ACCOUNT** from the UniVerse command line.
2. Select **convert Dictionaries** from the **convert Dictionary** submenu.
3. Select **F** if you want to see the items being converted.
4. Select **Y** if you are ready to begin the conversion.

Converting REALITY procs

If you are converting a `REALITY` account, you must convert all procs with the **convert Procs** option of the `convert Dictionary` submenu. This utility changes all instances of the `REALITY` proc command `"X"` (exit) to `"Q"` (quit) in all procs. This needs to be done because on `REALITY` systems, `"X"` exits the proc and returns to `TCL`, whereas on UniVerse (and most other Pick-related systems), `"X"` returns to the calling environment (another proc or a program). `"Q"` on UniVerse exits to `TCL`.

Only procs from `REALITY` systems need to be converted in this way. Do *not* use the **convert Procs** option on any other Pick account or on Prime INFORMATION accounts.

Converting dictionaries with the DC command

You run DC from the Account Conversion menu by using the CONVERT.ACCOUNT command. DC can also be executed directly from the UniVerse command line. The syntax for DC is:

```
DC [ -options ] [ filename [ record.IDs ] ]
```

options can be one or more of the following options, specified in any order:

Option	Description
L	Sends output to the system printer.
M	Specifies that the dictionary is in REALITY format.
N	Does not send output to the system printer.
O	Specifies that the dictionary is not in REALITY format.
P	Specifies that you intend to maintain Pick-style dictionaries.
U	Specifies that you intend to maintain UniVerse-style dictionaries.
Y	Sends output to the system printer.

DC Options

filename is the name of the file to be converted.

record.IDs is the list of record IDs of the records to be converted. Separate record IDs with spaces.

If you enter DC without arguments and there is no active select list, the following prompt appears:

```
Enter name of dictionary to convert, or "*" for all local
dictionaries?
```

Entering a file name converts a single file and sends a report to the printer. Entering an asterisk (*) builds a select list of all local dictionaries, and prompts you to approve each dictionary before converting it. It also sends a printed report.

If there is an active select list and no files are included on the command line, the elements of the select list are used as the dictionaries to be converted.

If the dictionary converter encounters any items it cannot convert, it prints the following message:

```
Error listing written to file "&SAVEDLISTS&" item
"DC.ERRORS_#####".
```

is the time in internal format, and #### is the date in internal format.

This record contains just the file/record ID pairs that caused DC trouble. You can examine the record with the Editor.

Converting UniVerse BASIC program files

You must compile all programs and \$INCLUDE files contained in the Pick or Prime account you are converting before you can use them in the UniVerse environment. Before you compile programs, convert the source code files to UniVerse type 1 files, and delete the old object code. The CONVERT.PGMS program converts Pick BASIC source programs and \$INCLUDE files into type 1 files and deletes the obsolete object code.

1. Select **convert source Files** from the Convert basic programs submenu to run the CONVERT.PGMS program.
2. On the data entry screen, enter the names of all files containing programs to be converted, or enter * to specify all program files.

Select BASIC programs - CONVERT.PGMS

BASIC program file names

1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
.1.10

Enter BASIC program file name ?

2. Source Machine :

Help Region

Press <F1> for longer help about any particular entry, or <ESCAPE> to exit program.

Enter name of BASIC program, <I> to insert a value, <DE> to delete a value, or + to page down.

Recompiling and cataloging BASIC programs

The UniVerse BASIC compiler accepts the same BASIC source code as a Pick compiler, but produces object code that is very different. You must recompile all BASIC programs in UniVerse before you can use them. Select **Compile basic files** from the Convert basic programs submenu to recompile all BASIC programs.

When you convert the Master Dictionary (or VOC file) with the CONVERT.VOC utility, it creates a paragraph &CATALOG.ALL& that contains the names of all programs defined as catalog entries in the MD. You can edit and execute this paragraph to catalog all programs you want to catalog.

1. Select **Compile basic files** from the Convert basic programs submenu to run the CONVERT.PGMS program.
2. On the data entry screen, enter the names of all files containing programs you want to convert, or enter * to specify all program files.

See Chapter 8, "[Chapter 8: UniVerse BASIC](#)," for more information about compiling and cataloging BASIC programs.

Where to go from here

At this point you can use the system. The remaining chapters in this book summarize important concepts and describe the UniVerse processors, detailing differences that Pick, REALITY, and IN2 users will notice, as well as specific differences in account flavors.

Chapter 3: The VOC file

The UniVerse command processor (TCL)	3-3
Special character interpretation	3-3
PHANTOM processes	3-4
The VOC file (master dictionary).	3-6
VOC file record format.	3-6
VOC file entry types	3-9
The VOCLIB file	3-16
Listing VOC file entries	3-16
The command processor sentence stack	3-17

This chapter explains the UniVerse versions of the Terminal Control Language (TCL) and the Master Dictionary (called the VOC file in UniVerse), both familiar to Pick users. The last part of the chapter introduces the TCL sentence stack.

The UniVerse command processor (TCL)

The UniVerse Command Processor is the equivalent of the Pick Terminal Control Language (TCL). The Command Processor examines every input line entered at the system prompt as well as command lines that are entered from a stored command sequence, a proc, or a UniVerse BASIC program.

Depending on the verb in the command sentence, the Command Processor either executes the command itself or calls the proper processor to complete the execution. The action taken by the Command Processor, as well as by other processors, depends on definitions contained in the VOC file for the words in the command line.

UniVerse, unlike Pick, gives you the ability to store any sentence or sequence of sentences (called a *paragraph*) in the VOC file for execution later. It also maintains a list of the last 99 command lines entered at the terminal. This list is called the *sentence stack*. The default number of commands to be saved in the stack is 99, but it can be changed by the System Administrator. You can use the sentence stack to recall, delete, change and reexecute a previous command, or to save a sentence or sequence of sentences in the VOC file.

Special character interpretation

UniVerse uses the traditional Pick system delimiter characters. You can change these defaults with the PTERM command. The Command Processor recognizes the following three control characters as having special significance:

Control Character	Description
^^	(the control key and the up-arrow or caret key) is a field (i.e., attribute) mark (^254).
^]	(the control key and the right bracket) is a value mark (^253).
^\ ^\\	(the control key and the backslash) represent a subvalue mark (^252).

Special Control Characters

The Command Processor also recognizes that the period (.) and the hyphen (-) can be used interchangeably in verbs and keywords. That is, if you enter the command CREATE-FILE and the Command Processor cannot find a VOC entry with that ID, it searches for CREATE.FILE.

Other characters that have special functions can be displayed by typing the PTERM DISPLAY command.

```

>PTERM DISPLAY
MODE          EMULATE
CC            INTR   = ^C  QUIT   = ^\  SUSP   = OFF DSUSP  = OFF
              SWITCH = ^@  ERASE  = ^H  WERASE = OFF KILL   = ^U
              LNEXT  = OFF REPRINT = OFF EOF   = ^D  EDL    = ^@
              EOL2   = ^@  FLUSH  = OFF START  = ^Q  STOP   = ^S
              LCONT  = ^_  FMC    = ^^  VMC    = ^J  SMC    = ^\
              TMC    = ^T  SQLNULL = ^N

INPUTCTL      ON
CARRIER      RECEIVE -HANGUP -LOCAL
CASE          -UCIN -UCOUT -XCASE INVERT
CRMODE        -INLCR -IGNCR ICRNL ONLCR -OCRL -ONOCR -ONLRET -CRONLY
DELAY         BSB CRB FFB LFB VT0 TAB0 -FILL
ECHO          ECHO ERASE=BSB KILL=LF CTRL -LF
HANDSHAKE     XON -ANY -TANDEM -DTR
OUTPUT        POST -TILDE -BG CS -EXPAND
PROTOCOL      LINE=0 BAUD=38400 DATA=8 STOP=1 NONE DISABLE -STRIP
SIGNALS       ENABLE FLUSH BREAK=INTR
>

```

This report shows all the terminal characteristics currently set for a Wyse 50 terminal.

PHANTOM processes

Phantom processes are not new to most Pick users. UniVerse handles them differently. You can run any command in the background provided it does not require terminal input. To start a background process, simply add the verb PHANTOM before the command. Each phantom process creates a record in a type 1 file called &PH& to store output resulting from the command. If the process requires terminal input, you must use DATA statements in a paragraph to supply the input. See Chapter 9, [“Chapter 9: Application development tools,”](#) for information about paragraphs.

When the process is started, a message such as the following is displayed:

```
Phantom process started as Process ID pid#.
```

UNIX assigns the process ID number, *pid#*.

To monitor the phantom process, type **STATUS ME**. The phantom process appears under your account name.

When a phantom process finishes, UniVerse notifies you the next time you return to the UniVerse prompt. If you want to be notified immediately, use the NOTIFY ON command.

You can also stop a phantom process (started before the current login session) before it is finished by typing the following:

```
LOGOUT pid#
```

The *pid#* is the process ID number assigned when the process started.

The VOC file (master dictionary)

The UniVerse equivalent to the Pick Master Dictionary is the VOC file. It is called the VOC file because it contains the vocabulary of commands and keywords that are available to users of the account. In PICK, IN2, and REALITY flavor accounts, the VOC file contains a Q-pointer to the VOC file called MD; this lets you refer to the VOC file either as MD or as VOC.

As on other Pick systems, every word you use in a UniVerse sentence must be defined either in the VOC file or in the dictionary of the file being processed by the command; otherwise it will be interpreted as a record ID in the data file being processed, or as a constant representing a data value stored in a field.

Every UniVerse account has its own VOC file containing records for every verb, sentence, paragraph, menu, file name, remote pointer, and keyword that you can use while working in the account. The VOC file can also contain phrases that are not specific to a particular data file and thus are not stored in the file's dictionary. VOC file records are also called VOC entries.

Unlike other Pick systems, the VOC file is a UniVerse file on the same level as any other data file or file dictionary and has its own dictionary, DICT VOC.

Using the Editor you can create sentences, paragraphs, and phrases and save them in the VOC file. You can also create and save sentences and paragraphs using the sentence stack commands. You can create menus using the menu processor or the Editor. See Chapter 9, "[Chapter 9: Application development tools](#)," for more information about sentences, paragraphs, and menu entries.

Every entry in the VOC file must have a name. You can then use that name in a command line. The Command Processor verifies that the entry is in the VOC file, then processes it according to the entry type.

VOC file record format

The internal formats of UniVerse VOC entries are different from Master Dictionary entries on other Pick systems. The CONVERT.VOC program handles these differences.

The VOC file dictionary (DICT VOC) contains the names and formats of the fields, which differ for different types of entry. Record IDs in the VOC file can be referenced by two field names: NAME and F0. F0 is equivalent to *A0 on Pick systems. The record ID is sometimes called field 0, but, as on other Pick systems, it is not truly a field in the VOC entry.

The first field of every VOC entry is the same for all VOC entry types and is defined in the VOC dictionary as F1 (*A1, or D/CODE, on Pick systems). F1 contains the *type code* and can have two components: the type code itself, and a brief description of the entry. The first one or two characters in F1 define the type of VOC entry. The type code must be one of the codes in the following table:

Code	Description	Pick Equivalent
D	Data field definition	A, S, X
F	File pointer	D
I	Interpretive field definition	—
K	Keyword	C
M	Menu record	—
PA	Paragraph	—
PH	Phrase	—
PQ	Proc	PQ
PQN	Proc	PQN
Q	Q-pointer	Q
R	Remote command	—
S	Stored sentence	—
V	Verb	P
X	User record	X

VOC File Dictionary

The VOC file dictionary (DICT VOC) defines fields 2 through 5 as F2, F3, F4, and F5. It also includes synonym definition items which differ for each entry type. For example, if the entry is a file definition, field 2 contains the UNIX path for the data file and field 3 contains the UNIX path for the file dictionary. DICT VOC therefore contains the item DATA.FILE as a synonym definition of field 2, and the item DICT.FILE as a synonym of field 3.

Fields after field 5 are generated by UniVerse and contain a variety of information, such as file names of multiple data files associated with the same file dictionary.

The following table shows the record formats for 11 VOC entry types. Information shown in curly braces is optional. Most of these VOC entry types are described in more detail in the following sections. Full details on VOC entries are in *UniVerse System Description*.

VOC Entry Type					
Field	Verb	Remote Command Item	Stored Sentence	Paragraph	File Defining Item
F1	V [<i>descr</i>]	R [<i>descr</i>]	S [<i>descr</i>]	PA [<i>descr</i>]	F [<i>descr</i>]
F2	processor	filename	sentence	sentence1	UNIX path for data file
F3	dispatch type	record ID	[<i>continuation of sentence</i>]	sentence2	UNIX path for file dictionary
F4	processor mode	[<i>security subroutine</i>]		...	[M]
F5	reserved			...	[<i>UNIX path for Pick dictionary</i>]

VOC Entry Types

VOC Entry Type					
F6	reserved			...	[<i>subitem field</i>]
F7	reserved				[<i>filenames, multiple data files</i>]
F8	reserved				[<i>UNIX paths, multiple data files</i>]

VOC Entry Types (Continued)

VOC Entry Type						
Field	Q-Pointer	Keyword	Phrase	Menu	User Record	ProVerb
F1	Q [<i>descr</i>]	K [<i>descr</i>]	PH [<i>descr</i>]	M [<i>descr</i>]	X [<i>descr</i>]	PQ [<i>descr</i>]
F2	account	operation number	phrase text	filename	user-supplied information	proc statement 1
F3	filename	[<i>sentence</i>]	[<i>continuation of phrase</i>]	record ID	...	proc statement 2

VOC Record Entry Types

VOC file entry types

This section points out the major differences between UniVerse VOC file entries and their Pick equivalents in the Master Dictionary. Attribute Definition items, which in UniVerse are called either Data Descriptors or I-Descriptors (“I” stands for “Interpretive”). PQ and PQN entries, which are procs. A complete description of the contents of each field for each record type can be found in *UniVerse System Description*.

Verbs and other commands

The UniVerse VOC file contains four entry types that define commands, or that point to verbs or other commands stored in other accounts or files. Verbs in UniVerse are similar to verbs on other Pick systems. Remote command items, stored sentences, and paragraphs may be new to you.

V: verb

Verbs are identified by a “V” in field 1, which is equivalent to “P” on other Pick systems. UniVerse does not distinguish between TCL-I, TCL-II, and ACCESS verbs, so the “V” is not followed by an optional second code letter (as it may be on a Pick system). The VOC entry for a verb specifies the processor that the verb invokes, the dispatch type, and the flags that the processor uses. See *UniVerse System Description* for information about dispatch types and processor flags.

To display all the verbs in the VOC file, use the LISTV command. LISTV is the UniVerse equivalent of the LISTVERBS proc.

R: remote command item

A remote command item is a record in the VOC file that points to a command stored as a record in another file. By placing the remote command item in the VOC file, you can store larger command records, such as paragraphs or procs, in another file (the VOCLIB file, for instance) and keep the VOC file from becoming unwieldy. You can also store all the records that pertain to an application in a single file and simply point to them from the VOC file. The VOC file entry for a remote command item specifies the UniVerse file name and the record ID of the remote item.

To display the remote command items in the VOC file, use the LISTR command.

S: stored sentence

A sentence is a complete command line, including the verb. If you often use the same sentence and want to avoid typing the entire command each time, you can store the sentence in your VOC file. This feature is especially useful if the command line is very long. Each sentence can be one of the following:

- a complete sentence

- the name of another stored sentence
- the name of a paragraph
- the name of a menu

The record ID is the sentence name. To use the sentence, enter the sentence name as the first word in a command line. Any other words that you enter after the sentence name are appended to the stored sentence before the command line is executed.

You can use the Editor to create a stored sentence in the VOC file, or you can use the sentence stack command `.S` to save a sentence.

To examine the stored sentences in the VOC file, use the `LISTS` command.

PA: paragraph

A paragraph is a series of sentences stored together under one name. This feature lets you run several commands by typing the name of the paragraph. The VOC file entry for a paragraph contains the sentences that make up the paragraph and special paragraph control statements. The record ID is the paragraph name. To use the paragraph, enter the paragraph name at the TCL prompt. UniVerse executes one sentence at a time, beginning with the first sentence and ending with the last sentence, as though each command were entered at the keyboard.

A nice feature of paragraphs is that they allow inline prompting. Inline prompting displays a prompt that requests an input value when the paragraph is executed. The paragraph is executed as if the input value had been entered instead of the inline prompt specification.

You can use the Editor to create a paragraph in the VOC file, or you can use the sentence stack command `.S` to save a series of sentences as a paragraph.

To examine the paragraphs defined in the VOC file, use the `LISTPA` command.

File defining items

UniVerse file-defining items are different from their Pick equivalents, although UniVerse Q-pointers are the same as they are on other Pick systems. The UniVerse equivalent to a D-pointer is an F-type (file-defining) entry in the VOC file. Because UNIX points to file locations by name instead of by a frame address, a UniVerse file-defining item specifies a UNIX path as the location of the file to which it points.

F: file description

The UniVerse equivalent of a D-pointer is an item in the VOC file with a type code of “F” (for File). A file-defining item normally specifies both the data file and the file dictionary. The record ID is the UniVerse file name.

Normally both the data file and its dictionary are created at the same time to avoid confusion. If you create only the file dictionary, only field 3 contains a path. If you create only the data file, only field 2 contains a path. (Note that you can create a data file without an associated dictionary in UniVerse, which you ordinarily cannot do on a Pick system.) You might create only a file dictionary if you were using an existing data file and wanted to create a new dictionary for that file. You might create only a data file if you wanted to use an existing dictionary with a new data file. You can use the Editor or REVISE to add the name of the existing file to a VOC entry. See Chapter 4, [“Chapter 4: UniVerse file structure,”](#) for a complete discussion of paths for the data file and the file dictionary.

To list the UniVerse files defined in your VOC file, use the LISTF command. LISTF is the UniVerse equivalent of the LISTFILES proc. You can use the LISTFILES command in PICK, REALITY, and IN2 flavor accounts. To display only the files that are located in your own UniVerse account, use the LISTFL command. And to display only the files that are located in remote accounts, use the LISTFR command.

Creating file synonyms

You can create synonyms for any UniVerse file. Synonyms let you refer to the same file using a different name. Since you can define multiple dictionaries for the same data file, UniVerse file synonyms are a little different from Pick file synonyms. With UniVerse file synonyms, you can use different file names to refer to the same data file but using different dictionaries.

For example, you might have a data file called EMPLOYEES with its associated dictionary called D_EMPLOYEES. Suppose you want to use another dictionary called D_EMPL.PAYROLL with the EMPLOYEES data file. To do this, create a synonym for the EMPLOYEES file that specifies the D_EMPL.PAYROLL dictionary in field 3 (instead of D_EMPLOYEES). Call the synonym EMPL.PAYROLL. The VOC file entry for EMPL.PAYROLL will look as follows:

```
          EMPL.PAYROLL
0001 F Synonym for EMPLOYEES file using EMPL.PAYROLL dictionary
0002 EMPLOYEES
0003 D_EMPL.PAYROLL
```

This capability means that you need not use the USING keyword in a RETRIEVE sentence to specify another file dictionary when referencing a data file.

Remote file pointers

You can also create VOC entries that point to files in other UniVerse accounts, just as you can on other Pick systems. You can point to a remote file either directly, using a full or relative UNIX path, or through the VOC file of the remote account, using a Pick-style Q-pointer. A remote file pointer using UNIX paths must be changed if you change the location of the account where the file is located. For this reason a

Q-pointer, which points indirectly to the remote file, is more portable than a remote file pointer. Provided the system administration files are updated properly, the

Q-pointer continues to point correctly to the remote file even if the account is moved to a different UNIX directory.

You must have the proper access authorization for such files in order to gain access to them. Use the SETFILE command or the Editor to create an entry using the UNIX path. Use SET-FILE or SET.FILE to create Q-pointers. For more information on file access permissions, see Chapter 4, [“Chapter 4: UniVerse file structure.”](#)

(Note that SETFILE and SET.FILE are two different commands: SETFILE is one word, SET-FILE and SET.FILE are two words connected by a hyphen or a period.) Note that SET.FILE creates a permanent Q-pointer in the VOC file; it does not create a temporary record called QFILE that changes every time the SET.FILE command is used. See Section 4.8 for details about how to use the SETFILE and SET.FILE commands.

To display the remote files referenced in your VOC file, use the LISTFR command.

Keywords and phrases

Keywords are similar to connectives, modifiers, and options on other Pick systems. User-defined phrases are new to most Pick users. Phrases are parts of command sentences.

K: keyword

Keywords either define an operation that is to take place in the sentence, or they modify the action of a verb. The VOC file entry for a keyword specifies the internal operation number for that keyword. The record ID is the keyword itself.

You can create a synonym for a keyword by creating a new record in the VOC file that contains the same information as the original keyword. You can then use the new keyword in a sentence or phrase and obtain the same result as if you had used the original keyword.

To examine all the keywords defined in the VOC file, use the LISTK command. LISTK is the UniVerse equivalent of the LISTCONN proc.

PH: phrase

A phrase is a part of a Retrieve or REVISE sentence, very much like a phrase in an English sentence. Phrases can contain any element of a Retrieve sentence except the verb. The VOC file entry for a phrase contains words that make up the phrase. The record ID is the phrase name. Normally phrases are stored in the dictionary of the file in which they are to be used. You may, however, want to define phrases in the VOC file that can be used with several files in the same account.

Use the Editor or REVISE to create a phrase entry in the VOC file. For example, you might define the following part of a Retrieve sentence as a phrase called DATA.ONLY:

```
WITH TYPE = "F" F1 F2
```

When you type

```
LIST VOC DATA.ONLY
```

UniVerse will execute the command as if you had typed

```
LIST VOC WITH TYPE = "F" F1 F2
```

To examine all phrases defined in the VOC file, use the LISTPH command.

Menu items

A menu is a record in a UniVerse file that the Command Processor uses to display a list of choices on the terminal. You can select a process from the menu by entering the number of one of the choices. The VOC file entry for a menu contains the name of the UniVerse file where the menu is stored and the record ID of the menu.

You can create menus using the Menu Processor or the Editor. See the *UniVerse System Description* for details about menus.

To display all the menu records defined in your VOC file, use the LISTM command.

User records

Pick users should note that UniVerse user records (X entries) do not function as place-keepers in a sequence of default output specifications.

A user record stores information defined by the user. Use the Editor or REVISE to create a user entry in the VOC file.

Use the LISTO (for LIST Other) command to list the user records in your VOC. LISTO also lists D entries and I entries contained in the VOC file.

RELLEVEL and STACKWRITE are two X records defined in standard VOC files.

RELLEVEL defines the release level of the software and is used to verify that the VOC file has been updated to the new level. If the UniVerse release level does not agree with that in RELLEVEL, UniVerse displays a warning message and requests permission to update the VOC file. Using an out-of-date VOC file can cause errors.

STACKWRITE tells the sentence stack processor whether or not to maintain the sentence stack after you log out. If the second field of this record contains ON, the sentence stack is saved when you log out. If this field contains OFF, the sentence stack is not saved.

&NEXT.AVAILABLE& is another X record, which is created and maintained in a dictionary when REVISE is invoked with the NEXT.AVAILABLE keyword. It supplies the next value in a numerical sequence to be used as the record ID.

The VOCLIB file

The VOCLIB file is a standard UniVerse file that is created when a new account is created. It is empty when a new account is created. You can use the VOCLIB file to store long paragraphs which are pointed to by remote command items in the VOC file. We recommend that you store your long paragraphs in this file and use remote command items in the VOC file to point to items in the VOCLIB file. This practice keeps the VOC file from becoming too large and ensures rapid access to its records.

Listing VOC file entries

Throughout this section, several commands that let you see the different types of entry in the VOC file were mentioned. They are summarized in the following table.

Sentence	Description	Pick Name
LISTF	List all file definitions.	LISTFILES
LISTFL	List definitions of files stored in this account.	—
LISTFR	List definitions of files stored in other accounts.	—
LISTK	List all keywords.	LISTCONN
LISTM	List all menu selector records.	—
LISTO	List “other” entries.	—

Summary of Commands for Listing VOC Entries

Sentence	Description	Pick Name
LISTPA	List all paragraphs.	—
LISTPH	List all phrases.	—
LISTPQ	List all procs.	LISTPROCS
LISTR	List all remote command items.	—
LISTS	List all stored sentences.	—
LISTSL	List all verbs that can use a Select List.	—
LISTUN	List the UNIX commands in your vocabulary.	—
LISTV	List all verbs.	LISTVERBS

Summary of Commands for Listing VOC Entries

The command processor sentence stack

Sentence stack commands begin with a period (.). Use sentence stack commands to list sentences stored in the stack; to save, edit, delete, recall, and reexecute stored sentences; and to insert new sentences. Sentence stack commands let you avoid repetitive typing. You can also correct mistakes without retyping an entire sentence. Sentence stack commands themselves are not saved in the sentence stack. For more information about sentence stack commands see the *UniVerse System Description*.

You can also save your sentence stack after you log out by changing the value of field 2 of the STACKWRITE entry in your VOC to ON.

Chapter 4: UniVerse file structure

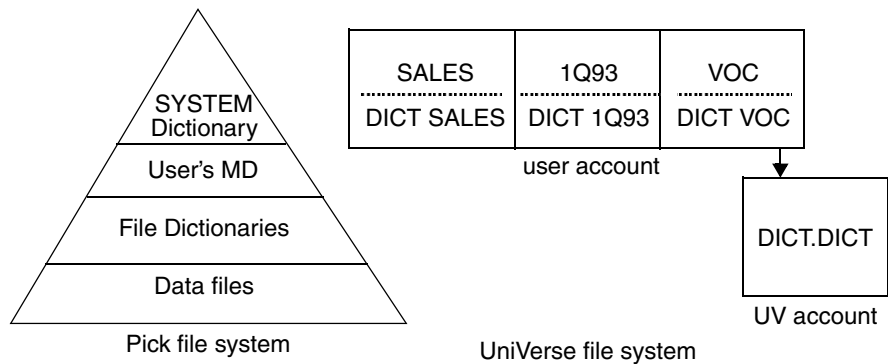
The UniVerse file system	4-3
Hashed and nonhashed files	4-5
Records in a hashed file	4-5
Static hashed file structure	4-7
Dynamic hashed file structure	4-8
Nonhashed file structure	4-10
B-Tree files and secondary indices	4-12
Secondary indices	4-13
Multiple data files	4-14
File pointers	4-16
SETFILE command	4-16
The SET.FILE command	4-17
File protection	4-19
Explicit locks	4-19
Implicit locks	4-20
UNIX file protection.	4-21
Maintaining files	4-24
GROUP.STAT and HASH.TEST commands.	4-24
More file statistics	4-25
Maintaining dynamic files	4-26
Resizing files	4-26
Cleaning up an account	4-28

This chapter describes the different kinds of file available in the UniVerse environment, focussing on those not found on standard Pick systems.

UniVerse files are similar to files on other Pick systems. However, UniVerse has added several enhanced capabilities to its file system that may be new to you. Since the file system itself is handled at the UNIX level, it differs significantly from the four-level Pick file hierarchy found on generic Pick systems.

The UniVerse file system

The UniVerse file system does not use the standard Pick pyramid-like structure, with the SYSTEM Dictionary at the top level, user account Master Dictionaries at the second level, file dictionaries at level three, and data files at level four. In UniVerse there is no SYSTEM Dictionary, nor is there an equivalent to a top-level file.



Pick Structure vs. UniVerse Structure

UniVerse accounts can be located in any UNIX or Windows directory at any level of the file hierarchy. There is no equivalent in UniVerse to the SYSPROG account, although the UV account in the UniVerse directory is in some ways analogous to it.

An account's VOC file (Master Dictionary) is like any other UniVerse data file. It is not on a higher level than other UniVerse files. Unlike Master Dictionaries on other Pick systems, each VOC file has its own dictionary (DICT VOC). In addition, each account's VOC file has a pointer to a file in the UV account called DICT.DICT that acts as a master dictionary for all file dictionaries in all UniVerse accounts on the system.

All of an account's data files and file dictionaries, including its VOC file, are located in the same directory. UniVerse file dictionaries are stored at the same "level" as their associated data files. Thus UniVerse does not use (or need) D-pointers in its file dictionaries to point to the associated data files. Instead, UniVerse uses two different records to handle the functions of the Pick D-pointer.

- The file definition record in the VOC file defines the relationship between data files and their associated dictionaries.
- The @ID record in any file dictionary handles display functions such as column heading, display format, and so forth, for record IDs in the data file.

Hashed and nonhashed files

The relationship between a data file and a file dictionary is defined by a file definition entry in the VOC file, not by a D-pointer in the file dictionary. Data files and their dictionaries are implemented using UNIX files and directories. In addition, the UniVerse BASIC language provides facilities for accessing UNIX files that are not UniVerse files.

As on all Pick systems, data files and file dictionaries are organized as collections of records. Each record is identified by a record ID that must be unique within the dictionary or data file. UniVerse provides several kinds of organization for data files:

- Hashed files, which can be static or dynamic
- B-tree files
- Nonhashed files

Data files can be either hashed or nonhashed, depending on the type of data stored in them. File dictionaries are usually hashed.

Nonhashed files are used to store text, program source code, or other data that does not have much structure to it (for example, binary data). A nonhashed file is actually implemented as a UNIX directory. Each record in a nonhashed file is a UNIX file in that UNIX directory.

Hashed files are like Pick files: they use a hashing algorithm to distribute records in one or more groups of the file. UniVerse, however, uses several different hashing algorithms rather than the single one used by most Pick systems.

Records in a hashed file

As on other Pick systems, each record in a hashed file is a variable-length string of characters. Records are organized as they are in Pick, with fields, values, and subvalues separated by the familiar field marks (attribute marks—ASCII 254), value marks (ASCII 253), and subvalue marks (ASCII 252). The information for each group, record and item is stored differently on disk. In each group buffer, records are stored as linked lists of strings containing:

- The data
- The record ID and its delimiter
- 12 bytes of control information

Each record in a file comprises a header, a record ID, and the data. Record headers contain forward and backward link pointers (four bytes each), and four bytes used for record status flags. The record ID is separated from the data by a segment mark (ASCII 255). The header is composed of three “words.” The first word is a pointer to the beginning of the next record in the group. The second word contains a pointer to the beginning of the previous record in the group, EXCLUSIVE ORed with the length of the record. (The information contained in the second header word is included only for redundancy of data. The system implicitly knows the position of the previous record, since it had to go through that record to gain access to the current record, and the length of the record is implied in the pointer to the next record.) The third word of a record header contains several flags for administration of the record.

Static hashed file structure

Static hashed files are handled as on any standard Pick system. When you create a static hashed file, *separation* and *modulo* specify the group buffer size and the number of group buffers allocated for the file.

In addition, in UniVerse you can also specify a *file type*. The file type determines which of several possible hashing algorithms should be used to distribute the records in groups. You select a file type based on the kind of record ID you are using.

When you create a static hashed file, UniVerse creates a system-level file that contains the number of groups specified by *modulo*. Unlike Pick, where *separation* is the number of contiguous frames, UniVerse uses *separation* to specify the size of a group in multiples of 512 bytes. In addition, a header block the same size as the group is created. Each group is initialized with an empty record the size of the group.

As new records are written to the file, a group buffer may not be large enough to contain all the records hashed to it. The file handler then extends the system-level file with an *overflow block* (another group). The overflow block is chained to the primary group so that any number of records can be placed in a group.

As records are updated, the file handler tends to move records to the beginning of a group and move empty space to the end. If an overflow block is no longer needed, it is returned to a free block list for the file, to be used as an overflow block for some other group. The overall size of the system-level file is not reduced until an explicit RESIZE command is used.

Dynamic hashed file structure

Static hashed files, like their Pick counterparts, must be resized when the amount of data stored in them changes. As you know, this can be troublesome when files change size frequently.

To alleviate this problem, UniVerse provides the type 30 file, a hashed file that resizes itself dynamically. Dynamic files automatically change modulo as they grow or shrink. The number of groups increases by one whenever the percentage of data in the file exceeds a specified amount (by default, 80%), called the *split load*. The number of groups decreases by one whenever the percentage of data in the file falls below a specified amount (by default, 50%), called the *merge load*. In this way a dynamic file is sized efficiently even when the amount of data in it changes often. You can use a dynamic file just as you would use a hashed file, without manually resizing the file if it changes in size.

Use the CREATE.FILE command to create a dynamic file. The syntax for creating type 30 files is as follows:

```
CREATE.FILE filename 30 [ parameter [ value ]... ] [ description ]
```

You can use the keyword DYNAMIC in place of 30 to create the dynamic file. CREATE.FILE creates a dynamic data file called *filename* and its associated dictionary. (The dictionary is not created as a dynamic file.) The initial file has a group size of 1, or 2048 bytes, which is equivalent to a separation of 4. It uses a general hashing algorithm and has an initial modulo of 1.

A dynamic data file is implemented as a UNIX or Windows directory containing two files named DATA30 and OVER30. The DATA30 file has an internal structure similar to that of a static hashed data file with no overflow. The OVER30 file has an internal structure similar to the overflow block portion of a static hashed file.

You can specify additional parameters to the CREATE.FILE command which allow you to tune the specific characteristics of your dynamic file. Use the ANALYZE.FILE command to verify that your changes to the default parameters really increase the efficiency of the file. See the CREATE.FILE command in *UniVerse User Reference* for a complete description of the additional parameters for dynamic files.

To convert a static hashed file to a dynamic file (or to convert a dynamic to a static hashed file), use the RESIZE command.

For some files, efficiency can be increased by changing the default parameters for the file. To change file parameters for existing dynamic files, use the CONFIGURE.FILE or the RESIZE command. The *UniVerse System Description* discusses how to use these commands.

Nonhashed file structure

Nonhashed files are commonly used to store text files or BASIC source code and listing modules. A nonhashed UniVerse file is implemented as a UNIX or Windows directory. Each record in a nonhashed file is a UNIX or Windows file in that directory.

Unlike Pick, you store BASIC source programs in UniVerse nonhashed files. Each record in such a file contains an entire BASIC program, with ASCII NEWLINE characters separating each source line. Since the records in a nonhashed file are UNIX or Windows files, you can access them from a shell environment with the standard UNIX or Windows editors or utilities. You can access nonhashed files from the UniVerse environment using the Editor or other UniVerse processors. The UniVerse Editor treats the characters between NEWLINE characters as fields.

On many Pick systems it is standard practice to name the file that contains program source code BP. In UniVerse, the BP file should be created as a nonhashed file, as in the following example, done in an IDEAL flavor account

```
>CREATE.FILE BP 1
Creating file "BP" as Type 1.
Creating file "D_BP" as Type 3, Modulo 1, Separation 2.
Added "@ID", the default record for Retrieve, to "D_BP".
```

In a PICK, IN2, or REALITY flavor account, use the CREATE.BFILE command:

```
>CREATE.BFILE BP 1,2
Creating file "BP" as Type 1.
Creating file "D_BP" as Type 18, Modulo 1, Separation 2.
Added "@ID", the default record for Retrieve, to "D_BP".
```

If you create BP as a nonhashed file and use the Editor to create two BASIC programs called PROGRAM.1 and PROGRAM.2, UniVerse creates the following directory and files:

File/Directory	Description
BP	A UNIX or Windows subdirectory in the current working directory.
D_BP	A hashed UNIX or Windows file (the file dictionary) in the current working directory.
PROGRAM.1	A UNIX or Windows ASCII file in the BP directory. This file contains the source code for PROGRAM.1 separated by NEWLINE characters.
PROGRAM.2	A UNIX or Windows ASCII file in the BP directory. This file contains the source code for PROGRAM.2 separated by NEWLINE characters.

When you are working in the UniVerse environment, you need not be aware of the UNIX or Windows file structure to use UniVerse commands and utilities. Use the UniVerse file name and record names just as you would on any other Pick system. For example, to compile PROGRAM.1 from UniVerse, enter the following command:

```
>BASIC BP PROGRAM.1
```

To edit PROGRAM.1 source code using the UniVerse Editor, enter the following:

```
>BED BP PROGRAM.1
```

To edit PROGRAM.1 source code using *vi*, however, whether from a UNIX shell environment or from UniVerse, you must use the UNIX path, as in the following examples:

```
$vi BP/PROGRAM.1
>VI BP/PROGRAM.1
```

B-Tree files and secondary indices

Unlike Pick, you can use a UniVerse B-Tree file to improve access to stored data when searching on the first part of a key. In a hashed file, each record is stored in a group buffer so that only one group buffer need be searched to find any record. In a B-tree file, records are stored in sorted order. To find a record, its record ID is compared to the value at the center of the tree. If it is greater, the search continues with the subtree to the left of the center element; if it is less, the search continues with the subtree to the right. This process continues until the record is found.

B-tree files are type 25 files. You create a type 25 file by using the `CREATE.FILE` command. The syntax for the `CREATE.FILE` command is as follows:

```
CREATE.FILE [ DICT ] filename 25 [ description ]
```

Because record IDs are stored in sorted order, `LIST` displays the records from a

B-tree file sorted by record IDs. Unlike other nonhashed file types, you should not store large records in type 25 files—records larger than 1K increase access time significantly.

When you create a B-tree file, UniVerse creates a single UNIX or Windows file that contains a 2K header and a series of 8K buffers that are organized into leaf node buffers and internal node buffers. The internal node buffers contain the B-tree structured keys, and the leaf node buffers contain the data parts of the records in the file. As records are added to the B-tree file, additional 8K buffers are allocated to the UNIX or Windows file as needed to contain the key and data portions of the records. As records are deleted, the buffers that are emptied are not returned to the UNIX or Windows file system. To reduce the size of the file, you must use the `RESIZE` command:

```
RESIZE filename 25
```

Secondary indices

Secondary indices are new to most Pick users. They are stored in nonhashed type 25 files. A secondary index is a sorted list of the values in a specified field. The secondary index facility allows fields other than the record ID to be used as the key field in selection and sort expressions with more efficiency. Take as an example the following selection sentence:

```
>SELECT FILE WITH F2 < 100
```

Without secondary indexing, every record in the file named FILE is searched for records with the value of field 2 less than 100 to create the select list. If field 2 has a secondary index, each item in the index is searched in increasing order until any item is greater than 100. When an item is found that is greater than 100, all other items are assumed not to match and the search is ended. The select list then contains all the record IDs of the elements searched except the one that did not match. If all the items in the file have a value for field 2 less than 100, the search time is the same as without a secondary index, but if only a few of the items have a value less than 100, the search time is greatly reduced.

Multiple data files

As on other Pick systems, multiple data files can be associated with a single shared dictionary. In UniVerse you can use the CREATE.FILE command to create each data file that you want to associate. UniVerse, unlike Pick, creates a UNIX or Windows directory where multiple data files are stored. Each data file is a UNIX or Windows file or subdirectory in that directory, depending on whether the data file is static hashed, dynamic hashed, nonhashed, or B-tree.

All UniVerse commands recognize multiple data files. They are specified on the command line using the same syntax as is used on other Pick systems:

filename,data.file.name

filename is the name of the UNIX or Windows directory that contains multiple data files, and *data.file.name* is the name of one of the data files. A comma separates the file name from the data file name, and no blank spaces can follow the comma.

Any UniVerse command that accepts a file name works with the above syntax for multiple data files.

filename does not refer to the name of a file dictionary as it does on other Pick systems; it refers to the group of associated data files that share a single file dictionary. *filename* is really the name of a VOC entry which points to the UNIX directory containing the multiple data files.

Unlike Pick, a file comprising multiple data files is defined in a file definition entry in the VOC file, UniVerse does not use D-pointers. The file-defining entry contains an “M” (Multiple data files) in field 4. Field 2 of the entry specifies the path of the UNIX or Windows directory that contains the multiple data files. Fields 7 and 8 are associated multivalued fields. Field 7 contains the UniVerse data file names in sorted order, and field 8 contains the corresponding UNIX or Windows file names.

The following sample VOC file definition entry for a UniVerse file named STOCK has multiple data files, STOCK, ON.ORDER, and ON.HAND:

```
          STOCK
001:  F
002:  STOCK
003:  D_STOCK
004:  M
005:
006:
007:  ON.HANDVMON.ORDERVMSTOCK
008:  ON.HANDVMON.ORDERVMSTOCK
```

STOCK in field 2 is not the name of the data file STOCK: it is the relative path of the UNIX or Windows directory that contains the data files, in this instance, the files ON.HAND, ON.ORDER, and STOCK. To create this VOC entry, each data file was created with a separate CREATE.FILE command.

Note that the record ID of this VOC file entry is also STOCK. On a Pick system this entry would point to the file dictionary called STOCK, which in turn would contain D-pointers to each separate data file. In UniVerse, however, the entry points to all the related files: the file dictionary (D_STOCK, in field 3) and to the data files (in field 7). ON.HAND, ON.ORDER, and STOCK do not have separate file definition entries. All three data files share the same file dictionary, D_STOCK.

File pointers

Unlike Pick, UniVerse has two types of file pointer that you can use to point to files in other accounts: remote file pointers and Q-pointers. A remote file pointer uses a UNIX path to point to a file in another account. Remote file pointers are standard UniVerse VOC file entries with an F in field 1 (see Chapter 3 for information about F-type VOC entries). They should not be confused with Remote Command items which have an R in field 1. To list the remote file pointers in your account, use the LISTFR command.

A Pick-style Q-pointer also points to a remote file, but it does so through the VOC file of another account, as on other Pick systems. The Q-pointer is more portable than a remote file pointer because it points indirectly to the remote file. If the account moves in the system-level file structure, the Q-pointer still points to it correctly, provided that the system administration files are correctly updated. Using Q-pointers to gain access to a file is slightly slower than using remote file entries since the VOC file needs to be located before the files themselves are searched.

Use the SETFILE command to create remote file pointers and the SET.FILE and SET-FILE commands to create Pick-style Q-pointers.

SETFILE command

The SETFILE command creates a VOC file entry that points to an existing data file either in the local or in a remote account. The syntax for the SETFILE command is:

SETFILE [*pathname*] [*filename*] [**OVERWRITING**]

pathname is the full path of the data file for which you are creating an entry.

filename is the name used to gain access to the file from the current account.

To point to a file in the same account, enter the name of the file as the path. To point to a file in a remote account, enter the relative or full path of the remote file.

The OVERWRITING option replaces an existing VOC entry with the same name as *filename*.

This is the quickest way to gain access to a data file in another UniVerse account. If the file is moved to another directory, however, you can no longer gain access to the file, because its path has changed.

The SET.FILE command

Use SET.FILE to create a permanent Q-pointer in your VOC file to a remote file. The syntax is

SET.FILE [*account*] [*filename*] [*pointer*]

account is the name of the UniVerse account containing the remote file.

account can be specified as any of the following:

- Name of the UniVerse account as defined in the UV.ACCOUNT file. (This is the preferred way to specify the account.)
- UNIX or Windows path of the directory where the remote account is located. (If you use the UNIX or Windows path of the account, the Q-pointer loses its portability.)
- Name of a UNIX or Windows login account.

filename is the name of a file in *account*.

pointer is the record ID of the Q-pointer in your VOC file. If you do not specify the qualifiers, the SET.FILE command prompts for them. By default, UniVerse's SET.FILE command names the Q-pointer and that Q-pointer is permanent; it does not create a record called QFILE that is updated each time SET.FILE is invoked.

If you want SET.FILE to create a Q-pointer whose record ID is QFILE, omit the pointer name in the command line. At the name prompt, press the ENTER key. For example:

```
>SET.FILE PERSONNEL PAYROLL
Q name: <ENTER>
Q-pointer written to VOC file.
>
```

By changing the SET.FILE program, your System Administrator can suppress the "Q name:" prompt. This is done by changing the value of the variable PROMPT.FOR.NAME from 1 to 0, then recompiling the program. This change makes the SET.FILE command work as it does on other Pick systems, creating a QFILE item by default. The SET.FILE program is in the BP file located in the UV account.

File protection

Since UniVerse is a multiuser system, the file handler manages several different kinds of lock which prevent problems caused by more than one user trying to gain access to the same data at the same time. There are two different categories of lock:

- Explicit locks, which are under the control of the user.
- Implicit locks, which are managed by the UniVerse file handler without direction from the user.

Explicit locks

The two kinds of explicit lock are file locks and record locks. Record locks are also known as item locks and READU locks. Explicit locks can be set by various UniVerse BASIC statements such as FILELOCK and READU. They can also be set by various processors such as the Editor. An explicit lock prevents other users from setting the same lock or from writing to the locked file or record. Except for the exclusive use file lock used by the RESIZE command, an explicit lock does not prevent other users from reading the locked file or record. BASIC statements that attempt to set a lock or to write to a locked file can either wait for a lock to be released, or they can specify a series of statements to be executed if a lock is encountered. Subsequent attempts to set an explicit lock by the same user are successful, as are writes to a locked file or record by the same user.

A file lock is set with the FILELOCK statement in a UniVerse BASIC program. A file lock prevents other users from setting a file lock on the locked file, or from setting a record lock on any record in the locked file, or from writing any record in the locked file. A file lock is released by a FILEUNLOCK statement or by closing the file. A special exclusive use file lock is set by the RESIZE command. This file lock prevents any access to the file.

A record lock can be set with a READU, READVU, or MATREADU statement. A record lock prevents other users from setting a file lock on the file containing the locked record, or from setting a record lock on the locked record, or from writing the locked record. A user can write to a locked record without releasing the lock by using the WRITEU, WRITEVU, or MATWRITEU statements.

A record lock is released by a `RELEASE`, `WRITE`, `WRITEV`, or `MATWRITE` statement, or by closing the file. Record locks can also be released by the `RELEASE` and `UNLOCK` commands issued at the `TCL` prompt. The `RELEASE` command unlocks records locked by `READU`, `READVU`, and `MATREADU` statements; the `UNLOCK` command clears group, file, and record locks. These commands are similar to the `CLEAR-GROUP-LOCKS`, `CLEAR-ITEM-LOCKS`, and so forth, found on other Pick systems.

A file lock or any record locks set in a file are also released when the program that set the locks terminates, provided that the file variable used to set the locks is not in a named `COMMON`. If the file variable is in a named `COMMON`, the locks are preserved until explicitly released. All locks are released when the user exits the UniVerse environment. See Chapter 8, [“Chapter 8: UniVerse BASIC,”](#) for details about named `COMMON`.

Use the `LIST.READU` command to list all active group, file, and record locks. `LIST.READU` is similar to the `LIST-LOCKS`, `LIST-ITEM-LOCKS`, `LIST-GROUP-LOCKS`, and other such commands found on other Pick systems.

Implicit locks

An implicit lock is set by the UniVerse file handler on a group in a file in order to preserve the integrity of the links in that group. There are three types of implicit group locks: read, write, and informational.

A read lock is set any time any record in the group is being read. A read lock causes an attempt to write any record in the group to wait until all the reads have completed; other attempts to read a record in the group are allowed.

A write lock is set any time any record in the group is being written. A write lock prevents any other access, either read or write, to any record in the group.

An informational lock is set any time any record in the group has a record lock set. An informational lock is used during a write to reduce the overhead needed to check if the record being written has a record lock.

UNIX file protection

Permission to gain access to a user’s files is not controlled by retrieval and update codes specified in the D-pointer to a file or its dictionary, as it is on generic Pick systems. File permissions in UniVerse are a function of UNIX and are set by the file creation mask *umask*, specified either in the user’s *profile* (or *login*) file or in a UniVerse account’s login paragraph.

umask specifies permission bits using an octal value determined by adding together any of the following values:

Value	Description
400	no read by owner
200	no write by owner
100	no execute (search in directory) by owner
040	no read by group
020	no write by group
010	no execute (search in directory) by group
004	no read by others
002	no write by others
001	no execute (search in directory) by others

umask Values

These values can be combined. For example, consider the following values:

077	no read, write, execute (search) by group or other
133	no execute by owner, no write or execute by group or other

umask Examples

The *umask* command takes a value that specifies which bits of permission should be turned *off*. For example, the following command turns off bit 2, disabling write permission for users other than the owner or the owner’s group for any files created after the *umask* command is executed.

```
umask 2
```

The command below makes any files created after the *umask* command is executed accessible only to the owner.

```
umask 77
```

The UniVerse UMASK command performs the same function as the UNIX *umask* command.

The permission values for given files can be shown using the UNIX *ls* command with the *-l* option. The information displays in the following format:

```
-rwxrwxrwx
```

The first position shows the file type, as follows:

File Type	Description
-	regular file
d	directory
c	character special file (<i>/dev</i>)
b	block special file (<i>/dev</i>)
l	symbolic link
p	named pipe
File Types	

The default *umask* is set to 022 (octal) so that only the owner can write to the file but all users can read it.

In an open environment you might initialize this mask with a value of 0 (octal) so that files can be accessed by all users. Alternatively, you might set the mask with a value of 7 so that only users in the same group can have access to files. Likewise, you could manipulate this mask to give full access to the owner, read-only access to the group, and no access to outsiders, or to define any other combination of access.

In addition, the owner of a file can change the permission modes for a file using the UNIX *chmod* command. Unlike *umask*, *chmod* uses the octal mask to specify which bits in the mask should be turned *on*. For example, the following command gives read and write permission for *file* to the owner, the group and all other users.

```
chmod 666 file
```

chmod also supports symbolic arguments. For example the following adds execute permission for *file* for all users.

```
chmod +x file
```

See *chmod* in the *UNIX Programmer's Manual* for additional details. There is no UniVerse equivalent to *chmod*.

Maintaining files

This section introduces a set of UniVerse commands that are used to maintain efficient file structures for static hashed files. UniVerse provides two commands that will be familiar to most Pick users: GROUP.STAT (like the ISTAT command on other Pick systems) and HASH.TEST. In addition to these commands, UniVerse provides several other commands that greatly extend and enhance your ability to obtain file statistics and to test hypothetical file sizing parameters. This section tells how to repair damaged files, restore unused space, and clean up an account. For details on maintaining efficient file structures, see *UniVerse System Description* and *UniVerse System Administration*.

GROUP.STAT and HASH.TEST commands

GROUP.STAT is the UniVerse equivalent of ISTAT. It displays information about the actual distribution of records in each group. HASH.TEST takes a hypothetical modulo and separation specified by the user and produces a report similar in format to the GROUP.STAT report, showing how records would be distributed if the hypothetical modulo and separation were used. It does not make any changes to the existing file. Both GROUP.STAT and HASH.TEST display a histogram showing how evenly records are (or would be) distributed in each group.

Record distribution details

GROUP.STAT.DETAIL and RECORD are two other UniVerse commands that work like familiar Pick commands. The GROUP.STAT.DETAIL command is like the GROUP command in Pick: it lists detailed statistics for every record in each group. The RECORD command is like the Pick ITEM command: it lists the number of the group a specified record ID hashes to and indicates whether or not the record ID really exists in the file. The HASH.TEST.DETAIL command works like HASH.TEST: it shows how record IDs would be distributed if a hypothetical modulo and separation were used.

More file statistics

The following commands produce file statistics reports in a format different from those described in the previous section.

The FILE.STAT and HASH.AID commands

The FILE.STAT command is like ISTAT. It lists the following information about a file:

- Current file type
- Number of records and bytes in the file
- Average number of records per group, bytes per record, and fields per record
- Maximum and minimum numbers of records per group, bytes per record, and fields per record
- Number of groups that are a given percent full

The HASH.AID command produces the same statistics as the FILE.STAT report, but on a hypothetical basis. The user specifies a hypothetical file type, modulo, and separation.

The HASH.HELP and HASH.HELP.DETAIL commands

Once you have determined that a file needs restructuring, use the HASH.HELP and HASH.HELP.DETAIL commands to recommend the most efficient file type, modulo, and separation to use when you resize the file. Starting with the values recommended by HASH.HELP and HASH.HELP.DETAIL, try out hypothetical file parameters using the HASH.TEST and HASH.AID commands. If the results leave the record distribution unbalanced or the groups too full, try other values with HASH.TEST and HASH.AID.

Remember that ideally the modulo should be a prime number. Use the PRIME command to determine the prime numbers closest to a given number if you have calculated the approximate number of groups you want to have in your file.

The FILE.USAGE command

The FILE.USAGE command lists file usage statistics for hashed files. It is useful for discovering whether a file size that is efficient in terms of disk space is also acceptable in terms of system performance. FILE.USAGE will tell you if the system is experiencing performance degradation due to excessive overflow block accesses. If the number of oversized block accesses (for read or write) is low, the performance impact of the file size is negligible.

Maintaining dynamic files

Two commands that are useful for analyzing and reconfiguring dynamic hashed files are ANALYZE.FILE and CONFIGURE.FILE. The ANALYZE.FILE command lists the

- hashing algorithm
- modulo and minimum modulus
- large record size
- group size
- split load, merge load, and current load
- number of secondary indices
- total file size

Use the CONFIGURE.FILE command to change the parameters of a dynamic file. Use the RESIZE command to change a static hashed file into a dynamic hashed file.

Resizing files

Unlike most Pick systems, UniVerse features a command which lets you resize (reallocate) a file. Once you have determined the optimal file type, modulo, and separation, you can change these values for the file by using the RESIZE command. The syntax for RESIZE is

```
RESIZE [ DICT ] [ filename ] [ type ] [ modulo ] [ separation ]  
[ CONCURRENT | INPLACE | USING partition ]
```

If you want to retain the existing file type, modulo, or separation, type * as its value. For example, to change the VOC file's modulo to 2 but leave the file type and separation the same, type:

```
>RESIZE VOC * 2 *
```

When RESIZE changes the file structure, it really changes the physical structure of the file on the disk. Normally, RESIZE creates a file of the specified size. Records are then copied from the original file to the new file. The file is locked to prevent other users from accessing it while it is being resized.

You can let other users access a file while it is being resized by using the CONCURRENT option. If you do not have enough disk space to create the new file, the file can be resized in place using the INPLACE option. When the INPLACE option is specified, the file is resized in the same physical location on the disk.



***Note:** When you use the INPLACE option to decrease the size of a UniVerse file, the size of the system-level file does not shrink; instead, the unused space is moved to the end of the file. In some cases the size of the system-level file may increase because of work space allocated by RESIZE.*

Changing a static file to a dynamic file

You can convert a static file to a dynamic file using the RESIZE command. The RESIZE command uses the same syntax for this conversion as it does when converting between hashed types. Specify a type of 30 or the keyword DYNAMIC to convert the file to a dynamic file. RESIZE takes the name of the file from the command line or from an active select list and changes the file to a dynamic file with the default parameters. You can specify changes to all dynamic file parameters on the command line.

RESIZE cannot change the parameters on an existing dynamic file. Use the CONFIGURE.FILE command to change an existing dynamic file's parameters.

Repairing damaged files

Occasionally a system crash or other unexpected event damages a file. When this happens, you may not be able to gain access to the file. It might be possible, however, to repair the damage by using the RESIZE command. Enter the following command to attempt to repair the file:

```
>RESIZE filename * * *
```

The asterisks keep the file type, modulo, and separation the same.

Cleaning up an account

In addition to monitoring the structure of UniVerse files, it is also important to monitor the structure of your account for file structure problems and for old temporary files. UniVerse, unlike most Pick systems, provides a CLEAN.ACCOUNT command for this purpose. CLEAN.ACCOUNT performs routine maintenance activities on your account and can correct suspected problems with files. If you suspect your account has problems, notify your System Administrator before using CLEAN.ACCOUNT. If you do use this command, use it with care.

CLEAN.ACCOUNT searches for special temporary files such as &TEMP& and &PH&. It deletes the &TEMP& file and clears the &PH& file if you give it permission. It also searches the &SAVEDLISTS& file for temporary records and deletes them. For more information about any of these files, see *UniVerse System Description*. After examining the special files, CLEAN.ACCOUNT examines all the files defined in the VOC file. The names of any files with problems are listed on your terminal.

Chapter 5: UniVerse file dictionaries

UniVerse dictionary entries.	5-3
UniVerse dictionary fields.	5-4
Pick attribute conversion	5-7
More about dictionary records	5-9
I-descriptors	5-11
Defining I-descriptors	5-11
I-descriptor expressions	5-12
Compound expressions	5-15
Correlative and conversion codes	5-17
Multivalued data conversions	5-17
Phrases	5-21
Special phrases	5-21

A basic feature of any Pick system is its use of dictionaries that define data stored in separate data files. The UniVerse dictionary structure is different from the dictionary structure on generic Pick systems. This chapter describes the differences between UniVerse and Pick dictionaries.

UniVerse dictionary entries

Every UniVerse file normally comprises at least one data file and an associated file dictionary, although a file can be created that is only a dictionary or only a data file. You can also have a file that contains both dictionary entries and data records. Multiple data files can be associated with the same file dictionary.

As on other Pick systems, dictionaries contain entries that define data fields and interpretive fields. Data files, on the other hand, contain records that store data values in fields.

The structure of entries in a UniVerse file dictionary is a superset of the structure of items in a Pick dictionary. In many ways the UniVerse dictionary is simpler to use and more powerful, but you can continue to use most dictionary items exactly as you did on your Pick system.

There are five main types of UniVerse dictionary entry:

- **@ID** – The “@ID” entry specifies the output format for record IDs of records in the data file. On Pick systems this function is performed by the D-pointer to the data file.
- **Data Descriptors** – Entries that define fields for storing physical data are called data descriptors in UniVerse. They are the equivalent of Attribute Definition items on Pick systems. They specify the format and location of fields for records in the data file.
- **I-Descriptors** – Entries that define virtual fields are called I-descriptors in UniVerse (the “I” stands for “interpretive”). They specify the format of fields whose values are derived from other fields. I-descriptors are analogous to Pick Attribute Definition items that derive values from fields using correlative codes.
- **Phrases** – Entries can contain a *phrase*. A phrase is any part of a Retrieve sentence that does not include a verb. Most phrases are user-defined. They are often used to specify fields selected for output.
- **X-Descriptors** – Entries can contain other user-defined information, such as the next available record ID for the data file, or a constant used in several different I-descriptors.

You can convert each item in a Pick dictionary to one of these UniVerse entry types, or you can leave your “A” or “S” items in their original form. If you want to convert your Pick items into UniVerse dictionary entries to take advantage of UniVerse’s additional functionality and simpler BASIC syntax, use the DC command.

Generally speaking, if the Pick dictionary item has a D/CODE of “A” or “S” and does not have a correlative code in attribute 8, it is converted to a data descriptor. If the Pick dictionary item contains a correlative code in attribute 8, it is converted to an

I-descriptor. Data records that are stored in a dictionary have no special format and are copied from the Pick dictionary without any conversion.

Data descriptors and I-descriptors can be placed in the VOC file of an account as well as in file dictionaries. Putting descriptors in the VOC file makes these definitions available to every file in the account. For example, the standard VOC file has data descriptors defined for F1 through F10, to define the first ten fields of any file.

UniVerse dictionary fields

UniVerse dictionary fields look different than those on a Pick system. Like any record in a UniVerse file, each dictionary entry has a unique key or record ID. The record ID for a data descriptor or an I-descriptor specifies the name of a field, actual or virtual, in the data file. The following tables show the UniVerse dictionary fields, and their Pick equivalents.

The following table shows the contents of the first seven fields of a UniVerse dictionary entry.

Field	Field Name	Description	Pick Equivalent
1	Type & Description	Contains a type code and optional description of the field. Type codes can be: D for data descriptor I for interpretive descriptor PH for phrase X for user-defined information	A, S, or X
2	Field Definition	Contains different information depending on the type of dictionary entry. D-type field number I-type expression PH-type user-defined phrase	AMC Correlative code
3	Conversion Code	(Optional) A formula for converting data stored in internal format into external format.	
4	Column Heading	(Optional) A descriptive name used to label the field in a report. If column heading is not specified, the record ID of the dictionary entry is used.	S/NAME or TAG
UniVerse Dictionary Fields			

Field	Field Name	Description	Pick Equivalent
5	Output Format	The column width used when listing the field in a report, and the column justification: R right-justified (for numeric sorting) L left-justified (for alphabetic sorting) T wrapped text (text breaks between words)	V/TYP or V/MAX
6	Single/Multi	(<i>Optional</i>) Can be one of the following: S for single-valued fields (the default) M for multivalued fields	
7	Association	(<i>Optional</i>) The name of a phrase (defined by another entry in the dictionary) that links multivalued fields in an "association"	Controlling and dependent attributes

UniVerse Dictionary Fields (Continued)

Each dictionary has an @ID record that describes the record ID field of the data file. This record is created automatically by the CREATE.FILE verb when a UniVerse file is created. The @ID record is assigned a field definition of 0.

In the dictionary conversion process, a Pick dictionary item that has a D/CODE of D and an item ID that matches the name of the file is converted into the @ID record. On a REALITY system, an item with a D/CODE of D and an item ID of DL/ID is converted into the @ID. If none of these items is found, a default @ID record is automatically created.

Pick attribute conversion

This section describes the conversion of attributes from a Pick dictionary item to their equivalent fields in a UniVerse dictionary entry. It is meant to give only a general indication of the UniVerse equivalent to a Pick attribute.

Attribute	Field Name	Description
0	Item ID	The item ID is converted to the record ID.
1	D/CODE	<p>Most values in the D/CODE attribute are converted to a value in the TYPE field. "A" and "S" codes usually become "D" (for data descriptor). If a Pick item has a D/CODE of A or S and does not have a correlative code in attribute 8, it is implemented as a data descriptor with a Field Definition specified by the value in the A/AMC attribute.</p> <p>If the D/CODE is X, an X-descriptor record is created, and the remaining fields are copied without any further conversion. Any item that has a D/CODE other than A, S, or X is copied verbatim without conversion.</p>
2	A/AMC	The Attribute Mark Count is converted to a Field Definition (field 2) for data descriptor fields.
3	S/NAME	The S/NAME or TAG attribute is converted to the Column Heading (field 4).
4	S/AMC	The Controlling and Dependent attribute describes a relationship between values in multivalued fields. It is implemented as an Association in UniVerse. An Association defines a relationship between two or more multivalued fields. Associated multivalued fields in UniVerse are not related hierarchically; that is, no one field "controls" all the other "dependent" fields, as is the case on other Pick systems.
7	Conversions	Most conversion codes familiar to the Pick programmer are implemented in UniVerse. Many have been added for Pick compatibility. A complete list of correlative and conversion codes is provided in Correlative and conversion codes.

Pick Dictionary Fields

Attribute	Field Name	Description
8	Correlatives	Correlatives are implemented as I-descriptor expressions (see I-descriptors). I-descriptor expressions are actually lines of BASIC code; they should not be confused with Pick correlative codes.
9	Justification	The V/TYP attribute is converted into the justification specified in the Output Format field (field 5).
10	Column Width	The V/MAX attribute is converted into a column width specified in the Output Format field (field 5).

Pick Dictionary Fields (Continued)

UniVerse file dictionaries accept entries in the standard Pick 10-line format, provided that the D/CODE is A, S, or X. You can mix UniVerse and Pick dictionary entries in any file dictionary but you cannot mix UniVerse and Pick formats within the same dictionary entry. For example, you cannot use a type code of A or S in an entry that uses the 7-line UniVerse dictionary format.

More about dictionary records

The following sections discuss in more detail the format of records in a UniVerse dictionary. You can create dictionary entries using the UniVerse Editor or REVISE.

Field	Field Name (REVISE)	Description
0	Record ID	A name that identifies the field.
1	Type and Description(CODE)	Contains the entry's type code and optional description. D -Data descriptor I -I-descriptor PH-Phrase X -User defined information
2	Field Definition (D-LOC) (I-EXP) (PH and X-FUNC)	D-Location, or relative position, of the field in the data file defined by this entry. I-Expression to be interpreted to obtain the field value. PH-Elements that make up the phrase (field names).
3	Conversion Code (optional) (CONV)	D and I-A formula for converting data stored in internal format into external format.
4	Column Heading (optional) (NAME)	D and I-A descriptive name used as a column heading. If not specified the record ID is used.
5	Output Format (FORMAT)	D and I-The width and justification of the column used when listed in a RETRIEVE report. This field combines attributes 7, 9 and 10 of the Pick dictionary.

Dictionary Field Definitions for D Entries

Field	Field Name (REVISE)	Description
6	Single or Multiple Value (SM)	D and I-Single- (default) or multivalued field
7	Association (ASSOC)	The name of a phrase (defined by another entry in the dictionary) that links multivalued fields in an association.
8	SQL type (SQLTYPE)	The data type of this field, used by SQL statements
Dictionary Field Definitions for D Entries (Continued)		

I-descriptors

I-descriptors define interpretive fields that are evaluated when a Retrieve sentence is processed. They are equivalent to (but different from) Pick dictionary items containing correlative codes in attribute 8. These fields do not physically exist in the records of the data file. The value of an interpretive field is calculated using values in other fields in the record or using values in other files. I-descriptors are part of the file dictionary and let you use information from other fields and other files as if it existed in the data file.

I-descriptors are processed before other Retrieve processing (selection, sorting, and so forth) and behave similarly to Pick correlative codes. However, their syntaxes are different.

Defining I-descriptors

To create an I-descriptor in the dictionary file, specify “I” in field 1 and define the I-descriptor expression in field 2. The expression can be an arithmetic, logical, or string expression, BASIC function, external subroutine call, or a file translation function. In fact, almost any fragment of BASIC code can be specified. Pick users should note, however, that correlative codes as they are used on other Pick systems cannot be specified as I-descriptor expressions. Fields 3 through 7 are defined in the same way as when you are defining data descriptors.

Compiling I-descriptors

Before you use an I-descriptor in a Retrieve sentence, you should compile the expression to verify that its logic and syntax are accurate. Also, you should compile all the I-descriptors in a dictionary any time you change any D-type or I-type entry since another I-type may reference the changed entry. If you have not compiled an I-type before you include it in a sentence, Retrieve compiles it the first time it is used in a command.

Use the `COMPILE.DICT` command (or its abbreviation `CD`) to compile I-descriptors. When an I-descriptor is compiled, the format of the I-descriptor record changes. It includes extra fields that contain the compiled object code, and the time and date of the compilation. Depending on your terminal type, it may not be possible to display object code on the screen using commands such as `CT`, `LIST.ITEM`, or `ED`. However, you can display items containing object code if you use the Editor in up-arrow mode. To enter up-arrow mode in the Editor, enter a caret (`^`) at the Editor prompt.

I-descriptor expressions

An I-descriptor expression can contain any of the following:

- Field names of D-descriptors and I-descriptors
- Arithmetic, relational, logical, and conditional operators
- Numeric and string constants
- Internal variables called @-variables
- Substring extraction expressions
- `TRANS` function for file translation
- `TOTAL` function for accumulating totals
- BASIC functions and subroutines

Field names

An I-descriptor can refer to another field by specifying its field name. This field name must identify a data descriptor or I-descriptor that is defined in the dictionary when the I-descriptor is compiled. When the expression is evaluated, the value of the named field in the current data file record is used.

Before using one I-descriptor in a second I-descriptor, be sure that the second I-descriptor does not reference the first I-descriptor. UniVerse cannot evaluate such an I-descriptor.

Conditional expressions

You can use IF/THEN/ELSE expressions in I-descriptors. The syntax is as follows:

IF *expression1* THEN *expression2* ELSE *expression3*

If *expression1* is true, the value of the conditional expression is the value of *expression2*. If *expression1* is false, the value of the conditional expression is the value of *expression3*. Both THEN and ELSE expressions are required by the syntax.

The TRANS function

The TRANS function is similar to the *Tfile* correlative on other Pick systems, and to the *Tfile* conversion code in UniVerse. It is used to gain access to data in other files. The syntax of the TRANS function is as follows:

TRANS(*filename*, *record.IDs*, *field*, "code")

filename is the name of the remote file from which data is to be translated. The file name must be in your VOC file when the expression is compiled and when it is used.

record.IDs is an expression that evaluates to the record IDs of records in the remote file from which data is to be translated. *record.IDs* can be the name of a field in the current file that contains either record IDs for the records in the remote file or expressions that evaluate to record IDs. If *field* is multivalued, subvalues are returned for each record ID. If *record.IDs* is multivalued, the translation occurs for each record ID and the result is multivalued (that is, values are returned rather than subvalues). If both *record.IDs* and *field* are multivalued, subvalues are returned for each record ID.

field can be the name or number of one field in the remote file, or the @RECORD variable or its equivalent, -1. @RECORD indicates that the entire record in the remote file should be returned. When *field* is multivalued, all values are returned as subvalues.

code is a control code that tells TRANS what to do if the translation is not successful. The code must be enclosed in quotation marks. The code can be:

- C – returns the original record ID if the value in *field* is null or if a specified record does not exist.

- V -- verifies successful translation. If a record does not exist or if the value in *field* is null, a message is printed and a null value is returned.
- X -- returns a null value if a record does not exist or if the value in *field* is null.

The TOTAL function and the CALC keyword

The TOTAL function can be used in an I-descriptor to accumulate and print subtotal and total information in a Retrieve command report. The TOTAL function is not a UniVerse BASIC function; it can be used only in I-descriptors.

To use the TOTAL function, do one of the following:

- Create an I-descriptor that uses the TOTAL function for every field or expression for which you want to accumulate totals.
- Include the I-descriptor field preceded by the CALC keyword as one of the output specifications in a Retrieve command.

The syntax of the TOTAL function is as follows:

TOTAL(*expression*)

For example, an I-descriptor called PROFIT contains the following expression:

TOTAL((PRICE-COST)*QTY)

The I-descriptor PROFIT, preceded by the CALC keyword, is then used as one of the output specifications in the following RETRIEVE command:

LIST SALES PART.NO PRICE COST CALC PROFIT

For each line item in the report, the column displaying the PROFIT field contains values as if the PROFIT I-descriptor expression were

(PRICE-COST)*QTY

Retrieve keeps an accumulator for the values of (PRICE-COST)*QTY. At the end of the report, the LIST command prints an additional report line with the accumulated total in the PROFIT column.

If you do not use the CALC keyword, Retrieve ignores the TOTAL function in the

I-descriptor. If you use the CALC keyword but the I-descriptor does not contain a TOTAL function, Retrieve produces no totals.

You can use the BREAK.ON keyword in a Retrieve sentence such as the one in the preceding example. In this case Retrieve keeps an accumulator for both subtotals and totals for each expression in the TOTAL function. Subtotals are printed at each breakpoint, and totals are printed at the end of the report.

Retrieve keeps as many accumulators as are necessary to produce the proper subtotals and totals for the fields that are modified with the CALC keyword. For example, an I-descriptor called MARGIN might look like this:

$$(\text{TOTAL}(\text{PRICE}-\text{COST})/\text{TOTAL}(\text{COST}))*100$$

RETRIEVE accumulates the PRICE \- COST and COST values in separate accumulators and produces a subtotal or total value by dividing the PRICE \- COST accumulator by the COST accumulator and then multiplying that result by 100.

Although in many cases the TOTAL function works similarly to the TOTAL keyword in Retrieve sentences, it should not be confused with it. You must use the TOTAL function with the CALC keyword in cases where you wish to accumulate only part of an expression in an I-type containing the operators *, /, **, and ^. The example of the MARGIN I-descriptor could not be replaced with the TOTAL keyword.

Other Retrieve keywords that can be used when evaluating numeric fields are AVERAGE, BREAK.ON, GRAND.TOTAL, and PERCENT. They are described in detail in the *UniVerse User Reference*.

Compound expressions

A compound expression comprises two or more simple expressions separated by semicolons. Compound expressions can be used to improve the readability of expressions. The value of the compound expression is the value of the last simple expression in the list.

To reference the simple expressions in the compound expression, use @-variables:

- @ references the value of the previous expression.

- @*n* references the specific expression in the compound expression. The simple expressions are numbered from left to right, starting with 1.

The following example sums the values in the SALARY and BENEFITS fields, then checks the employee type. If the employee is exempt, the value of the I-descriptor is the sum of the salary and benefits, otherwise the value is 0.

```
SUM(SALARY + BENEFITS); IF EMPL.TYPE EQ "EXEMPT" THEN @1 ELSE 0
```

You cannot nest compound expressions, because the dictionary compiler is unable to properly evaluate such an expression. This means that a compound expression cannot contain the name of an I-descriptor that is another compound expression.

Correlative and conversion codes

UniVerse supports virtually all of the processing codes used as correlatives and conversions on other Pick systems. But since UniVerse has replaced the function of the Pick correlative code with the much more powerful I-descriptor expression, there is no need to use processing codes as correlatives unless you want to. Conversion codes in UniVerse are just like the ones on other Pick systems. In addition, most UniVerse conversion codes can be used as correlatives.

The UniVerse correlative and conversion codes can be used in any of the following places:

- As a conversion in field 3 of UniVerse dictionary entries (both data descriptors and I-descriptors).
- As a conversion in field 7 of a Pick-style dictionary entry.
- As a correlative in field 8 of a Pick-style dictionary entry.
- In the input and output conversion functions of UniVerse BASIC (ICONV, ICONVS, OCONV, OCONVS).
- In input and output conversion functions used in an I-descriptor expression.

Pick users should note that the A and F codes can be used as conversions as well as in the ICONV and OCONV functions of BASIC. For details about any of the code, refer to *UniVerse BASIC* or the *UniVerse User Reference*.

Multivalued data conversions

On most Pick systems, when conversion codes are used in BASIC programs, they function only on the first value or subvalue of a multivalued field. Thus if a Pick user writes the following in BASIC where B is multivalued, only the first value gets converted, and A gets assigned the singlevalued conversion result.

```
A = OCONV (B, "MD2 " )
```

The Pick retrieval language processors, on the other hand, call the conversion routines iteratively for each value and subvalue in the data, regardless of whether the conversion code is specified in the conversion field (field 7) or the correlative field (field 8). Thus if a user has the following two Pick dictionary items:

ID:	X	Y
001:	A	A
002:	3	3
003:	Fred	Ethel
004:		
005:		
006:		
007:	MD2	
008:		MD2
009:	L	L
010:	6	6

and data item RICKY has 256]47801 in the third attribute, the sentence

```
LIST filename 'RICKY' X Y
```

produces this result:

ID	Fred	Ethel
Ricky	2.56	2.56
	478.01	478.01

There are exceptions to these general rules: some conversion codes available to the ACCESS processor are not implemented in Pick BASIC; sometimes the entire data string is passed to the conversion routine, which then treats delimiters as plain ASCII characters; ML and MR may behave differently depending on whether you specify a precision or merely a pattern.

UniVerse BASIC provides two conversion functions, OCONV and OCONVS (as well as the OCONVS subroutine). OCONV is used on singlevalued fields and OCONVS is used on multivalued fields. Further, UniVerse BASIC supports *all* conversion codes that are valid in UniVerse dictionaries. (Some of them are more readable using simple BASIC statements. For instance, the statement

```
A = (REC<3> + REC<4>) / REC<5>
```

is cleaner than

```
A = OCONVS(@ID, "A; (3 + 4) / 5")
```

but UniVerse supports both statements.

If a conversion code is specified in field 3 of a UniVerse dictionary, the conversion routines are called iteratively for each data value and subvalue. If a conversion code is specified in field 2 of an I-descriptor using OCONV, the conversion routine is called once and operates on the entire data string, delimiters and all. A conversion routine named in field 2 of an I-descriptor using OCONVS is called iteratively and operates on successive data values and subvalues. The sole exception to this rule is the *Tfile* code, which does a lookup on each separate value and subvalue regardless of where or how it is called.

Some conversions appear to operate identically, whether called by OCONV or OCONVS, but in fact they do not. For instance, the statement below passes first *abc* and then *def* to the MCU conversion routine and puts the separate converted values (*ABC* and *DEF*) into A separated by a value mark:

```
A = OCONVS("abcVMdef", "MCU")
```

However the following statement passes the entire string *abc**VM**def* to the MCU conversion routine, which converts *a*, *b*, and *c* to uppercase, leaves the value mark alone, and converts *d*, *e*, and *f* to uppercase, returning the single converted string into A.

```
A = OCONV("abcVMdef", "MCU")
```

The results are identical, but the processes are different.

To produce UniVerse dictionary items that behave like their Pick equivalents, the dictionary converter (DC) turns all correlatives, with the exception of the F code, into I-descriptors that use the multivalued OCONVS conversion. The F code routine is fundamentally different from most other conversions. Most conversions convert data passed to them as an argument. F correlatives, on the other hand, name within themselves all the data to be used in the calculation—they do not really convert data passed to them as an argument. So the dictionary converter turns F correlatives into I-descriptors employing the singlevalued OCONV conversion. To use OCONVS would be to derive the same result over and over for each separate value or subvalue being “converted” by the F expression.

If you want a conversion to operate only on the first value of a multivalued string you must make a modification. You would change the Pick BASIC statement

```
A = OCONV(B, "MD2")
```

to its UniVerse equivalent

```
A = OCONV (B<1,1>, "MD2" )
```

to get the same singlevalued result. The UniVerse form transparently ports back to the Pick machine.

Phrases

A phrase is an incomplete Retrieve sentence that can contain any part of a sentence except the verb. Phrases are used to specify the fields in an association. Phrases are also used to define output specifications or to store often-used selection or sort expressions.

To define a phrase in the file dictionary, specify PH in field 1 and enter the phrase in field 2, the Field Description field. Use the remaining fields to continue the phrase if it is a long one.

To use a phrase in a RETRIEVE sentence, include the name of the phrase in the sentence. Here is an example of the dictionary of a PARTS file that contains the phrase:

```
          BOM
0001 PH
0002 BOM.PART BOM.DESC BOM.PRICE
```

You can get a report containing the part number, description, and price by entering:

```
LIST PARTS BOM
```

You can put phrases in the VOC file (see Chapter 3). This makes the phrase available to all files in the account.

Special phrases

Retrieve commands recognize three phrases as having special meaning: the @ phrase, the @LPTR phrase, and the @REVISE phrase.

The @ phrase defines default output. You can also put default sort or selection expressions in the @ phrase. If no output specification is included in a Retrieve command, the fields specified in the @ phrase are listed by default. If neither the file dictionary nor the VOC file contains an @ phrase, only record IDs are listed when a Retrieve command is used that does not include an output specification.

Create the @ phrase as you would any other phrase.

The dictionary converter converts the Pick default output specifications to a UniVerse default output specification as follows. The DC program creates an @ phrase that contains all consecutive numeric IDs found in the original Pick dictionary starting with 1. For example, if the Pick dictionary contains the records 1, 2, 3, 4, 7, 8, 9, the following @ phrase is created:

```
      @
0001 PH
0002 1 2 3 4
```

The records 7, 8, and 9 are not included because the numeric sequence breaks after 4.

The @LPTR phrase specifies the default output specification to be used when the LPTR keyword sends output from a RETRIEVE command to the printer.

Create the @LPTR phrase as you would any other phrase. If the file dictionary does not contain an @LPTR phrase, RETRIEVE uses the @ phrase. If neither the file dictionary nor the VOC file contains an @ phrase, only record IDs are printed.

The @REVISE phrase specifies the names of fields that REVISE displays by default as input prompts for the data file. If you specify field names in the command that invokes REVISE, the @REVISE phrase is ignored. If the dictionary does not contain an @REVISE phrase, REVISE uses the @ phrase. If neither an @REVISE phrase nor an @ phrase is defined, REVISE creates an @REVISE phrase when it is first invoked on the file. When the REVISE processor creates an @REVISE phrase, it contains the names of all D-type fields currently defined in the file dictionary.

Chapter 6: Retrieve

An overview of Retrieve verbs	6-3
Retrieve syntax	6-6
Specifying records	6-7
Using the WITHIN keyword.	6-7
Selection expressions	6-8
Print limiting	6-10
Report and output keywords	6-11
Parenthetical options	6-12
REFORMAT and SREFORMAT	6-14
Creating and processing select lists	6-15
NSELECT	6-15
Using active select lists.	6-16
Retrieve command parsing.	6-18

Retrieve is the name of UniVerse's version of the Pick query language. It is similar to, and in most ways compatible with, versions of the query language found on other Pick systems, such as ACCESS (PICK Systems), ENGLISH (REALITY), RECALL (Ultimate), and INFORM (Prime INFORMATION).

This chapter assumes that you are familiar with how the Pick query language works on generic Pick systems. The emphasis here is on UniVerse enhancements to ACCESS and on the ways in which Retrieve works differently from ACCESS. A complete description of Retrieve is in the *Guide to Retrieve*.

An overview of Retrieve verbs

Retrieve verbs, like their Pick counterparts, can be used to do any of the following:

- Display and print lists.
- Generate reports.
- Select records that meet complex criteria.
- Select records for processing by other commands and utilities.
- Sort records by record ID or by data in any field.

Retrieve verbs contain a Q (for Query) in field 3 of the VOC entries that define them. The Retrieve verbs are listed below:

Verb	Description
CHECK.SUM	Computes a checksum for a file, a record, or a field.
COUNT	Counts specified records.
ESEARCH	Creates a select list of records containing an occurrence of a specified string. Only field values are searched; record IDs are not searched. UniVerse's ESEARCH command is a synonym for SEARCH.
LIST	Lists records.
LIST.ITEM	Lists all field values for specified records, with each field value on a separate line.
LIST.LABEL	Creates labels from specified records.
EFORMAT	Lists records, sending output to another file or to tape. REFORMAT both copies stored data and redirects Retrieve output.
SEARCH	UniVerse's SEARCH command is a synonym for ESEARCH. REALITY users should note that the SEARCH command is identical to the ESEARCH command: both commands accept Retrieve syntax.
SELECT	Creates an active select list.

RETRIEVE Verbs

Verb	Description
<code>SORT</code>	Lists records in sorted order.
<code>SORT.ITEM</code>	Lists all field values for specified records, with each field value on a separate line. Records are listed in sorted order.
<code>SORT.LABEL</code>	Creates labels from specified records. Labels are listed in sorted order.
<code>SREFORMAT</code>	Lists records in sorted order, sending output to another file or to tape. <code>SREFORMAT</code> both copies stored data and redirects Retrieve output.
<code>SSELECT</code>	Creates a sorted select list.
<code>STAT</code>	Totals the values of one or more numeric fields, counts the number of records selected, and averages the values of each field.
<code>SUM</code>	Totals the values of one or more numeric fields.
<code>T.DUMP</code>	Writes specified records to tape.
<code>T.LOAD</code>	Loads specified records from a tape created by the <code>T.DUMP</code> command to disk.

RETRIEVE Verbs (Continued)

In addition to the verbs in the preceding list, UniVerse has several verbs that can be used to generate and manipulate different kinds of select lists. Although strictly speaking they are not Retrieve verbs, they are so often used with them that we list them here:

Verb	Description
<code>COPY.LIST</code>	Copies a saved select list.
<code>DELETE.LIST</code>	Deletes a saved select list.
<code>EDIT.LIST</code>	Edits a saved select list.
<code>FORM.LIST</code>	Creates an active select list from the fields of one record or from a saved select list. UniVerse's <code>FORM.LIST</code> command is similar to the <code>FORM.LIST</code> command found on REALITY systems. It does not conform to the SMA standard version of <code>FORM.LIST</code> , which is a synonym for the <code>QSELECT</code> command.

Verbs Used with Select Lists

Verb	Description
GET.LIST	Retrieves (makes active) a saved select list.
NSELECT	Compares elements of an active select list to elements in a file, and creates a new select list of only those listed elements that are <i>not</i> in the file.
QSELECT	Creates a select list from field values of specified records instead of from record IDs.
SAVE.LIST	Saves an active select list.

Verbs Used with Select Lists (Continued)

Select lists are not saved to a POINTER-FILE, as they are on other Pick systems. They are saved as records in a Type 1 file called &SAVEDLISTS&.

HASH.TEST and ISTAT (called GROUP.STAT in UniVerse) are two Pick ACCESS verbs that are also available in UniVerse. However, they are not included among the Retrieve verbs, and they do not use standard ACCESS syntax as they do on other Pick systems. S-DUMP, which is the SORT version of T-DUMP, is not supported by UniVerse.

Retrieve syntax

Retrieve command syntax is like Pick ACCESS:

```
verb [ DICT ] filename [ records ] [ FROM list.# ] [ selection ] [ sort ]  
[ modifiers ] [ output ] [ (options) ]
```

As on other Pick systems, only the verb and a file name are required.

records are specified either by an explicit list of record IDs or by a selection expression. Record IDs in an explicit list can be enclosed in single or double quotation marks, but need not be. Backslashes are accepted only in PICK, IN2, and REALITY flavor accounts. When a selection expression is used to select record IDs, the WITH keyword is not used. Single quotation marks are permitted as well as double quotation marks in record ID selection expressions.

The FROM syntax may be new to Pick users. It lets you specify a currently active select list by number. Up to 11 select lists can be active at any one time. They are described more fully in REFORMAT and SREFORMAT.

You can use sort expressions with all RETRIEVE verbs (such as LIST, SELECT, and so forth), not just with sorting verbs (such as SORT, SSELECT, and so forth).

You can specify the fields to be listed either explicitly in the Retrieve sentence or by including the name of an output phrase. Output phrases are entries, either in the file dictionary or in the VOC file, that specify the names of fields to be output by a Retrieve sentence. When you use the name of the phrase in the sentence, it is equivalent to naming output fields in the sentence. Output fields are listed in columns in the order in which they appear in the sentence or phrase.

The @phrase is a special phrase used to specify default output. If the Retrieve sentence includes no output fields and if the file dictionary or the VOC file does not contain an @ phrase, only record IDs are output. If an @ phrase is defined, values for the fields specified by the @ phrase are listed in the report. If you include display fields in the RETRIEVE sentence, any @ phrase is ignored.



Note: Default output in UniVerse is controlled entirely by @ phrases. Numeric field definition items in sequence starting from 1 are not used to specify default output fields.

Parenthetical options can be used in PICK, IN2, and REALITY flavor accounts; they cannot be used in IDEAL or INFORMATION flavor accounts.

Specifying records

In PICK, IN2, or REALITY flavor accounts, RetrieveVe treats any element enclosed in single quotation marks as a record ID. Otherwise the RetrieveVe processor determines whether a word in a sentence is a record ID either by the position of the word in the sentence or by checking the file dictionary and the VOC file for an entry that corresponds to the word.

If a word is not part of a selection expression or a sort expression by virtue of its position, RetrieveVe looks in the file dictionary and the VOC file for a definition of the word. If no definition is found, RetrieveVe considers the word to be a record ID.

You can force RetrieveVe to use a word as a record ID, even if it is defined in the file dictionary or the VOC file, by enclosing it in either double or single quotation marks (or backslashes in PICK, IN2, and REALITY flavors). To prevent unquoted record IDs from being considered as multiple values in a selection expression, do not type them after a selection expression.

Using the WITHIN keyword

The WITHIN keyword works, as it does on other Pick systems, to retrieve one record with all of its related subrecords.

Subrecords are stored in exactly the same way as regular records. To make a record a subrecord, its record ID is stored as a value in a field containing record IDs of subrecords. Multiple subrecords can be made by storing record IDs as multivalues in the subrecord field. To use subrecords in UniVerse you will need to modify field 6 of your VOC file-defining entry. Field 6 should read

`0006: recordIDvmrecordIDvmrecordIDvm...recordID`

In Pick the subrecords are in field 8, preceded by V; ; `recordID. . .`

The syntax of the WITHIN expression is

WITHIN *filename* '*record.ID*'

filename is the name of the file.

record.ID is a record that contains a field with subrecords to be included in the listing. Only one record ID can be specified in a Retrieve statement containing a WITHIN expression.

Each subrecord encountered is assigned a *level number*. The record ID specified in the WITHIN expression is Level 1. If this record has subrecords, they are assigned Level 2. If Level 2's subrecords have subrecords, they are assigned Level 3, and so forth, up to a maximum of 20 levels. Level numbers are listed in front of the record and its subrecords.

Selection expressions

The selection expression has the following syntax:

WITH [**EVERY**] *field keyword value*
[{ **AND** | **OR** } [*field*] [*keyword*] *value*] . . .

The keyword IF can be used as a synonym for WITH. In an IN2 account you can use the keyword EACH as a synonym for EVERY.

The *field keyword value* expression varies from flavor to flavor. In IDEAL and INFORMATION flavor accounts, the keyword is required. In PICK, IN2, and REALITY flavor accounts, if the keyword is omitted, the relational operator EQ (=) is assumed. And, unlike on other Pick systems, *value* can be either a constant or the name of another field (for example, *field1 keyword field2*). If a second field is specified in the expression, data in the first field is compared to data in the second field to determine whether the record should be selected.

UniVerse lets you select a limited example the selection of records. This is useful to check the format of a long report on your terminal before printing the report on the system printer. The SAMPLE keyword (and its synonym, FIRST) specifies that only the first few records are to be selected. The SAMPLED keyword selects only every *n*th record of those specified.

Pattern-matching keywords

In addition to the string-searching characters used on generic Pick systems—left and right brackets ([]) and the up-arrow or caret (^) — Retrieve has a group of pattern-matching keywords you can use in selection expressions, WHEN clauses, IF statements, and REVISE sentences.

Keyword	Example	Pick Equivalent
LIKE MATCHES MATCHING	WITH LNAME LIKE "...DON..."	WITH LNAME "[DON]"
UNLIKE NOT.MATCHING	WITH NAME UNLIKE M...	WITH NAME # "M]"
SAID SPOKEN ~ (tilde)	WITH FNAME~ "JON"	No equivalent

Pattern-matching Keywords

In UniVerse the brackets and caret work as pattern-matching characters only in PICK, IN2, and REALITY flavor accounts.

In selection expressions, WHEN clauses, and IF statements, MATCHING compares a field value or prompt response to the pattern you specify. The pattern can contain any combination of letters or numbers. You can specify a match on a string at the beginning, at the end, or anywhere in the field. If the pattern contains any special characters (such as * or –) or a blank space, enclose the entire pattern in quotes, and enclose the string in another set of quotes. The pattern has one of the following syntaxes:

Syntax	Description
<i>string...</i>	To find records that have a field that begins with <i>string</i> . The Pick equivalent is <i>string</i>].

Pattern-matching Syntaxes

Syntax	Description
<i>...string...</i>	To find records that have <i>string</i> anywhere in the field. The Pick equivalent is [<i>string</i>].
<i>...string</i>	To find records that have a field that ends with <i>string</i> . The Pick equivalent is [<i>string</i>].
<i>string1...string2</i>	To find records that have a field that starts with <i>string1</i> and ends with <i>string2</i> with any number of characters between them. There is no Pick equivalent.

Pattern-matching Syntaxes (Continued)

Like Pick, the SAID and SPOKEN keywords (and the tilde character, ~) select records with values that “sound like” the specified value. The value specified in a SAID expression is converted into a phonetic equivalent based on the Soundex algorithm.

Print limiting

On generic Pick systems you can add a print limiting expression to any of the following:

- Field names used as output specifications
- A BY-EXP or BY-EXP-DSND exploding sort expression
- An output specification preceded by the TOTAL keyword

Print limiting works differently in UniVerse depending on the account flavor. In IDEAL and INFORMATION flavor accounts, print limiting is done using a WHEN expression. In PICK, IN2, and REALITY flavor accounts, you can use the standard Pick print limiting syntax as well, though it may not always give the results you expect. The WHEN keyword limits output to specified values from multivalued fields. If you do not use WHEN, Retrieve prints a line for every value of a multivalued field.

The syntax of the WHEN expression is

WHEN *field keyword value*

Although WHEN expression syntax resembles the WITH selection expression, it is not a selection expression; it is a *print limiting* expression. The WHEN expression limits *output* to specified *values*. The WITH selection expression limits *selection* to specified *records*.

WHEN expressions are used only with multivalued fields. WHEN expressions are used only with the LIST, SORT, SUM, STAT, SELECT, and SSELECT commands. If you use a WHEN expression with the SELECT or SSELECT command, you must also use either the BY.EXP or the BY.EXP.DSND keyword—otherwise the whole record will be selected instead of just the desired multivalues.

Report and output keywords

You will recognize most of the modifiers and options found on other Pick systems, along with some unfamiliar ones. Many of the familiar ones have variant forms in UniVerse for compatibility with different Pick systems: for example, hyphens and periods are both accepted (DBL-SPC and DBL.SPC). Slight differences in spelling should also be noted: for example, both COL-SUPP and COL.SUP are accepted, but COL.SUPP is not.

Pick users should note that the syntax of the BREAK.ON keyword differs in IDEAL flavor accounts from the syntax used in PICK, IN2, and REALITY accounts. In the latter, as on other Pick systems, the syntax is

BREAK.ON *field* ["*text* '*options*' *text* ..."]

In IDEAL and INFORMATION flavor accounts the syntax is

BREAK.ON ["*text* '*options*' *text* ..."] *field*

Parenthetical options

In PICK, IN2, and REALITY flavor accounts, Retrieve commands accept the following options.

Option	Description
C	Same as the COL.HDR.SUPP keyword. Suppresses column headings, the time, date, and page number. It has the same effect as the keyword COL-HDR-SUPP.
D	Suppresses detailed output when used with sentences that contain the BREAK.ON keyword. It has the same effect as the DET.SUPP keyword.
H	Same as the HDR.SUPP keyword. Suppresses default headers.
I	Suppresses the record ID. It has the same effect as the ID.SUPP keyword.
N	Suppresses paging. Output is scrolled when sent to the terminal. The N option has the same effect as NOPAGE
P	Sends output to the printer. It has the same effect as the keyword LPTR.

RETRIEVE Command Options

Almost all RETRIEVE commands ignore the options B, F, G, and T.

The following options work only with certain Retrieve commands:

Option	Description
A	The A option works with the SEARCH and ESEARCH commands. It specifies that all strings must match in order for the record to be selected (same as ALL.MATCH).
F	The F option is the same as the FORM.FEED keyword. In Retrieve it works only with the LIST.ITEM and SORT.ITEM commands, listing each item on a separate page.
I	The I option is the same as the ID.SUPP keyword except when used with the SEARCH and ESEARCH commands, where it lists record IDs of all selected records (same as SQUAWK).

Special Retrieve Command Options

Option	Description
L	The L option works with the SEARCH and ESEARCH commands. It lists, in addition to the record ID, the field and value number containing the specified string.
N	The N option is the same as the NOPAGE keyword except when used with the SEARCH and ESEARCH commands, where it specifies that only records <i>not</i> containing the specified strings be selected (same as NO.MATCH).
S	When used with the SEARCH and ESEARCH commands, the S option is the same as the NO.SELECT keyword: it lists record IDs of selected records without creating a select list. When used with the LIST.ITEM and SORT.ITEM commands, the S option is the same as the NO.WARN keyword: it suppresses the display of line numbers.
Special Retrieve Command Options (Continued)	

REFORMAT and SREFORMAT

The REFORMAT and SREFORMAT commands do not always work the same way on different Pick systems. Pick users may find that these two verbs do not work in the UniVerse environment exactly as they do on other systems.

The syntax of the UniVerse REFORMAT and SREFORMAT commands is as follows:

```
[S]REFORMAT [ DICT ] filename [ records ] [ FROM n ] [ selection ]  
[ sort ] output [ modifiers ] [ (options) ]
```

Both the file name and the output specification are required. You must specify at least one output field in order for REFORMAT to work. No default output specification is used. Neither record IDs nor fields specified in an @ phrase are output unless you specify output fields.

On a few Pick systems, the record IDs are output as they are by all other ACCESS commands. UniVerse, like most other systems, automatically suppresses record IDs, so you need not use the ID.SUP keyword or the I option in a REFORMAT statement. If you want to include record IDs as part of the output, include @ID in the output specifications.

The FROM keyword lets you specify a numbered select list.

On some Pick systems you can respond to the `File Name:` prompt in two ways:

1. Entering another file name, which directs output to another existing file.
2. Entering the keyword TAPE, to direct output to an assigned tape device.

UniVerse does not accept ENTER at the `File Name:` prompt; you can specify only another file or TAPE. In other words, you cannot REFORMAT a file back on to itself.

You cannot use the BY.EXP and BY.EXP.DSND keywords with REFORMAT and SREFORMAT to create separate records with singlevalued fields for each line of an exploded sort.

Creating and processing select lists

As on other Pick systems, you can create select lists of record IDs that can be used by other UniVerse verbs, BASIC programs, the UniVerse Editor, REVISE, and UniVerse utilities. These lists are created with the SELECT, SSELECT, NSELECT, QSELECT, and FORM.LIST commands.

Most Pick systems let you create or activate only one select list at a time. UniVerse lets you have up to 11 active select lists. The syntax of commands that create select lists is a little different from that on other Pick systems:

```
[S]SELECT [ DICT ] filename [ records ] [ FROM list.# ] [ selection ]  
[ sort ] [ SAVING [ UNIQUE ] field ] [ TO list.# ]
```

The three new syntactical elements are the FROM, SAVING, and TO expressions.

Use FROM to use a numbered select list in the [S]SELECT query.

Use SAVING when you want to create a select list from values in a field rather than from record IDs. Multiple SAVING expressions are allowed and work like a Pick output specification of multiple field names (each value becomes a separate element in the select list). Multivalues, however, are not listed as separate elements. (To create a select list that explodes multivalues so that each multivalue becomes a separate element in the list, use QSELECT.) The UNIQUE modifier suppresses duplicate values in the select list.

Use TO to specify a numbered select list. If you do not specify a numbered list, the list becomes Select List 0. Select List 0 behaves the way select lists do on other Pick systems: it must be used by the next command or it is lost. Select lists 1 through 10, however, act differently: they remain active, no matter what commands are executed from TCL, until you use them or clear them, or until you log off.

NSELECT

This command lets you use a select list to compare lists of record IDs in two files and select only those records in one file that are *not* in the other file.

For example, in the UV account there are five data files in a file called NEWACC that serve as templates for the VOC files of the five UniVerse flavors. If you want to find out what VOC entries in the PICK NEWACC file are not included in the IDEAL NEWACC file, do the following:

1. Select all items in the PICK NEWACC file:

```
>SELECT NEWACC,PICK
```

```
459 record(s) selected to SELECT list #0.
>>
```

2. Select all items in the IDEAL NEWACC file:

```
>>NSELECT NEWACC
```

```
6 record(s) selected to SELECT list #0.
>>
```

The new select list replaces the first one.

3. Sort the select list:

```
>>SORT ONLY NEWACC,PICK
```

The following report is listed:

```
SORT ONLY NEWACC,PICK 04:27:18pm 16 Apr 1991 PAGE 1
VOC.....
```

```
ACCOUNT.REST
ORE
DC
LISTFILES
LOGOFF
MD
PRINT-ERR
```

```
6 records listed.
```

Using active select lists

The elements selected in an active select list can be used as record IDs with any of the following UniVerse commands:

ANALYZE.FILE	FILE.USAGE	REFORMAT
BASIC	FILE.USAGE.CLEAR	RESIZE
CHECK.SUM	FILE.USAGE.OFF	REVISE

UniVerse Commands for Select Lists

CLEAR.FILE	FORMAT	SAVE.LIST
COPY	GROUP.STAT	SEARCH
COUNT	GROUP.STAT.DETAIL	SELECT
DECATALOG	HASH.AID	SORT
DELETE	HASH.HELP	SORT.ITEM
DELETE.FILE	HASH.HELP.DETAIL	SORT.LABEL
DISABLE.INDEX	HASH.TEST	SPOOL
ED	HASH.TEST.DETAIL	SREFORMAT
ENABLE.INDEX	LIST	SSELECT
ESEARCH	LIST.ITEM	STAT
FANCY.FORMAT	LIST.LABEL	SUM
FILE.SIZE	NSELECT	T.DUMP
FILE.STAT	QSELECT	

UniVerse Commands for Select Lists

These commands use a select list by performing their functions on each record specified by a record ID in the select list.

The Command Processor reminds you that a select list is active by

- Changing the prompt to >> instead of > for Select List 0.
- for some commands, asking you whether you want the select list used when you issue a command that can use a select list.

You can have up to 11 active select lists at any one time. The lists are designated as select list 0 through select list 10. Command Processor commands use select list 0 unless you specify a numbered select list.

Retrieve command parsing

Like Pick's ACCESS, UniVerse Retrieve sentences are parsed in two stages. In the first stage Retrieve scans the sentence for a file name and the keyword DICT. It does this by taking each word in the sentence (starting with the second word) and looking for a VOC entry with that word as its record ID. If there is a VOC entry for the word and the VOC entry is a file descriptor, Retrieve parses that word as the file name and stops scanning.

In the second stage Retrieve goes back to the second word of the sentence and parses each word in the sentence using the following rules:

1. If the word is quoted, Retrieve treats it as an explicit record ID or a constant value, depending on its position in the sentence. In PICK, IN2, and REALITY flavor accounts a word enclosed in single quotation marks is always treated as a record ID.
2. If the word corresponds to an entry in the file dictionary, Retrieve treats the word according to that definition. If the file specified in the Retrieve sentence is a dictionary, Retrieve uses the DICT.DICT file (located in the UV account) as the file dictionary for this step.
3. If the word does not correspond to an entry in the file dictionary, Retrieve checks to see if the word corresponds to an entry in the VOC file. If it does, Retrieve processes the word according to the type of VOC file entry.
4. If the word is not defined in either the file dictionary or the VOC file, Retrieve treats it as an explicit record ID or a constant value, depending on its position in the command line.

Once the command line has been parsed into the various components, Retrieve executes the specified function.

Chapter 7: Printer, terminal, and tape commands

Using logical print channels	7-3
Specifying printing forms	7-4
Printers and the UniVerse spooler	7-5
Printer configuration and status	7-5
Directing spooler output	7-6
Checking spool queues.	7-8
Managing print jobs.	7-8
Managing hold files	7-9
Spooling print jobs to and from tape	7-10
Terminals	7-11
Magnetic tapes	7-13

This chapter describes how to send output from UniVerse commands to printers and tape drives. It assumes you are familiar with the standard Pick spooler found on most R83 systems. It focuses on UniVerse spooler commands that are the functional equivalents of their Pick counterparts, and on specific differences between the ways Pick spooler commands and UniVerse spooler commands work.

The chapter is divided into four sections. The first section explains how to use logical print channels to route system output to the UniVerse spooler, to an assigned output device, or to a hold file in the special file &HOLD& in your account. The second section discusses printers and the UniVerse spooler. The third section describes the UniVerse commands used for setting, displaying, and changing UniVerse's terminal characteristics. The fourth section is an overview of UniVerse's tape-handling commands, most of which are familiar to Pick users.

Using logical print channels

UniVerse provides up to 256 logical print channels for redirecting output, compared to the single print channel available on most Pick systems. UniVerse's logical print channels are just like the ones used by the PRINT ON statement (BASIC) on other Pick systems, except that in UniVerse they are also available at the TCL level. If you do not specify a logical print channel for your output, it is sent to your terminal through a special terminal channel. UniVerse also provides eight logical tape channels for use with tape-related commands like T.READ or T.DUMP and tape-related UniVerse BASIC statements like READT or WEOF. Tape output is discussed in Magnetic tapes.

When you enter the UniVerse environment, all 256 logical print channels are available for use, with the default settings set. Default settings can be displayed for any logical print channel by entering the SETPTR command and the number of the channel. Default settings are shown in the following example:

```
>SETPTR 155
Unit Number      : 155
Page Width       : 132
Page Depth       : 66
Top Margin       : 3
Bottom Margin    : 3
Print mode       : 1 - Spooled Output

Default spool banner : "UniVerse"
```

You can display settings for only one logical print channel at a time. There is no UniVerse equivalent to the Pick LISTABS command, which lists spooler assignments for some or all lines on the system.

To specify which logical print channel to use for output, use the LPTR keyword followed by the channel number, or use the PRINT ON statement in a UniVerse BASIC program. In PICK, REALITY, and IN2 flavor accounts, you can use the parenthetical option P instead of the LPTR keyword. The P option sends output only to the default logical print channel, which is 0 (zero).

Specifying printing forms

On a generic Pick system, printer output is sent through the print channel to a specific spooler *form queue*. Printers are assigned to a form queue by the system administrator using the STARTPTR command. Printers service the form queue they are assigned to until they are explicitly reassigned to another form queue.

In UniVerse, printer output is sent to the spooler through one of the 256 logical print channels, whose characteristics are set by the SETPTR command. SETPTR has a FORM option that lets you specify which form should be used when printing the job. The FORM specified by SETPTR must match a FORM mounted on one of the system printers for the job to be printed. If it does not, the job is kept in the queue in a “wait” state until the proper form is mounted on the printer.

The System Administrator mounts a form on a printer at the time it is defined in the *sp.config* file. Multiple logical printer definitions for the same physical printer can be set up in *sp.config*, each with a different form mounted. This means that the administrator need not stop and restart a printer each time a different form is to be used.

Printers and the UniVerse spooler

Information about how to install and configure printers, including how to start, stop, and restart printers, how to remove them from the system, and how to assign forms to printers, is in *Administering UniVerse*. Look there, too, for information about starting, stopping, and restarting the UniVerse spooler.

Printer configuration and status

Information about printer configuration is located in a UNIX file called *sp.config* and in a UniVerse file called `&DEVICE&`. The *sp.config* file is usually located in the `/usr/spool/uv` directory, the `&DEVICE&` file is located in the directory where the UV account is located (usually `/u1/uv`). For information about these files, see *Administering UniVerse*.

On Pick systems the status of system printers can be shown with either the LISTPTR command or the SP-STATUS command. The UniVerse equivalent to LISTPTR is the SPOOL command with the `-LIST` option. SPOOL `-LIST` displays information about each printer as well as information about print jobs in all spool queues. The report looks like the following:

```
Printer: lp                Q: on      P: on      Form:
Job # Job description User name Pri Forms Size Cps Status Delay
00001 test              julie    32    1437    1 active 0:14
00003 portrait.file     ken      65 LW  12342    1 wait

Printer: lp2              Q: on      P: on      Form:
no entries.

Printer: lw               Q: on      P: on      Form: LW
no entries.

Printer: lwlscape         Q: on      P: on      Form: LSCAPE
Job # Job description User name Pri Forms Size Cps Status Delay
00002 lscape.file       walter   86 LSCAPE 213764    1 active

Printer: remote           Q: on      P: on      Form: REMOTE
no entries.
```

The report contains a line for each printer defined in the *sp.config* and `&DEVICE&` files, listing the printer name, the current status of queuing (on or off), the current status of printing (on or off), and the name of the currently mounted form (forms are discussed in *Specifying printing forms*). Any print jobs currently queued for printing are listed below the appropriate printer entry.

You can also display this report by choosing the “Status” option from the Printer Administration menu.

Directing spooler output

On a Pick system the SP-ASSIGN command directs spooler output to a specified destination: a printer, a tape device, or a hold file. SP-ASSIGN specifies output assignments for the account you are currently logged on to. Only one set of assignments for an account can be specified at any one time.

In UniVerse, printer output assignments are set with the SETPTR command. SETPTR assigns parameters to any of the 256 logical print channels that are available, although assignments for only 16 logical print channels can be set at any one time. SETPTR also lets you set line length, page length (number of lines per page), and top and bottom margins.

Use the SETPTR to specify the *output mode*. In UniVerse, three print channel output modes send output to three different destinations:

- to the spooler
- to an assigned device
- to a record in the &HOLD& file in your account.

The SETPTR command specifies several other printing parameters, such as which printer or form is to be used, number of copies to be printed, whether to eject a blank page between print jobs, and so forth.

You can also set some of the same print job characteristics with UniVerse’s SP.ASSIGN command, which is provided for compatibility with other Pick systems. SETPTR, however, is more versatile. If you do use SP.ASSIGN, be aware that it replaces any settings specified with a previous SETPTR command.

Output modes

The output mode is specified with the *mode* parameter of the SETPTR command as mode 1, 2, or 3.

Mode 1: the spooler.

Output directed to system printers is stored in a print file on disk for later printing by the system spooler. This is the default output mode. It corresponds to the way printer output is handled on most other Pick systems. You can use the SETPTR command to direct output to a specific printer, to spool jobs with special form names, to change the default page width and length, and to specify the number of copies to be printed.

Mode 2: assigned devices.

Typical Pick applications use UniVerse's default print channel 0 and do not need to use the ASSIGN command at all. You can use the ASSIGN command to take exclusive control of an output device by assigning one of the output devices defined in the &DEVICE& file to one of the 256 logical print channels. Output is sent immediately to the assigned device, bypassing the spooler. Output devices can be tape devices, printers, serial ports, disk files, UNIX pipes, and so forth. The device must have the proper permissions in order for you to assign it. The system administrator sets permissions when he defines the device in the *sp.config* and &DEVICE& files.

ASSIGN is like the P.ATT command, which attaches a printer to the user's account, and the T.ATT command, which attaches a tape device to an account. When you finish using the device, use the UNASSIGN command to release it for use by someone else. UNASSIGN is like the P.DET and T.DET commands, which detach a printer or a tape drive from an account.

Mode 3: hold files.

UniVerse has two types of hold files: print files with hold status (which are like hold files on other Pick systems—they are held in the spool queue before, during, and after printing), and print jobs that are stored as records in a type 1 file called &HOLD& in your account. You can use the SETPTR command to direct output from a logical print channel to a record in the &HOLD& file. Output sent through that logical print channel is collected in that record, which is held for later printing or other processing. Print files with hold status are described in Spooling print jobs to and from tape

Checking spool queues

On Pick systems the LISTPEQS command lists the status of all print jobs in the spooler form queues. The SPOOL command with the -LIST option displays not only a printer status report but also a report on the current state of the UniVerse spooler queues, including all waiting jobs, all jobs with hold status, and so forth. For each print job the report displays a line listing the following:

- Job number
- Description
- User name
- Priority
- Printing form to be used
- Number of copies to be printed
- Current status of the job
- Time delay

The job description is actually the source of the print job. For all print jobs generated in the UniVerse environment, the job description will be “UniVerse.” Notice that the user name is not the name of the UniVerse account from which the print job was generated, but the UNIX login name of the user who sent the job.

Managing print jobs

On Pick systems print jobs on the spool queue are managed with the SP-KILL and SP-EDIT commands. UniVerse makes it easier to manipulate print jobs. For example, on many Pick systems to cancel a print job waiting in the queue, you have to first make the print file a hold file using the SP-KILL command with the F option, then delete the hold file from the queue using the SP-EDIT command. In UniVerse, all you have to do is enter `SPOOL -CANCEL jobnumber`.

The easiest way to manage print jobs is from the Printer Administration menus, which are invoked by the PRINT.ADMIN command. See Printer configuration and status for a description of the available options.

Managing hold files

UniVerse uses two different types of hold files. Print jobs with hold status in the spool queue are similar to hold files on other Pick systems. Print files stored in the file &HOLD& in a user's account, however, may be new to Pick users.

Print jobs in the &HOLD& file

Hold files in the file &HOLD& are not queued for printing by the spooler. Sending a print job to &HOLD& is like using SP-EDIT on other Pick systems to send a print job to any regular file, where it can be stored for later printing.

To send a print job to the &HOLD& file, use the SETPTR command to set the output mode to 3. For example:

```
>SETPTR 0,,,,,3 BRIEF
>SORT CUSTOMERS BY LNAME LPTR
```

To print a file stored in &HOLD&, use the SPOOL command. For example, to print the &HOLD& file WBG_1097, enter the following:

```
>SPOOL &HOLD& WBG_1097
```

Print jobs with hold status

You can specify two kinds of hold status for a print file sent to the spool queue: permanent and temporary (most Pick systems support only the permanent variety). A temporary hold file is kept in the queue only until it is released for printing; a permanent hold file is like a Pick hold file—it is kept until you delete it from the spool queue.

Use the RETAIN or REQUEUE options of SETPTR to create a print job with permanent hold status. When you list such jobs with SPOOL -LIST, an asterisk after the word "hold" in the STATUS column indicates that the job will be kept even after printing. The RETAIN option is similar to the H option of SP-ASSIGN: it prints the job but keeps the print file as a hold file in the spool queue.

Use the HOLD option of SETPTR to create a print job that you want to save for later printing but that you do not want to keep permanently in the queue. Such jobs display the word “hold” in the STATUS column without the asterisk. Use the SP.EDIT command to send the job to the printer. After printing, the job is removed from the queue.

Spooling print jobs to and from tape

On Pick systems you can direct printer output to an attached tape device by specifying the T option of the SP-ASSIGN command. Print files stored on tape can be sent to the spooler for printing with the SP-TAPEOUT command.

In UniVerse you can direct a print job to a tape drive by assigning the tape drive to a logical print channel and then sending output to that channel. The syntax of the ASSIGN command for this is

ASSIGN *device* TO LPTR *n* [-WAIT]

device must be a record ID in the &DEVICE& file.

LPTR *n* specifies the number of the logical print channel.

Use the WAIT option to wait for the device to be unassigned by the current user. It is then assigned to you.

On generic Pick systems the SP-TAPEOUT command sends print jobs stored on tape to the spooler for printing. In UniVerse, use the SP.TAPE command. The tape drive must be defined in the &DEVICE& file, and you must assign the drive with the ASSIGN command before you use SP.TAPE.

Terminals

The terminal capabilities that Pick users are used to are greatly enhanced in the UniVerse environment. You can take advantage of the way UNIX handles terminals and create hardware-independent applications. This section summarizes the UNIX features that let you work with terminal characteristics. You will find more detailed information about UniVerse's terminal capabilities in *UniVerse System Description* and *Administering UniVerse*.

The UNIX operating system supports many different terminal types ranging from teleprinter terminals to Arts with sophisticated graphics capabilities. UniVerse, like many other programs available under UNIX, uses different terminal characteristics to do its job. Some programs use only the most basic features of a terminal, whereas others, such as the text editor *vi*, require that the program have a detailed knowledge of the terminal's capabilities.

In addition to the terminal type, several serial line characteristics (for example, baud rate and parity) may need to be defined for a terminal to work properly with the system.

The type of terminal attached to each serial line, and the line characteristics, are usually specified by the System Administrator. On some lines you may have to specify these characteristics after logging on to the system.

UniVerse has several commands for specifying and examining your terminal characteristics. They are GET.TERM.TYPE, SET.TERM.TYPE, TERM, and PTERM. GET.TERM.TYPE displays the code, model, and the page width and length of the terminal UniVerse thinks you are using. SET.TERM.TYPE sets or changes the terminal type code of your terminal. It corresponds to the TERM command with the *type* option on Pick systems.

TERM corresponds to the Pick TERM command. It sets or displays terminal and printer characteristics such as line and page length, line and form feed delays, number of lines skipped at the bottom of the screen, and the ASCII number corresponding to the backspace key. TERM does not change the actual characteristics of your terminal; it changes only how Retrieve commands perceive your terminal's characteristics when generating reports.

The PTERM command sets or displays those terminal characteristics that in the UNIX environment are set or changed by the UNIX *stty* command. Normally, terminal characteristics are taken from the *terminfo* entry for your terminal, but if any characteristics are set or changed with the PTERM command, the PTERM settings supersede the *terminfo* settings for as long as you remain in the UniVerse environment.

Magnetic tapes

Besides the 256 logical print channels that UniVerse provides, there are also eight logical tape channels, numbered 0 through 7. There is no default tape channel. You must use ASSIGN or T.ATT to assign a tape drive to one of the tape channels before you can use it. The logical tape channels, in conjunction with the magnetic tape commands, let you access a magnetic unit. The proper permissions must be set on the tape unit for you to gain access to it. Permissions are set by the System Administrator when the device is defined in the &DEVICE& file.

Most of the Pick magnetic tape commands are supported for compatibility. Differences in the way these commands work have to do mostly with differences between the tape format used by Pick machines and UniVerse. UniVerse supports the following tape handling commands:

ASSIGN	T.EOD	T.SPACE
SP.TAPE	T.FWD	T.UNLOAD
T.ATT	T.LOAD	T.WEOF
T.BCK	T.RDLBL	T.WTLBL
T.DET	T.READ	UNASSIGN
T.DUMP	T.REW	

UniVerse Tape Handling Commands

T.ATT is like ASSIGN. T.DET is like UNASSIGN. Both are supported for compatibility with other systems. For complete information on all of the magnetic tape commands, see the *UniVerse User Reference*.

Chapter 8: UniVerse BASIC

The UniVerse BASIC command	8-3
Options to the UniVerse BASIC command	8-3
Compiler directives	8-6
Successful compilation	8-12
The RUN command	8-14
Debugging your UniVerse BASIC program	8-16
Cataloging a UniVerse BASIC program	8-17
Redimensionable arrays	8-18
Pick-style COMMON	8-18
Pick-style DIM statement	8-19
Executing UniVerse commands	8-20
Sequential I/O access	8-21
Vector functions	8-28
The MATPARSE and MATBUILD statements	8-31
Using MATPARSE	8-31
Using MATBUILD	8-32
REMOVE statement	8-33

This chapter introduces UniVerse BASIC. It is intended for programmers who already know how to use BASIC in a Pick environment. For a complete description of UniVerse BASIC, see *UniVerse BASIC*.

This chapter describes features of UniVerse BASIC that may be new to a Pick user and that are not found in most versions of Pick. These include:

- Redimensionable arrays
- Sequential file processing
- Vector functions
- The MATPARSE and MATBUILD statements
- The REMOVE statement

The UniVerse BASIC command

The UniVerse BASIC command is essentially the same as that on other Pick systems. The principal difference is that parenthetical options are available only in PICK, REALITY, and IN2 flavor accounts. The syntax is as follows:

BASIC *filename* [*programs*] [*options*]

filename is the name of the file containing BASIC programs, and *programs* is a record ID list of one or more programs. You can compile more than one program at a time, but the programs must all be in the same file. Use an asterisk (*) to specify all programs in a file.

If you want to compile programs in the background, use the PHANTOM command. For example, the following command compiles all programs in the file BP. All output to the terminal is stored in a record named BASIC_ - *time_date* in your account's &PH& file (*time* and *date* are a time and date stamp):

>PHANTOM BASIC BP *

Options to the UniVerse BASIC command

Generic Pick systems provide several parenthetical options that can be used with the UniVerse BASIC command. UniVerse supports the following six Pick options to the BASIC command:

Option	Description
L	Generates a source code listing.
X	Generates a cross-reference table of statement labels and variable names.
S	Sends a listing of the source code to the printer.

Options to the BASIC Command

Option	Description
T	Suppresses the symbol table.
O	Performs a source-to-source optimization of the program before generating object code.
I	Suppresses execution of RAID or VLIST on a compiler or a UniVerse BASIC program.

Options to the BASIC Command (Continued)

These options can be specified in different ways, depending on the flavor of the account.

The following table lists the most common parenthetical options available on most Pick systems, and indicates how they are implemented in different UniVerse account flavors.

Option	Description	Implementation
A	Produces an assembly language listing of the compiled code.	Ignored.
C	Suppresses END OF LINE opcodes in the object code	EOL codes are not used.
E	List erroneous lines only.	Ignored.
L	List the source code.	Invokes -LIST option of BASIC command. In PICK, REALITY, and IN2 flavor accounts, can be specified with the parenthetical option L. The listing is saved as a record in another file whose name is the same as the source file name but with the suffix.L added to it.
M	Maps variables and statement labels.	Ignored.

Parenthetical Options to the BASIC Command

Option	Description	Implementation
P	Sends a listing of the source code to the printer.	Invokes -SPOOL option of BASIC command. In PICK, REALITY, and IN2 flavor accounts, can be specified with the parenthetical option P (the L option need not be specified).
S	Suppress the symbol table and line number table from object files.	Invokes -T option of BASIC command. In PICK, REALITY, and IN2 flavor accounts, can be specified with the parenthetical option S.
X	Generates a cross-reference listing showing the line numbers where each variable is referenced or assigned.	Invokes the -XREF option of BASIC command. In PICK, REALITY, and IN2 flavor accounts, can be specified with the parenthetical option X. The table is not saved in the BSYM file; it is saved as a record in a file whose name is the same as the source filename but with the suffix .L added to it.

Parenthetical Options to the BASIC Command (Continued)

The six UniVerse BASIC command options are -LIST, -SPOOL, -T, XREF, -I, and -O.

The -LIST option produces listings that contain all source lines, all lines inserted with the \$INSERT or \$INCLUDE statements, and all symbolic substitutions made by the EQUATE statement. The maximum number of variables and symbolic substitutions that the symbol table can hold is approximately 1,000, although the exact number of variables varies from program to program.

The -SPOOL option directs output to a printer rather than to a file. This allows you to spool program listings for printing.

The -T option suppresses the table of symbols and the table of line numbers that are appended to the end of object files. These tables are used for handling run-time error messages. Suppressing the tables results in somewhat smaller object code files, but run-time error messages will have no access to the line number or variable involved in the error. You cannot use the RAID debugger if you use this option.

The -XREF option produces an alphabetical cross-reference listing of every label and variable name used in the program. The listing shows each variable name and statement label with the line number on which it was defined or assigned a value (indicated by an asterisk) and the line number on which it was referenced.

The -I option suppresses execution of RAID or VLIST on a compiler or a BASIC program.

The -O option performs a source-to-source optimization of the program before generating object code.

Compiler directives

Unlike Pick, UniVerse has a set of BASIC statements that direct the behavior of the compiler. Among the functions performed by compiler directives are:

- including source code from one program in another program during compilation
- setting compile-time compatibility with another UniVerse flavor
- placing a condition upon compilation of certain parts of a program.

All these statements, except those specifying conditional compilation, are prefixed by a dollar sign (\$).

Including other programs

The statements \$INCLUDE, \$INSERT, and \$CHAIN instruct the compiler to include the source code of another program in the program currently being compiled. \$INCLUDE and \$INSERT insert other code in your program during compilation, returning afterwards to compile the next statement in your program. \$CHAIN also reads in the code of another program but after doing so does not return control to the source file. Any program statements following a \$CHAIN statement are not compiled.

Selecting flavor compatibility

A \$OPTIONS statement is a compiler directive that sets compile-time compatibility with any UniVerse flavor. By default, compatibility settings match the flavor of the account in which the program is compiled. Individual settings can also be specified within a program, overriding the usual setting. Specifying compatibility settings does not allow object code compiled in one flavor to execute in another; it allows only the emulation of capabilities of one flavor from within another flavor. The syntax for the \$OPTIONS statement is as follows:

```
$OPTIONS [ flavor ] [ options ]
```

flavor can be any of the following:

Flavor	Description
PICK	Compatibility with other Pick systems
INFORMATION	Prime INFORMATION compatibility
REALITY	REALITY compatibility
IN2	Intertechnique IN2 compatibility
DEFAULT	IDEAL UniVerse

UniVerse Flavors

For instance, in a PICK flavor account you could compile a program compatible with the IDEAL UniVerse by including the following statement in the source:

```
$OPTIONS DEFAULT
```

This statement instructs the compiler to use the set of compatibility options specified for the IDEAL UniVerse.

The \$OPTIONS statement does not change the flavor of an account or otherwise affect the compatibility of the account in which the program is run. We recommend that you run your program in the same flavor account as specified by the \$OPTIONS statement.

The following table lists the options available with the \$OPTIONS statement.

Option	Description	Default Flavors
COUNT.OVLP	Counting overlaps in INDEX and COUNT functions.	PICK, REALITY, IN2
END.WARN	A warning message is printed if there is no final END statement.	INFORMATION, REALITY
EXEC.EQ.PERF	<p>EXECUTE means PERFORM.</p> <p><i>Note: If the syntax of the EXECUTE statement is changed so it is no longer compatible with the PERFORM statement, UniVerse ignores EXEC.EQ.PERF. For example, UniVerse ignores EXEC.EQ.PERF in the following program:</i></p> <pre>0001 "\$OPTIONS EXEC.EQ.PERF 0002 EXECUTE 'DATE' CAPTURING RESULTS 0003 END</pre>	INFORMATION
EXTRA.DELIM	For INSERT and REPLACE functions, handles null fields, values, and subvalues differently from IDEAL flavor. In particular, if you specify a negative one (-1) parameter, IN2 flavor adds another delimiter, except when starting with a null string.	INFORMATION, IN2
FOR.INCR.BEF	Index for FOR...NEXT loop is incremented before instead of after the bound checking.	IDEAL, PICK, REALITY, IN2

Options to the \$OPTIONS Statement

Option	Description	Default Flavors
FORMAT.OCONV	Allows format masks to be used as output conversion codes.	REALITY
HEADER.EJECT	HEADING statement causes initial page eject.	INFORMATION, REALITY
HEADER.DATE	Time and dates within headings are displayed in fixed format (for example, they do not change from page to page). Dates are displayed in D2- instead of D format. Allows page number field specification by multiple invocations of 'P' in single quotes.	INFORMATION
IN2.SUBSTR	Use IN2 definitions for BASIC substring handling (<i>string</i> [<i>n,m</i>]). If a single parameter is specified, it specifies <i>n</i> (start), not <i>m</i> (length), and a length of one is assumed.	IN2
INFO.ABORT	ABORT syntax follows Prime INFORMATION instead of PICK.	INFORMATION
INFO.LOCATE	LOCATE statement uses Prime INFORMATION syntax instead of REALITY syntax. The Pick format of the LOCATE statement is always supported.	INFORMATION
INPUT.ELSE	Accepts options THEN...ELSE statements on INPUT statement.	PICK

Options to the \$OPTIONS Statement (Continued)

Option	Description	Default Flavors
NO.RESELECT	If Select List 0 is already active for SELECT and SSELECT statements, it remains active: another selection or sort will not be done. The next READNEXT statement uses Select List 0.	PICK, IN2
ONGO.RANGE	If the value used in an ON...GOTO or ON...GOSUB statement is out of range, execute the next statement rather than the first or last branch.	PICK, IN2, INFORMATION
PCLOSE.ALL	PRINTER CLOSE statement closes all print channels.	PICK, REALITY, IN2
PERF.EQ.EXEC	PERFORM means EXECUTE	REALITY
RADIANS	Calculates trigonometric operations using radians instead of degrees.	IN2
READ.RETAIN	If a READ, READU, READV, or READVU fails, the result variable retains its value; it is not set to null.	PICK, REALITY, IN2
REAL.SUBSTR	Use REALITY definitions for substring handling (<i>string</i> [<i>n,m</i>]). If <i>m</i> or <i>n</i> is less than 0, the starting position for substring extraction is defined as the right side, or the end of the string.	REALITY
RNEXT.EXPL	READNEXT returns exploding select list 0	INFORMATION

Options to the \$OPTIONS Statement (Continued)

Option	Description	Default Flavors
SEQ.255	SEQ(' ')=255 (instead of 0).	PICK, REALITY, IN2
STATIC.DIM	Arrays are created at compile time, not at run time. The arrays are not redimensionable, and they do not have a zero element.	PICK, REALITY, IN2
STOP.MSG	STOP and ABORT print error messages from ERRMSG.	PICK, REALITY, IN2
SUPP.DATA. ECHO	Input statements suppress echo from the data stack.	PICK, REALITY, IN2
USE.ERRMSG	PRINT ERR prints error messages from ERRMSG.	REALITY
VAR.SELECT	SELECT TO variable, creates a local select variable instead of using numbered select lists.	PICK, IN2
VEC.MATH	Use vector arithmetic instructions for operating on multivalued data.	INFORMATION

Options to the \$OPTIONS Statement (Continued)

Individual options can be turned on or off by specifying them in a \$OPTIONS statement. If prefixed by a minus sign (-), the option is turned off.

See the \$OPTIONS statement page in *UniVerse BASIC* for other ways to select compile-time flavor compatibility.

Specifying conditional compilation

You can specify the conditions under which all or part of a BASIC program is compiled, using a modified version of the IF statement. Conditional compiling is useful for customizing programs that are to be used by more than one kind of user. It also helps to reduce the size of the object code and to increase program efficiency.

The syntax of the conditional compilation statement requires a special version of the test expression, the syntaxes are

IF \$TRUE THEN *statements* ELSE *statements*

IF \$T THEN *statements* ELSE *statements*

IF \$FALSE THEN *statements* ELSE *statements*

IF \$F THEN *statements* ELSE *statements*

The conditional compilation statement can specify a variable name rather than one of the test specifications listed above. If it does, an EQUATE statement must appear at the beginning of the source code, equating the variable to the test specification and using the following syntax:

EQUATE *variable* LIT "\$T"

Note: Code that is not compiled because the condition evaluates to false must nevertheless be syntactically correct; otherwise no part of the program will be compiled.



Successful compilation

As the compiler attempts to compile a program, it displays various warning messages and error messages disclosing any problems that exist in the source code (for example, statement format errors). Warning messages do not terminate the compilation process. Error messages abort compilation. All errors must be corrected before compilation is successful.

During compilation, an asterisk appears on the screen for every 10 lines (instead of every line) of source code successfully compiled. If an error is encountered, a question mark appears.

When the source code contains no errors, the compiler produces an object code record. The object code record is stored in a file with the same name as the source code file suffixed with ".O". The record ID of the object code record is the same as the record ID (program name) of the source code record.

For example, if your source code record name is MAIN and it is stored in the file BP, executing the following command:

```
>BASIC BP MAIN
```

compiles the source code in record MAIN in file BP, producing object code that is stored in record MAIN in file BP.O. The compiler automatically creates the object code file if it does not exist before compilation.

The RUN command

After you have successfully compiled your program, you can run it from TCL with the RUN command. The syntax is as follows:

RUN [*filename*] *program* [*options*]

The RUN command appends .O to *filename* and then executes that object file. Unlike Pick, if *filename* is omitted the default file BP is assumed. *program* is the name that identifies the program record in the file.

options can be any of the following:

Option	Description
NO.WARN	Causes suppression of all warning (nonfatal) error messages. In PICK, REALITY, and IN2 flavor accounts you can specify this option as (S).
NO.PAGE	Automatic paging is turned off. (Programs that position the cursor with @ functions do not need to disable pagination.) In PICK, REALITY, and IN2 flavor accounts you can specify this option as (N).
KEEP.COMMON	If the program is executed from within a chain, links the unnamed common.
LPTR	Spools program output to the printer rather than to the terminal screen. In PICK, REALITY, and IN2 flavor accounts you can specify this option as (P).
TRAP	Enters the interactive debugger, RAID, whenever a nonfatal error occurs.

RUN Command Options

For example, the following command executes the file BP.O/MAIN:

>RUN BP MAIN

Run-time error messages are printed on the terminal screen as they are encountered, unless the NO.WARN keyword or the S option was specified.

Generic Pick systems provide several parenthetical options to the RUN command. Of these, UniVerse supports only the N (no page), P (printer), and S (suppress messages) options. The A, D, and E options are ignored, and the I and L options have not been implemented.

Debugging your UniVerse BASIC program

RAID, like Pick's DEBUG, is an interactive debugger. It is a powerful tool for detecting errors in UniVerse BASIC code. RAID can function as either an object code or a source code debugger. It lets you set and delete breakpoints, set watchpoints, step through and display source code, examine object addresses, and examine and modify variables. For complete information about RAID, see *UniVerse BASIC*.

Cataloging a UniVerse BASIC program

You must catalog a UniVerse BASIC program in order to

- Use the program as a subroutine in an I-descriptor.
- Execute the program without using the RUN command.

UniVerse BASIC provides three ways to catalog a program: locally, normally, and globally. On generic Pick systems, including IN2, all cataloging is *local*; on REALITY systems, all cataloging is *normal*. In PICK, IN2, and REALITY flavor accounts, the CATALOG command functions as it does on a Pick, IN2, or REALITY system. Its syntax lets you specify an item-list of program names or use an asterisk to specify all records in a file.

In an IDEAL flavor account, the CATALOG command works differently, allowing you to choose from global, normal, or local cataloging options. For more information, see the *UniVerse System Description*.

Redimensionable arrays

On most Pick systems, dimensioned arrays are allocated at *compile time*. Once the dimensions are specified (with the DIM statement), they cannot be changed. Arrays can be dimensioned anywhere in a program, but they cannot be redimensioned at run time. Indices specified in the DIM statement must be integer constants. If the number of elements assigned to an array (for example, by the MATPARSE, MATREAD, or MATWRITE statements) is greater than the dimensions of the array, the overflow is stored in the last element of the array.

In UniVerse, dimensioned arrays are allocated at *run time*. They can be redimensioned at any time. Variables and expressions can be used as indices in the DIM statement. Overflow values are stored in a special *zero-element*, which can be specified as *var(0)* (for vectors) or *var(0,0)* (for matrices).

Pick-style dimensioned arrays are called *fixed* arrays; UniVerse's redimensionable arrays are called *standard* arrays. Standard arrays are the default type in IDEAL UniVerse and INFORMATION flavor accounts. Fixed arrays are the default in PICK, IN2, and REALITY flavor accounts. You can use the STATIC.DIM option of the \$OPTIONS statement in any flavor account to specify that you want the DIM statement to use fixed dimensioned arrays.



Warning: *Do not mix programs and subroutines that use different kinds of array. Data could be corrupted.*

Pick-style COMMON

Arrays declared in a Pick-style COMMON statement are dimensioned at compile time. Arrays in COMMON statements act like a set of scalars. For example, the following declaration specifies eight variables.

```
COMMON A, B, C(5), D
```

The syntax of the COMMON statement is as follows:

```
COM[MON] [ /name/ ] variable [ ,variable ... ]
```

UniVerse allows a named COMMON in all account flavors. The difference between named and unnamed COMMON is that an unnamed COMMON is lost when the program completes its execution and control returns to the UniVerse command level, whereas a named COMMON remains accessible for as long as the user remains in the UniVerse environment.

name must be enclosed between slashes and can be any number of characters, but only the first 31 characters of *name* are significant.

Pick-style DIM statement

On most Pick systems the DIMENSION statement is evaluated at compile time. In UniVerse, the behavior of the DIM statement is flavor-dependent. In IDEAL and INFORMATION flavor accounts the DIM statement is executed at run time; in PICK, IN2, and REALITY flavor accounts, DIM is evaluated at compile time.

The advantage of the way UniVerse handles the DIM statement is that the amount of memory allocated is not determined until the DIM statement is executed. This means that arrays can be redimensioned at run time. However, this method imposes the stricture that a program must flow through the DIM statement before encountering any array element.

Executing UniVerse commands

Like Pick, the EXECUTE statement submits a TCL command for processing from a UniVerse BASIC program. The command can be any UniVerse command, stored sentence, paragraph, menu, or proc.

You can specify multiple commands in the EXECUTE statement in the same way that you specify them in the body of a UniVerse paragraph. Separate each command or line with a field mark.

In its simplest form, the EXECUTE syntax is

EXECUTE *commands*

The EXECUTE statement evaluates the expression *commands* as one or more commands to be executed in UniVerse. Control then returns to the UniVerse BASIC program at the next statement following the EXECUTE statement. This differs from the CHAIN statement, which terminates execution of a UniVerse BASIC program after executing a UniVerse command. After executing a CHAIN statement, control returns to the environment that called the UniVerse BASIC program.

UniVerse BASIC accepts several different forms of the EXECUTE statement that allow specification of input to the executed command or capturing of the output of the executed command. For more information, see *UniVerse BASIC*.

Sequential I/O access

Unlike Pick, UniVerse supports I/O access to standard hashed files and sequential I/O access. Any of the following can be opened to a UniVerse file variable for sequential processing by using the OPENSEQ or OPENSEQ statements:

- A record in a UniVerse nonhashed file
- A UNIX file such as *myfile* or */etc/passwd*
- A device name such as */dev/tty* or */dev/rmt0*

OPENSEQ opens a file for sequential processing. The first form is used to open a record in a UniVerse nonhashed file. The second form is used to open a UNIX file or any UNIX device, specified by its full path.

```
OPENSEQ file, record.ID TO file.variable [ LOCKED statements ]  
{ THEN statements [ ELSE statements ] | ELSE statements }  
  
OPENSEQ pathname TO file.variable [ LOCKED statements ]  
{ THEN statements [ ELSE statements ] | ELSE statements }
```

When a file is first opened for sequential processing, the current position is the beginning of the file. If the file does not exist, the open will fail. However, a write operation (WRITESEQ or WRITEBLK) automatically creates the file, or the file can be created explicitly with a CREATE statement.

Use the CLOSESEQ statement to close a file opened with OPENSEQ.

```
READSEQ variable FROM file.variable  
{ THEN statements [ ELSE statements ] | ELSE statements }
```

READSEQ reads data from the current position in the file up to a NEWLINE (ASCII 10) and assigns the data to *variable*. The current position is incremented to after the last character read. The NEWLINE character is not part of the data assigned to *variable*. The ELSE clause is taken if the file is not readable or if the end of file is encountered.

```
WRITESEQ expression { ON | TO } file.variable  
{ THEN statements [ ELSE statements ] | ELSE statements }
```

WRITESEQ writes data starting at the current position, and terminates the data with a NEWLINE (ASCII 10). The current position is incremented to beyond the last character written. The ELSE clause is taken if the file is not writable. WRITESEQ does not require the current position to be at end of file.

In this example the OPENSEQ statement is used to open a UNIX file by specifying its full path. */etc/passwd* is a system file that contains the login information about users.

```
0001:  OPENSEQ '/etc/passwd' TO log
0002:  THEN PRINT "/etc/passwd OPENED" ELSE ABORT
0003:  FOR N=1 TO 4
0004:      READSEQ name FROM log THEN PRINT name
0005:  NEXT N
0006:  CLOSESEQ log
```

When the program is run, a line from */etc/passwd* is read and assigned to the variable *name*; this line is printed and the program loops back to read another line.

```
>RUN DBP SEQ
/etc/passwd OPENED

root:kpjN8fTbDXhis:0:0:0000-Admin(0000),,:/bin/tcsh
alis:6V03/.3SZ1agQ:0:0:ALIS Administrator account -
cac:/rd/alis:/bin/tcsh
msgs*:24:1:messages:/usr/msgs:
daemon*:1:1:0000-Admin(0000):/:
```

The following statements are additions to the UniVerse BASIC language that give greater flexibility and power to sequential processing operations:

CREATE	OPENDEV	SEEK	WEOFSEQ
FLUSH	OPENPATH	STATUS	WRITEBLK
NOBUF	READBLK	TTYCTL	

Commands for Sequential Processing

A brief description of these statements follows.

```
CREATE file.variable { THEN statements [ ELSE statements ] | ELSE
statements }
```

CREATE is primarily used in an ELSE clause of an OPENSEQ statement to prepare the file for a NOBUF, READSEQ, or READBLK operation. An OPENSEQ for the file variable must be executed before the CREATE statement in order to establish the pathname or nonhashed record to be created. CREATE is not needed if the first operation to the file is a WRITESEQ operation since a WRITESEQ will automatically create the file. It is needed if the first operation is to be a NOBUF, READSEQ, or READBLK.

```
FLUSH file.variable { THEN statements [ ELSE statements ] | ELSE
statements }
```

FLUSH immediately writes all buffers. Normally SEQ I/O uses buffering, and writes are not done immediately. The ELSE clause is taken if the file is not open.

```
NOBUF file.variable { THEN statements [ ELSE statements ] | ELSE
statements }
```

NOBUF turns off buffering for the sequential file. It eliminates the need for FLUSH operations, while also eliminating the benefit of buffering. The NOBUF statement must be executed after a successful open or CREATE statement and before any I/O has occurred. The ELSE clause is taken if the file is not open, or if the NOBUF statement is executed after I/O has occurred.

```
OPENDEV device TO file.variable [ LOCKED statements ] { THEN
statements [ ELSE statements ] | ELSE statements }
```

OPENDEV opens a device. The device or file specified by *device* must be a record ID of a device definition record listed in the &DEVICE& file. The ELSE clause is taken if the device cannot be opened.

```
READBLK variable FROM file.variable, blocksize { THEN statements
[ ELSE statements ] | ELSE statements }
```

READBLK reads a block of data of length *blocksize* beginning at the current position in the file, and assigns the data to *variable*. The current position is incremented to beyond the last character read. The ELSE clause is taken if the file is not readable, or if the end of file is encountered. If the ELSE is taken, *variable* is set to a null value.

```
SEEK file.variable [ ,offset [ ,relto ] ] { THEN statements [ ELSE
statements ] | ELSE statements }
```


SEEK moves the current pointer *offset* number of bytes relative to the position specified by *relto*. If *relto* is:

Setting	Description
0	relative to the beginning of the file
1	relative to the current position
2	relative to the end of the file

***relto* Settings**

If *relto* is not specified, 0 is assumed. The ELSE clause is taken if the file is not open, if the seek is before the beginning of the file, or if the device is not seekable.

Seeking beyond the end of the file and then writing creates a gap or “hole” which occupies no physical space and reads as ASCII 0 (not the number or character 0). A negative *offset* specifies a position in the file before the reference point specified by *relto*.

```
STATUS variable FROM file.variable{ THEN statements [ ELSE  
statements ] | ELSE statements }
```

STATUS returns the file status of the sequential file as a dynamic array variable defined as follows:

Field	Stored Value
1	Current position in the file
2	End of file reached
3	Error accessing file
4	Number of characters available to read
5	File mode
6	File size
7	Number of hard links
8	User ID of owner
9	Group ID of owner
10	I-node number
11	Device on which i-node resides
12	Device for character special or block
13	Time of last access
14	Date of last access
15	Time of last modification
16	Date of last modification
17	Time of last status change
18	Date of last status change
19	Number of characters left in output queue (applicable to terminals only)
20	UNIX file name

STATUS Codes

Field	Stored Value
21	UniVerse file type
22	UniVerse file modulo
23	UniVerse file separation

STATUS Codes (Continued)

TTYCTL *file.variable argument* { THEN *statements* [ELSE *statements*]
| ELSE *statements* }

TTYCTL controls sequential files when they are associated with a TTY (terminal device). *argument* is defined as follows:

Arg.	Operation
0	No operation. Used to determine if a device is a TTY.
1	Set HUP (hang-up data line) on close of file.
2	Clear HUP on close of file.
3	Set exclusive use flag for TTY.
4	Reset exclusive use flag.
5	Set the BREAK.
6	Clear the BREAK.
7	Turn on DTR (Data Terminal Ready).
8	Turn off DTR.
9	Flush input and output buffers.
10	Wait for the output buffer to drain.

TTYCTL Arguments

The ELSE statements are executed if an error is encountered during execution of the TTYCTL operation or if the file is not open.

WEOFSEQ *file.variable* { THEN *statements* [ELSE *statements*] | ELSE *statements* }

WEOFSEQ truncates the file at the current position.

```
WRITEBLK expression ON file.variable { THEN statements [ ELSE  
statements ] | ELSE statements }
```

WRITEBLK writes the value of *expression* as data, starting at the current position in the file. WRITEBLK does *not* add a NEWLINE character at the end of the data. The current position is incremented to beyond the last character written. The ELSE clause is taken if the file is not writable.

Vector functions

UniVerse BASIC provides a number of special functions that are designed to perform common operations on dynamic arrays. They are sometimes called *vector* functions because they process lists of data rather than single values. Use the \$OPTIONS statement to enable the arithmetic operators (+, -, *, /) to operate on dynamic arrays as lists of data, or vectors.

Standard functions in UniVerse BASIC process all variables as singlevalued variables, treating delimiter characters as part of the data. Vector functions, on the other hand, recognize delimiter characters and process each field, value, and subvalue individually. In fact, they process singlevalued variables as if they were dynamic arrays with only the first value defined. In all other respects vector operations are essentially the same as those performed by the standard functions or operators corresponding to them.

Vector functions have also been implemented as subroutines for compatibility with existing UniVerse BASIC programs. Each subroutine has a name made up of the name of the function preceded by a hyphen. The subroutines are cataloged globally and can therefore be accessed using the method described in the CALL statement.

Function	Corresponding Instruction for Singlevalued Field
ADDS (<i>m1</i> , <i>m2</i>)	<i>s1</i> + <i>s2</i>
ANDS (<i>m1</i> , <i>m2</i>)	<i>s1</i> AND <i>s2</i>
CATS (<i>m1</i> , <i>m2</i>)	<i>s1</i> : <i>s2</i>
CHARS (<i>m1</i>)	CHAR (<i>s1</i>)
COUNTS (<i>m1</i> , <i>p1</i>)	COUNT (<i>s1</i> , <i>p1</i>)
DIVS (<i>m1</i> , <i>m2</i>)	<i>s1</i> / <i>s2</i>
EQS (<i>m1</i> , <i>m2</i>)	<i>s1</i> EQ <i>s2</i>
NES (<i>m1</i> , <i>m2</i>)	<i>s1</i> NE <i>s2</i>
LES (<i>m1</i> , <i>m2</i>)	<i>s1</i> LE <i>s2</i>

Functions for Dynamic Arrays and Singlevalued Fields

Function	Corresponding Instruction for Singlevalued Field
LTS (<i>m1, m2</i>)	<i>s1</i> LT <i>s2</i>
GES (<i>m1, m2</i>)	<i>s1</i> GE <i>s2</i>
GTS (<i>m1, m2</i>)	<i>s1</i> GT <i>s2</i>
NOTS (<i>m1</i>)	NOT (<i>s1</i>)
FIELDS (<i>m1, p1, p2, p3</i>)	FIELD (<i>s1, p1, p2, p3</i>)
FMTS (<i>m1, p1</i>)	FMT (<i>s1, p1</i>)
ICONVS (<i>m1, p1</i>)	ICONV (<i>s1, p1</i>)
IFS (<i>m1, m2, m3</i>)	IF <i>s1</i> THEN <i>s2</i> ELSE <i>s3</i>
INDEXS (<i>m1, p1, p2</i>)	INDEX (<i>s1, p1, p2</i>)
LENS (<i>m1</i>)	LEN (<i>s1</i>)
MODS (<i>m1, m2</i>)	MOD (<i>s1, s2</i>)
MULS (<i>m1, m1</i>)	<i>s1</i> * <i>s2</i>
NUMS (<i>m1</i>)	NUM (<i>s1</i>)
OCONVS (<i>m1, p1</i>)	OCONV (<i>s1, p1</i>)
ORS (<i>m1, m2</i>)	<i>s1</i> OR <i>s2</i>
SEQS (<i>m1</i>)	SEQ (<i>s1</i>)
STRS (<i>m1, p1</i>)	STR (<i>s1, p1</i>)
SPACES (<i>m1</i>)	SPACE (<i>s1</i>)
SPLICE (<i>m1, p1, m2</i>)	<i>s1</i> : <i>p1</i> : <i>s2</i>
SUBSTRINGS (<i>m1, p1, p2</i>)	<i>s1</i> [<i>p1, p2</i>]
SUBS (<i>m1, m1</i>)	<i>s1</i> – <i>s2</i>
TRIMS (<i>m1</i>)	TRIM (<i>s1</i>)

Functions for Dynamic Arrays and Singlevalued Fields (Continued)

The first column of preceding table shows the functions for manipulating dynamic arrays that are available in UniVerse BASIC. The second column shows the corresponding instructions to use for singlevalued variables. *m1* and *m2* represent dynamic arrays; *s1* and *s2* represent singlevalued variables; *p1*, *p2*, and so on, represent singlevalued parameters. The value of the function is the resulting dynamic array.

When two dynamic arrays are processed by a function or an operator, the lists are processed in parallel. In other words, the first value of field A and the first value of field B are processed together. Then the second value of field A and the second value of field B are processed together, and so on.

The REUSE function can be used with vector functions when the two dynamic arrays being processed by them contain an unequal number of fields, values, or subvalues. When vector functions are used without the REUSE function, zeros, or empty strings are added to the shorter array until the two lists are equal. The REUSE function causes the last value in the shorter list to be reused instead, until all of the elements in the longer list are exhausted or until the next higher delimiter is encountered.

The MATPARSE and MATBUILD statements

The MATPARSE statement loads a string or dynamic array into elements of a dimensioned array; the MATBUILD statement writes the elements of a dimensioned array into a dynamic array.

Using MATPARSE

With MATPARSE, the string or dynamic array is parsed by supplying a delimiter used to separate fields. The syntax of the MATPARSE statement is

```
MATPARSE array FROM dynamic.array [ ,delimiter ]  
MATPARSE array [ ,start [ ,end ] ] FROM dynamic.array [ USING  
delimiter ] [ SETTING elements ]
```

dynamic.array is parsed according to the specified *delimiter* and read into consecutive elements of *array*. The *array* must be named and dimensioned in a DIMENSION or COMMON statement before it is used. A comma separates *delimiter* from *dynamic.array*. The *delimiter* can be specified as follows:

Delimiter	Description
empty pair of quotation marks	Parses <i>dynamic.array</i> so that each character becomes one element of <i>array</i> .
single character	Parses <i>dynamic.array</i> taking the substrings that are between successive delimiter characters as elements in the <i>array</i> . The delimiter characters are not stored in <i>array</i> .
two or more characters	Parses <i>dynamic.array</i> by taking as elements the substrings that are between any two successive delimiters. All of the characters up to, but not including, any single delimiter character are stored as an element of <i>array</i> . The delimiter character and any identical consecutive delimiter characters are stored as the next element. The search then continues as at the start of the string.

Delimiters

A delimiter such as `/:` includes the two characters `/` (slash) and `:` (colon). It might be used to separate hours, minutes, seconds and month, day, year in the formats `12:32:16` and `1/23/85`. A delimiter of two spaces `" "` might be used to separate tokens on a command line that contain multiple blanks between tokens. See *UniVerse BASIC* for more information.

Using MATBUILD

The MATBUILD statement is the inverse of the MATPARSE statement. A dynamic array converted into a dimensioned array with MATPARSE can be reconstructed, using the same delimiters, with MATBUILD. The syntax of MATBUILD is:

```
MATBUILD dynamic.array FROM array [ ,start [ ,end ] ] [ USING
delimiter ]
```

It builds a dynamic array by concatenating the elements of *array*, beginning with *start* and finishing with *end*. The *array* must be named and dimensioned in a DIMENSION or COMMON statement before it is used. If *start* and *end* are not specified or are out of range, they default to 1 and the size of the array, respectively.

delimiter can be specified as follows:

Delimiter	Description
empty pair of quotation marks	Each element of <i>array</i> is placed into <i>dynamic.array</i> without delimiters.
single character	Each element of <i>array</i> is placed into <i>dynamic.array</i> separated by <i>delimiter</i> .
two or more characters	The fields of <i>array</i> are loaded into <i>dynamic.array</i> as alternating data and delimiter elements.

MATBUILD Delimiters

If *delimiter* is not specified, a field mark is used. For more information, see *UniVerse BASIC*.

REMOVE statement

The REMOVE statement extracts consecutive elements, separated by system delimiters, from a dynamic array. Because the REMOVE statement maintains a pointer in the dynamic array, it is an efficient method for extracting successive fields, or successive values in a multivalued field, from a record in a UniVerse file.

REMOVE *element* FROM *dynamic.array* SETTING *variable*

The REMOVE statement extracts one element each time it is executed, beginning with the first element in *dynamic.array*. When a system delimiter is encountered, the value of the extracted element is assigned to *element*. A code value is assigned to *variable* corresponding to the system delimiter terminating the element just removed. For more information, see *UniVerse BASIC*.

Chapter 9: Application development tools

The ProVerb processor (PROC)	9-3
How ProVerb works.	9-3
Using sentences and paragraphs	9-5
Using menus	9-8

This chapter provides an overview of the following application development tools:

- The ProVerb processor (PROC)
- Stored sentences and paragraphs
- Menus

Pick users will already be familiar with the use of procs. Sentences, paragraphs, and menus, however, may be new to you. For complete information, see the *UniVerse System Description*.

The ProVerb processor (PROC)

The UniVerse ProVerb processor corresponds to the PROC processor on other Pick systems; it interprets a sequence of command statements stored in a proc (stored **procedure**).

UniVerse supports both standard Pick-style procs (PQ) and REALITY-style procs (PQN). As on other Pick systems, field 1 of a proc always contains a type code of either PQ or PQN. One of the main differences between PQ procs and PQN procs is that PQ procs use blanks as field delimiters in the input and output buffers, whereas PQN procs use field marks (for example, attribute marks). In UniVerse *all* UniVerse procs, both PQ and PQN, use field marks to delimit parameters in the buffers. This means that PQ and PQN procs can be used interchangeably on UniVerse. For example, a PQ proc can call a PQN proc and vice versa.

How ProVerb works

The ProVerb processor works pretty much the way the PROC processor works on other Pick systems: a sequence of stored ProVerb commands that define a procedure or a set of operations for building UniVerse commands is submitted to the UniVerse Command Processor (TCL). ProVerb uses the four familiar variable-length buffers: two for input (primary and secondary) and two for output (primary and secondary). There are also nine file buffers for accessing UniVerse files, and eleven select registers for processing select lists.

ProVerb procs do all the same things Pick procs do: relational testing of data, conditional and unconditional control transfers, transfers to local and external subroutines, arithmetic processing, display of the contents of a buffer for debugging purposes, and format of output for the terminal screen or printer.

One important difference between procs on generic Pick systems and procs in UniVerse is that user exits are implemented as UniVerse BASIC subroutines. This is because UniVerse does not support the Pick Assembler. See Appendix A for a list of user exit codes that have been rewritten as UniVerse BASIC subroutines.

Another difference that REALITY users should note is that the proc command X exits not to TCL but to the calling process. REALITY procs should therefore convert all X commands to Q (quit). In UniVerse the Q command exits to TCL. See the section titled, [Converting REALITY procs](#) in Chapter 2, “[Chapter 2: Understanding UniVerse accounts,](#)” for information about converting REALITY procs to UniVerse procs.

See the *UniVerse User Reference* for detailed information about ProVerb commands.

Using sentences and paragraphs

Using sentence stack commands can save you a lot of time. They also let you customize your VOC file by giving familiar names to tasks that you frequently do. By saving related sentences in a paragraph, you can automate some of the repetitive tasks of your application.

This section summarizes the use of sentences and paragraphs in the UniVerse environment. For detailed information, see the *UniVerse System Description*.

The .S command lets you store a copy of a sentence in your VOC file for later execution. The syntax for saving a sentence is

.S name [sentence#]

The name of the sentence must be a unique record ID in the VOC file. If you do not specify the number of the sentence to be saved, the .S command saves the first sentence in the stack. If the sentence you want to save is not in the first position, use the .R command to move it, or specify the number of its position on the stack after the sentence name in the .S command line.

In the following example of the .S command, the sentence to be saved is in the third position in the stack:

```
03  SORT SUN.MEMBER BY LNAME BY FNAME FNAME LNAME
02  ED DICT SUN.MEMBER FNAME
01  SORT ONLY SUN.MEMBER
```

The following .S command saves sentence 03 in the VOC file under the name ROSTER. It does not delete the original sentence 03 from the stack. ROSTER is the record ID of the sentence entry in the VOC file:

>.S ROSTER 3

You can also append keywords to a stored sentence at the time of execution. For example, if you enter the following, the keyword ID.SUPP is added to the sentence and the report does not display record IDs. The saved sentence itself is not changed.

>ROSTER ID.SUPP

The `.S` command can also save two or more sentences from the stack. These sentences make up an entry in the VOC file called a paragraph. The syntax for saving a group of sentences as a paragraph is

`.S name start# end#`

The sentences to be taken from the stack are specified as a range of sentences, using numbers *start#* to *end#*, as in the following example:

`>.S WEEKLY.REPORT 3 6`

This command stores sentences 3 through 6 as WEEKLY.REPORT in the VOC file.

To execute the paragraph, enter its name at the UniVerse prompt, just as you would to execute a sentence. Each sentence in the paragraph is executed in succession, beginning with the sentence that had the highest number in the stack. In the example, the first sentence to be executed is sentence 6, followed by sentence 5, and so on.

You can recall a sentence or paragraph without executing it with the `.R` command. This lets you make changes in the sentence or execute the sentences in a paragraph separately. The syntax for the `.R` command is:

`.R name`

name must be the record ID of a sentence or paragraph in the VOC file. A sentence is recalled to the first position in the sentence stack. A paragraph will occupy additional positions in the stack.

Paragraphs can also make use of the following:

- DATA statements
- In-line prompting
- IF...THEN GO statements
- Statement labels
- Comments
- LOOP...REPEAT control statements

A paragraph can also invoke another paragraph or a stored sentence. But be sure that the second paragraph does not create a loop by reflexively invoking the calling paragraph.

In addition to the .S command, you can also use the Editor to create sentences and paragraphs. Be sure to put "S" in field 1 to identify a sentence, and "PA" to identify a paragraph.

Using menus

Stored sentences and paragraphs help you eliminate repetitive typing tasks, save time, and automate your application. You can further automate your application by creating menus.

A menu contains a numbered list of actions to be done. The actions are names of commands, sentences, paragraphs, or other menus, that are stored as part of the menu. Unlike using stored sentences or paragraphs, however, you do not have to remember many sentence or paragraph names to use an option on a menu. To select an action, simply enter the number and the indicated operation begins.

UniVerse provides two ways to create menus: the Menu Maintenance processor and the UniVerse Editor. The Menu Maintenance processor uses a combination of menus and REVISE to create, change, print, and delete menus.

To create menus, first create one or more files for storing them. Then use the Menu Maintenance processor to create a menu description record in one of the menu files. This record defines

- The menu's title
- The text for each choice and, optionally, where on the screen to display each choice
- The command statement, sentence, or paragraph that corresponds to each choice
- A brief explanation of each choice
- The input that can be used to exit from the menu and return either to the calling process or to TCL.
- The text used to prompt for a selection number and, optionally, where on the screen to display this text.

Finally, use the Editor or the Menu Maintenance processor to create a menu entry in the VOC file that can be used as a command to invoke the menu.

For a complete description of how to create and use menus, see the *UniVerse System Description*.

Appendix A: User exit codes

UniVerse provides many standard user exits that are available on other Pick systems. You can also write your own BASIC subroutines. Appendix A describes user exits as they are implemented in UniVerse. This appendix explains

- what user exits are
- how to write one
- how to use it one

User exits are routines written in Pick assembler and referenced by the hexadecimal address of the location of the routine in the Pick monitor. UniVerse supports the references to user exits through UniVerse BASIC subroutines with a specialized name and calling convention. If your application uses a user exit that is not supplied, or if you have modified the way a standard user exit works, you must write a UniVerse BASIC routine to perform the desired function.

On a Pick system user exits are referenced by a name in the following format:

Unnnn

The U stands for “user exit,” and *nnnn* is a four-digit hexadecimal number that represents the address of the code. In UniVerse user exits are cataloged UniVerse BASIC programs. They are referenced in the same way as on other Pick systems. The *nnnn* in the name does not indicate the location of the code, of course, but it must be the same as the name used by the proc or conversion code that references it. The UniVerse BASIC routines that implement the user exit must be globally cataloged with a name in the format: *\$nnnn*.

Writing user exits

To create and use a user exit in UniVerse, do the following:

1. Write the UniVerse BASIC program that you wish to define as a user exit.
2. Catalog the program.
3. Execute the program by including its name in a proc, UniVerse BASIC program, or in the conversion field (field 3) of a file dictionary.

A user exit can be executed in two ways: by including it in a proc or by specifying it as a conversion in a UniVerse BASIC program (using OCONV or ICONV functions) or in a dictionary entry. We explain each of these methods, since the user exit must be written differently for each of them, even if they perform the same function.

The following sections also present two user exit examples: one is to be called from a proc, and the other is to be called from a BASIC program or a dictionary. The APP.PROGS file, located in the UV account, contains the predefined user exits provided by UniVerse. You can examine these to see how user exits are defined.

Writing user exits to be called from UniVerse BASIC programs

If you are planning to call a user exit from a UniVerse BASIC program or from a conversion, the user exit will have a different set of arguments than if you call the user exit from a proc. If the user exit is to be called from a UniVerse BASIC program, it must have the form:

```
SUBROUTINE Unnnnn (answer, status, data, type)
```

The following table describes the arguments to the subroutine.

Argument	Description
<i>answer</i>	The string or value returned by the user exit subroutine.
<i>data</i>	The input string or value to be converted or processed.
<i>status</i>	0 (zero) if the conversion succeeds. This valued is returned in the STATUS() variable in the BASIC program calling the user exit.
<i>type</i>	0 (zero) if OCONverting, 1 (one) if ICONverting.

User Exit Arguments

User exit U50MB

The following subroutine is an example of a user exit written to be called from a UniVerse BASIC program. It returns the user's account name and the terminal number. User exit U50MB is essentially the same as the predefined user exit U50BB (the program 50BB in the APP.PROGS file). For this example we simply modified U50BB to return the account name before the terminal number.

```
SUBROUTINE U50MB(ANS,STATUS,DATA,TYPE)
*****
*
*          50MB - WHO SUBROUTINE EXAMPLE
*
*****

$OPTIONS DEFAULT

ANS = @WHO:' ':@USERNO
STATUS = 0
RETURN
END
```

Writing user exits to be called from procs

If you create a user exit to be called from a proc, it must have the form:

```
SUBROUTINE Unnnn(proc, ibn, pib, sib, ip, obn, pob, sob)
```

The following table describes the arguments to the subroutine:

Argument	Description
<i>proc</i>	The text of the proc itself.
<i>ibn</i>	The current input buffer switch (0 = primary; 1 = secondary).
<i>pib</i>	The primary input buffer.
<i>sib</i>	The secondary input buffer.
<i>ip</i>	The input buffer pointer.
<i>obn</i>	The current output buffer switch (0 = primary; 1 = secondary).
<i>pob</i>	The primary output buffer.
<i>sob</i>	The secondary output buffer.

Subroutine Arguments

User exit U31MD

The following subroutine is an example of a user exit written to be called from a proc. It displays the user's terminal number on the screen, or returns it into the primary input buffer or current output buffer of a proc.

User exit U31MD is essentially the same as the predefined user exit U31AD (in the APP.PROGS file). For this example we simply modified user exit U31AD to return the terminal number without adding a P to the end of the number.

```

subroutine U31MD( proc, ibn, pib, sib, ip, obn, pob, sob )
*****
*
*       Retrieve nn into a proc, where nn is the terminal number
*
*****

* The REMOVE pointer is set at the line immediately following the
* user exit call in the proc. Therefore, this REMOVE statement
* sets target equal to the line immediately after the user exit
* call.

REMOVE target FROM proc SETTING x
target = TRIM( target )
answer = @USER.NO
IF LEN( answer ) < 3 THEN answer = "0" : answer

```

```

BEGIN CASE
  CASE target = "S"
    IF( obn ) THEN
      sob = answer
    END ELSE
      pob = answer
    END

  CASE target = "A"
    IF( obn ) THEN
      pob = answer
    END ELSE
      sob = answer
    END

  CASE target = "P"
    IF pib[ ip, 1 ] = @AM THEN ip += 1
    first.part = pib[ 1, ip - 1 ] : answer : @AM
    last.part = pib[ ip + 1, len( pib )]
    DEL last.part< 1 >
    pib = first.part : last.part

  CASE target = "T"
    print answer

  CASE target = "T+"
    print answer:

END CASE
RETURN
END

```

Cataloging user exits

Once you have written and compiled the UniVerse BASIC subroutine, you must catalog the program. User exits are cataloged globally by specifying the user exit name with a dollar sign in front of it. For example, to catalog the user exit U50MB in the example shown earlier, use the CATALOG command as follows:

```
>CATALOG
    CATALOG NAME or LOCAL = $50MB
    FILENAME = APP.PROGS
    PROGRAM NAME = 50MB
```

Or you can use the form:

```
>CATALOG APP.PROGS $50MB
```

Like any other program, a user exit must be compiled before you can catalog it. See Chapter 8, “[Chapter 8: UniVerse BASIC](#),” for more information about cataloging UniVerse BASIC programs.

Cataloging programs in PICK accounts

PICK and IN2 flavor accounts do not permit global cataloging; therefore you must either catalog the program in a different flavor account or create an INFORMATION flavor CATALOG entry in your PICK account. To create an INFORMATION flavor CATALOG entry, create a VOC entry which is the same as the entry for CATALOG, except that field 6 is defined as INFORMATION.FORMAT instead of PICK.FORMAT.

Executing user exits from UniVerse BASIC programs

The syntax for including a user exit in a BASIC program is

var = OCONV(*expression*, "U*nnnn*")

var is the variable containing the returned value.

nnnn is the user exit name.

expression is the value to be converted.

The following BASIC code fragment, which calls user exits U50BB and U50MB, illustrates the different output from each.

```
*
* Testing U50BB
PRINT "U50BB OUTPUT = PORT # AND ACCOUNT NAME"
* The value to be converted is specified as null (")
* since that value is meaningless for this user exit.
ANSWER = OCONV ( "","U50BB")
PRINT ANSWER
*
*
* Testing U50MB
PRINT "U50MB OUTPUT = ACCOUNT NAME AND PORT #"
ANSWER = OCONV ( "","U50MB")
PRINT ANSWER
*
```

It produces the following output:

```
U50BB OUTPUT = PORT # AND ACCOUNT NAME
65   uv
U50MB OUTPUT = ACCOUNT NAME AND PORT #
uv   65
```

Executing user exits from procs

The syntax for including a user exit in a proc is shown in the following example:

```
0001:  PQN proc containing user exit
0002:  Comment - proc commands can precede or follow user exits
0003:  Unnnn
0004:  target
0005:  normal return
```

nnnn is the name of the user exit.

target is one of the following single-letter codes, specifying where to send the output from the user exit:

- T Print on the terminal screen.
- P Place in primary input buffer.
- S Place in the current output buffer.
- A Place in the other output buffer.

The following proc executes user exits U31AD and U31MD and displays the output on the terminal.

```
0001:  PQN Show the terminal number
0002:  Comment Include a "P" after terminal number
0003:  U31AD
0004:  T
0005:  Comment Don't include the "P" after terminal number
0006:  U31MD
0007:  T
```

It produces the output:

```
65P
65
```

User exits called from ProVerb

You call the following user exits from procs.

- U0190 Perform arithmetic on reverse Polish string in current output buffer.
- U0192 Perform format operations.
- U01A2 Perform an Nway branch.
- U01A6 Position the cursor.
- U01AD Perform a file lookup.
- U11A2 Zero-pad the current input buffer.
- U2196 Return the user's port number into the primary input or active output buffer.
- U21A2 Delete values from current output buffer.
- U31AD Return the user's port number with a P appended to it into the primary input or active output buffer.
- U41AD Replace the current item in the primary input buffer.
- U61A2 Execute the HUSH verb.
- UA1A2 Clip back one parameter from the primary input buffer.

User exits called from UniVerse BASIC programs

You call the following user exits from UniVerse BASIC programs

- U0196 Replace system delimiters with tildes (~).
- U01BE Limit input to specified number of characters, then CR/LF.
- U029E Get a message from the ERRMSG file.
- U035A Validate input.
- U1114 Limit input to one character.
- U11BE Limit input to specified number of characters; no CR/LF.
- U20E0 Return the invoking TCL sentence.
- U307A Sleep until specified time.
- U30E0 Check to see whether a Select List is active.
- U407A Sleep a specified number of seconds.
- U50BB Return the port number, a space, and the account name.
- U5114 Accept input with a specified end-of-input character.
- U60E0 Return the page width.
- U70E0 Perform ECHO ON.
- U7201 Check for the presence of characters in the input buffer.
- U80E0 Perform ECHO OFF.
- U81F5 Return the user's port number.

User exits called from dictionaries

You call the following user exits from dictionaries.

- U201E Count the number of fields in the currently active record.
- U508E List the contents of the currently active record.

Alphanumeric list of user exits

This section lists the standard UniVerse user exits in alphanumeric order, describes the action of each user exit in detail, and gives the complete syntax for each.

U0190

This user exit, called from a proc, takes the string in the current output buffer (delimited on the left by an arbitrary character specified in the calling sequence, on the right by a ?), evaluates the entire string, deletes the string (including any delimiters), and sends the answer to the specified target. The syntax is

U0190 *left.delimiter target*

target can be one of the following:

- T terminal
- S current output buffer
- A alternate output buffer
- P primary input buffer

U0192

This user exit, called from a proc, performs one of two kinds of formatting. The first controls horizontal or vertical spacing; the second fetches (and optionally applies conversions to) values or outputs constant data. The syntax is

```
U0192
format.item
format.item
...
format.item
-> T
```

Parameter	Description
Xnn	places nn blanks on the current output line; if no nn is specified, U0192 produces 0 blanks.
Son	spaces nn lines after printing the current line; if no nn is specified, U0192 prints 1 linefeed.
P	places a form feed in the output, resets the current page and line counters, and blanks out the heading.
L	routes output to the printer instead of the terminal.

U0192 User Exit Parameters

format.items:

Vnn file.reference item.reference field.number.reference

reads the specified field and puts its value in the output line, optionally beginning at the *nn*th column of the output line. References can be explicit or indirect through the primary input buffer or either output buffer. If no *nn* is specified, U0192 produces 0 blanks.

**nn file.reference item.reference field.name.reference*

reads the named field, applies any conversions present in field 3 of the dictionary item, and puts the converted value in the output line, optionally beginning at the *nn*th column of the output line. References can be explicit or indirect through the primary input buffer or either of the output buffers.

Hnn string

places the string in the output line. If *nn* is present, the string begins in the *nn*th column of the output line, otherwise it begins in the next column.

Hnn %m

places the value located in the *m*th position of the primary input buffer into the output line. If *nn* is present, the string begins in the *nn*th column of the output line, otherwise it begins in the next column.

Hnn #m

places the value located in the *m*th position of the current output buffer into the output line. If *nn* is present, the string begins in the *nn*th column of the output line, otherwise it begins in the next column. –>T must be used to end the format sequence. It means “route to the terminal.” The T can be followed by an optional +, which suppresses the final carriage return / linefeed.

U0196

This user exit, called from a BASIC program, takes the data string and replaces its system delimiters with tildes (~). The syntax is:

$X = \text{OCONV}(n, \text{"U0196"})$

where *n* is the string to convert.

U01A2

This user exit, called from a proc, performs the equivalent of a BASIC "ON X GOTO LINE.1 LINE.2 LINE.3 LINE.4... LINE.N" The syntax is:

```
U01A2
index arg1 arg2
line 1
line 2
.
.
.
```

Control is passed to the line corresponding to the argument equal to the value of *index*.



U01A6

This user exit, called from a proc, takes a number pair (either or both of which may be indirect references) and positions the cursor to that column and row. B sounds the terminal bell. C clears the screen. X indicates the hexadecimal equivalent of an ASCII character. The syntax is:

```
U01A6
(col.reference, row.reference)
```

All references can be indirect through the buffers.

Note: *U01A6 is extremely slow. Using T positions the cursor from a proc much more efficiently.*

U01AD

This user exit, called from a proc, takes the file name, record ID, and translation field number specified, and returns the looked-up value to the named target. The syntax is:

```
U01AD
file.ref item.ref field.ref target
return here if error
return here if success
```

target can be one of the following:

Target	Description
T	terminal
S	current output buffer
A	alternate output buffer
P	primary input buffer
V	verify existence (produces no output)
VA	verify nonnull attribute

U01AD User Exit Targets

All references can be indirect through the buffers. Work files are not supported.



U01BE

This user exit, called from a UniVerse BASIC program, accepts a specified number of characters into a variable and sends the cursor to the leftmost column of the next line. The syntax is

$$X = \text{OCONV}(n, "U01BE")$$

n is the number of characters to accept.

Note: Replacing $X = \text{OCONV}(n, "U01BE")$ with $\text{INPUT } X, n_$ produces more efficient code.

U029E

This user exit gets a message from the ERRMSG file. It reads the message *msg#* in the ERRMSG file, and uses *par1*, *par2*,... as the parameters of the message. The syntax is:

$$\text{variable} = \text{ICONV}(\text{msg}\#\{\wedge\text{par1}\{\wedge\text{par2}\}\dots, "U029E")$$

\wedge is an attribute mark (field mark).

U035A

This user exit validates input by specifying how many characters and which characters are to be accepted. You can also specify an initial value to be displayed. The syntax is:

$$\text{variable} = \text{ICONV}(\text{length}\wedge\text{valid.chars}\wedge\text{value}, "U035A")$$

\wedge is an attribute mark (field mark).

length is the maximum input length.

valid.chars is a list of acceptable characters.

value is the initial value to be displayed. The variable returned by this user exit has the format:

$$\text{input.value}\wedge\text{valid.chars}\wedge\text{code}$$

code is either 0 (not modified) or 1 (modified). When entering input it is possible to move the cursor to the left with CTRL-Y, to the right with CTRL-X, to delete the character under the cursor with CTRL-B, and to insert a space before the cursor with CTRL-A.

U1114

This user exit accepts the input of one character from the keyboard. Input is requested as with the INPUT statement, but only one character can be input. Any character can be entered. The syntax is:

variable = ICONV(1,"U1114")

U11A2

This user exit, called from a proc, left zero-fills the string in the current input buffer until the number of digits requested is achieved. It returns the result to the current input buffer. The syntax is:

U11A2
requested.number.of.digits

U11BE

This user exit, called from a UniVerse BASIC program, accepts a specified number of characters into a variable and leaves the cursor in position after the last character entered. The syntax is:

X = OCONV(*n*,"U11BE")

n is the number of characters to accept.

Note: Replacing *X = OCONV(n,"U11BE")* with *INPUT X,n_*: produces more efficient code.



U201E

This routine, called from the conversion field of a dictionary entry, lists the number of fields in the currently active record. The syntax is:

```

    DICT FRED FOO
    001 D
    002 irrelevant
    003 U201E
    004 irrelevant
    005 irrelevant
    006 irrelevant
```

The following sentence lists the record IDs in the file FRED along with the number of fields in each record:

```

    LIST FRED FOO
```

U20E0

This user exit, called from UniVerse BASIC, returns the last statement issued at the TCL prompt. The syntax is:

```

    X = OCONV(n, "U20E0")
```

n is irrelevant.

Note: *X = @SENTENCE* is more efficient.



U2196

This user exit returns the user's terminal number either to the terminal, or to the primary input buffer or current output buffer. The syntax is:

```

    U2196
    target
```

target can be one of the following:

Target	Description
T	print on the terminal screen
P	place into primary input buffer
S	place into current output buffer
A	place into the other output buffer

U2196 User Exit Targets

U21A2

Delete values from the currently active output buffer. The syntax is:

U21A2
n

If *n* is 0, all values are deleted from the output buffer. If *n* is 1, the last value is deleted from the output buffer.

U307A

This subroutine, called from UniVerse BASIC, causes the program to sleep until the time specified in the DATA argument. The syntax is:

$X = \text{OCONV}(n, \text{"U307A"})$

n is the time to wake up, specified in any legal time format.

U30E0

This subroutine, called from UniVerse BASIC, returns a 1 if there is an active select list, a 0 if there is none. The syntax is:

$X = \text{OCONV}(n, \text{"30E0"})$

n is irrelevant.

Note: $X = \text{SYSTEM}(11)$ is more efficient.



U31AD

This user exit returns the user's terminal number in file name form (for example, with an uppercase P concatenated onto the end) either to the terminal or into the primary input buffer or current output buffer. The syntax is:

U31AD
target

target can be one of the following:

Target	Description
T	print on the terminal screen
P	place into primary input buffer
S	place into the current output buffer
A	place into the other output buffer

U31AD User Exit Targets

U407A

This subroutine, called from UniVerse BASIC, causes the program to sleep for the number of seconds specified. The syntax is:

X = OCONV (*n*, "U407A")

n is the number of seconds to sleep.

Note: SLEEP *n* is more efficient.



U41AD

This user exit replaces the current item in the primary input buffer with a character string from the line in the proc after the invocation. The string can contain blanks, which causes multiple items to be inserted, with the remainder right-shifted. The syntax is:

```
U41AD
string.to.insert
```

U508E

Called from the conversion field (field 3) of a dictionary entry, this routine prints the contents of all or specified records in a file in a form that resembles that of the UniVerse Editor. The syntax is

```
DICT FRED FOO
001 D
002 irrelevant
003 U508E
004 irrelevant
005 irrelevant
006 irrelevant
```

The following sentence list all records in the file FRED:

```
LIST FRED FOO
```

U50BB

This subroutine, called from UniVerse BASIC, returns the port number, a space, and the account name. The syntax is:

```
X = OCONV (n,"50BB")
```

n is irrelevant.

Note: X = @USERNO : " " : @WHO is more efficient



U5114

This user exit accepts a specified number of characters into a variable along with a specified end-of-input character. Control characters other than those in the following table are ignored.

ASCII	Hexadecimal	CTRL-Key
10	0A	^J
11	0B	^K
12	0C	^L
13	0D	^M
14	0E	^N
23	17	^W
26	1A	^Z

The syntax is:

variable = ICONV(*length*,"U5114")

length is the maximum input length excluding the end-of-input character.
length should not exceed 500 characters.

U60E0

This subroutine, called from UniVerse BASIC, returns the page width. The syntax is:

X = OCONV (*n*,"60E0")

n is irrelevant.

Note: *X* = SYSTEM(2) is more efficient.



U61A2

This user exit, called from a proc, executes the HUSH verb. The syntax is:

U61A2

U70E0

This subroutine, called from UniVerse BASIC, turns on the terminal's echoing of characters. The syntax is:

$X = \text{OCONV}(n, \text{"U70E0"})$

n is irrelevant.

Note: ECHO ON is more efficient.



U7201

Called from UniVerse BASIC, this subroutine returns a 1 if there are characters in the input buffer, a 0 if there is none. The syntax is:

$X = \text{OCONV}(n, \text{"U7201"})$

n is irrelevant.

Note: INPUT X, -1 is more efficient.



U80E0

This subroutine, called from UniVerse BASIC, turns off the terminal's echoing of characters. The syntax is:

$X = \text{OCONV}(n, \text{"U80E0"})$

n is irrelevant.

Note: ECHO OFF is more efficient.





U81F5

Called from UniVerse BASIC, this subroutine returns the user's port number. The syntax is:

`X = OCONV (n,"U81F5")`

Note: X = @USERNO is more efficient.

UA1A2

Called from a proc, this user exit moves the pointer in the primary input buffer back one parameter each time it is called. There are no arguments to this user exit. The syntax is:

`UA1A2`

Appendix B: Flavor-dependent commands

Implementing accounts of different flavors enables UniVerse to be compatible with various versions of the Pick system. Appendix B contains a list of commands in PICK, REALITY, and IN2 flavor accounts that are different from the corresponding commands in IDEAL flavor accounts. These differences are minor: certain commands in PICK, REALITY, and IN2 flavor accounts use variations of the syntax for commands in the IDEAL flavor. For the most part, these commands and keywords work as on other Pick systems.

The following commands are found on other Pick systems but are not supported by UniVerse. Where there is a UniVerse command that corresponds to the unsupported Pick command, this is indicated.

Pick Command	UniVerse Substitute
CHARGE-TO	
CHARGES	
COMPILE	BASIC
DEBUG	RAID
GROUP	GROUP.STAT.DETAIL
ISTAT	GROUP.STAT
ITEM	RECORD
LISTACC	

Pick Commands Not in UniVerse

Pick Command	UniVerse Substitute
LISTCONN	LISTK
LISTDICT	LIST.DICT
LISTPEQS	SPOOL -LIST
LISTPROCS	LISTPQ
LISTPTR	SPOOL -LIST
LISTVERBS	LISTV
MSG	MESSAGE
S-DUMP	SSELECT (to sort), then T-DUMP
SP-CLOSE	To close a print file kept open with the SETPTR KEEP option, use SETPTR with no options specified.
SP-KILL	SPOOL -CANCEL or PRINT.ADMIN
SP-OPEN	SETPTR with the KEEP option
SP-STATUS	SPOOL -LIST or PRINT.ADMIN
SP-TAPEOUT	SP.TAPE
WHAT	
WHERE	LISTU or STATUS USERS

Pick Commands Not in UniVerse (Continued)

The following pages show commands used in Pick and UniVerse. Additional features, and options unique to UniVerse are explained.

ACCOUNT.RESTORE

The ACCOUNT.RESTORE command loads a Pick account from any device or file, or from standard input, and creates the necessary files for a UniVerse account. It does not perform any conversion.

Syntax

ACCOUNT.RESTORE { *-a pathname* | *-t device* } [*-19*]

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>-a pathname</i>	read input from the specified UNIX pathname.
<i>-m</i>	use for Microdata Reality tapes.
<i>-n</i>	create files with a separation of 4, for ADDS Mentor or PICK 370, where a separation of 1 means 2K bytes.
<i>-t device</i>	read input from a tape device defined in the &DEVICE& file. <i>device</i> is the ID of the entry in the &DEVICE& file.
<i>-19</i>	create type 19 files instead of type 1 files.

ACCOUNT.RESTORE Parameters

Note: If you use the ACCOUNT.RESTORE command without arguments, the option *-t MT0* is used as the default.



Description

ACCOUNT.RESTORE executes the *acct.restore* UniVerse executable. The preferred method for restoring an account is to use the Account Importation option of the System Administration menus from the UV account. The System Administration menus invoke *acct.restore* with one or more filters that handle labels and multireel account-save tapes correctly. You can use ACCOUNT.RESTORE to restore most single reel account-save tapes that have supported label formats. See *Administering UniVerse* for a full discussion of account restoration procedures.

ACCOUNT.RESTORE creates a special file called PICK.VOC to hold the Pick Master Dictionary items. If there is a D/CODE of DC in attribute 1 of the D-pointer, ACCOUNT.RESTORE creates type 1 UniVerse files for storing Pick BASIC source programs. (When BASIC programs are brought over from systems that make no distinction between program source files and user data files, ACCOUNT.RESTORE does not automatically create nonhashed files; you must do this with the RESIZE command.) Each Pick file restored to UniVerse comprises a data file (*filename*) and a UniVerse file dictionary (*D_filename*).

To use the restored account, you must convert the Master Dictionary items, convert the dictionaries of all files, and convert and recompile all BASIC programs. This can be done using the Account Conversion menu, which is invoked by the CONVERT.ACCOUNT command.

ACCOUNT.RESTORE shortens record IDs for type 1 and type 19 files if they exceed the maximum length for file names. ACCOUNT.RESTORE breaks long record IDs of type 1 files into 14-character lengths, each of which becomes a subdirectory, until the record ID length is used up or the 41-character maximum length for record IDs is reached. Type 19 files are truncated to the machine-dependent maximum, taking into consideration any character conversion necessary for UNIX special characters. Machine-dependent maximums are 14 characters for System V UNIX, 247 characters for Berkeley UNIX, and varying lengths for UNIX look-alikes.

If a record already exists with this new truncated ID, the record ID is truncated by three more characters, and a three-digit sequence number is appended until a unique record ID is found. Since the VOC entry for this file used the original long name for its ID, the user operating in the UniVerse environment need not be aware of this truncation. The user operating at the UNIX level, however, may find the truncated names to be hard to remember.

ACCOUNT.RESTORE lists all ID truncations in the &TRUNCATED& file, located in the account being restored. If you use ACCOUNT.RESTORE to restore over existing accounts, and if you have kept the &TRUNCATED& file intact, ACCOUNT.RESTORE will map the long IDs to their truncated IDs correctly.

When ACCOUNT.RESTORE creates new UniVerse files, it looks at the VOC entry for CREATE.FILE in the directory from which it is being run, in order to determine whether or not to create long file names. In order to create long file names, the VOC entry for CREATE.FILE must contain a V in field 4 and the word LONGNAMES in field 5.

In PICK, REALITY, and IN2 accounts, the following parenthetical options can be specified. The right parenthesis is optional.

Option	Description
L	is used to produce a listing of the source code. The listing will be placed in <i>filename.L</i> . If the listing file does not exist, it is created.
P	causes a listing of the source code to be sent to the printer.
S	suppresses the symbol table and line number table usually appended to the object file for run-time error messages. This results in smaller object files. Note that run-time error messages will not know the line number or variable involved in the error.
X	causes the compiler to generate a cross-reference listing, showing the line numbers where each variable is referenced or assigned.

ACCOUNT.RESTORE Parenthetical Options

Example:

>BASIC BP PROG1 (P)

CATALOG

The CATALOG command catalogs a UniVerse BASIC program. In a PICK flavor account, as on most Pick systems, all cataloging is *local*.

Beginning at UniVerse 11.2.3, the ICATALOG command is available. The ICATALOG command is not flavor-dependent, and it is recommended that users use the ICATALOG command instead of the CATALOG command, when needing to globally catalog a program.

Syntax

CATALOG [*filename*] [*programs*] [NOXREF] [COMPLETE]

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>filename</i>	is the name of the file containing UniVerse BASIC programs. The source code must have been compiled before the CATALOG command is used. UniVerse assumes that the object code to be cataloged is in the corresponding object code file named <i>filename.O</i> .
<i>programs</i>	is the list of programs to be cataloged. An asterisk (*) specifies all programs in a file. CATALOG can also take program names from a Select List.
NOXREF	indicates that the program should be cataloged without the cross-reference and symbol table information. This makes it difficult to use any of the UniVerse debugging tools. Use this only when a program has been thoroughly tested. NOXREF cannot be specified at a prompt.
COMPLETE	indicates that the VOC entry for a locally cataloged program should be the absolute path. The VOC entry normally specifies the location of the program relative to the user's account.

CATALOG Parameters

If you do not specify *filename*, you cannot specify any other qualifiers on the command line. In this case, CATALOG prompts you for the qualifier values one at a time. If you enter a null reply to any of the prompts, CATALOG terminates without cataloging anything. NOXREF cannot be specified at a prompt.

Description

A program must be cataloged before it can be called as an external subroutine in a UniVerse BASIC program. A subroutine in the same file as the main program, however, need not be cataloged.

CATALOG creates a VOC entry for the catalog name that points to the file and the record that contain the object code for the cataloged program. A locally cataloged program can be accessed only from the account in which it was cataloged, unless you copy the VOC entry for the cataloged program to the VOC file of another account, changing field 2 of the copied VOC entry so that it contains the full UNIX path of the object code file. As on other Pick systems, locally cataloged programs need not be recataloged every time you recompile them.

The following example shows a subroutine being cataloged:

```
>CATALOG
File Name           = BP
Program Name        = TSUB1
"TSUB1" cataloged.
```

CHECK.SUM

Syntax

CHECK.SUM VOC (C)

In PICK, REALITY, and IN2 accounts, the following parenthetical options can be specified. The right parenthesis is optional.

Option	Description
C	The same as COL-HDR-SUPP.
D	The same as DET-SUPP.
H	The same as HDR-SUPP.
I	The same as ID-SUPP.
N	The same as NO.PAGE.
P	The same as LPTR.
F	Ignored.
B	Ignored.
G	Ignored.

CHECK.SUM Parenthetical Options

CONVERT.ACCOUNT

Use the CONVERT.ACCOUNT command to convert your Pick Master Dictionary or Prime VOC entries into a form usable in your UniVerse VOC file.

Syntax

CONVERT.ACCOUNT

Description

CONVERT.ACCOUNT invokes the Account Conversion menu, which provides two submenus, one for converting accounts, the other for converting UniVerse BASIC programs. The Dictionary Conversion submenu lists the following choices:

- **convert md or Voc.** Choose this option to convert your Pick Master Dictionary or Prime VOC file entries into a form usable in the UniVerse VOC file.
- **convert Dictionaries.** Choose this option to convert your Pick file dictionaries to UniVerse file dictionaries.
- **convert Procs.** Choose this option to convert procs from a REALITY system to a form compatible with the UniVerse ProVerb processor.

The UniVerse BASIC program conversion submenu lists the following choices:

- **convert basic program Files.** Choose this option to convert your UniVerse BASIC program files to UniVerse type 1 files. Your old object code records will be deleted.
- **compile Basic programs.** Choose this option to compile all of your UniVerse BASIC programs.
- **Catalog basic programs.** Choose this option to catalog all of your compiled UniVerse BASIC programs.
- **create a basic Makefile.** Choose this option to create a *makefile* for the account you are currently working in.

To choose an option, enter the mnemonic letter at the prompt, or use the cursor keys to move the highlighted selection bar to the option you want, then press **ENTER**.

CONVERT.VOC

Use the CONVERT.VOC command to convert Master Dictionary items and produce equivalent entries in the VOC file. The Pick Master Dictionary is neither changed nor deleted by the program. The CONVERT.VOC program is usually invoked from the Account Conversion menu displayed by the CONVERT.ACCOUNT command.

Syntax

CONVERT.VOC [FROM *filename*] [TO *filename*]

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
FROM <i>filename</i>	specifies the name of the Pick Master Directory or Prime VOC file that you want to convert. The default is PICK.VOC. If this source file is in Prime VOC format, its dictionary, DICT VOC, must exist as well.
TO <i>filename</i>	specifies the name of the file where the converted items are to be stored. The default is VOC. The destination file must already exist. The CONVERT.VOC program does not create the file.

CONVERT.VOC Parameters

Description

CONVERT.VOC asks the user to specify the type of Pick system on which the account originated. Verbs and connectives from the Pick Master Dictionary are converted to the nearest equivalent in UniVerse. D-pointers are converted to file definition entries, and Q-pointers are implemented as they are in Pick. Procs are copied without any changes. If any unconverted verbs or connectives are used in the PROC you must remove them or specify an equivalent in order for them to work properly.

Before creating a VOC file entry, CONVERT.VOC checks to make sure that it will not overwrite an existing record in the VOC file. If an existing record in the VOC is to be overwritten with a converted record, the converted record is compared to the existing record to determine if they are different. If there is a difference, the VOC file entry takes precedence and is not overwritten.

An error report is produced that lists the item IDs of any items that could not be converted.

COPY

The COPY command copies records to other records in the same file or to another file, or to the terminal.

Beginning at UniVerse 11.2.3, the ICOPY command is available. The ICOPY command is not flavor-dependent. It uses the IDEAL/INFORMATION flavor syntax for the COPY command.

Syntax

COPY [DICT] *filename records* [(*options*)

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
DICT	specifies the file dictionary.
<i>filename</i>	is the name of the file containing the records to be copied.
<i>records</i>	is the list of records to be copied. If there are more than one, they must be separated by blanks. An asterisk (*) specifies that all records are to be copied. If records are copied within the same file, the copy must be given a record ID different from that of the original record.

COPY Parameters

Description

UniVerse provides two versions of the COPY command. The COPY command in IDEAL flavor accounts is compatible with Prime INFORMATION's COPY command. The one in PICK, REALITY, and IN2 flavor accounts is compatible with the generic Pick version of COPY. This entry describes the COPY command found in PICK, REALITY, and IN2 flavor accounts.

The following parenthetical options can be specified in the command line. You can list as many as you want, either separated by commas or blanks, or not.

The following table describes the COPY display options.

Option	Description
F	Lists each item on a separate page.
N	suppresses automatic paging.
P	copy is sent to the printer.
S	suppresses line numbers.
T	copy is displayed on the terminal.
X	copies data in hexadecimal format.

COPY Display Options

The next table descriptions the COPY options.

Option	Description
C	lets you copy records into an SQL table with CHECK constraints. The OPENCHK configurable parameter must be set to 0 (false), otherwise the COPY command fails.
D	deletes the source record after it is copied.
G	suppresses automatic paging.
I	suppresses listing of record IDs.
N	prevents copying new records to the destination file unless the record already exists there. If the N option is chosen, no new records will be created.
O	overwrites existing records.
S	suppresses error messages.
X	copies data in hexadecimal format.

COPY Options

Note that the S and N options operate differently depending on whether the copy is to the terminal or printer, or to a record in a file.

If the copy is to be sent to a file, the COPY processor responds with:

TO:

The reply should be in the following syntax:

[([**DICT**] *filename* [)]] [*records*]

Parameter	Description
filename	If you are copying a record to a different file, the destination file name must be enclosed in parentheses. The right parenthesis is optional. Do not, however, omit the left parenthesis. If you do, the COPY processor interprets the file name as a record ID of a record in the same file rather than as a destination file name; the actual record ID, if specified, is ignored. If you are copying a record to other records in the same file, omit the file name.
records	If you are copying to a different file and the record IDs of the records to be copied are to be changed, or if you are copying to the same file, the list of new record IDs should be entered, separated by blanks. Any record ID that includes blanks must be enclosed in double quotes. If there is no change in the record IDs, the destination record list can be omitted.

Reply Parameters

If nothing is entered at the TO: prompt, the copy is displayed on the terminal as if the T option had been specified.

You cannot copy records into an SQL table with CHECK constraints if the OPENCHK configurable parameter is set to 1 (true). If OPENCHK is 0 (false), you must use the C option to copy records to such an SQL table.

Examples

```
>COPY SUN.MEMBER 7100
TO: 2300
    1 7100 copied to 2300 in file "SUN.MEMBER".

1 record copied.
```


>COPY SUN.MEMBER 2300

TO: (VOC)

1 2300 copied to 2300 in file "VOC".

1 record copied.

COPY.LIST

The COPY.LIST command copies Select Lists either within the &SAVEDLISTS& file or to another file, or to the terminal. UniVerse provides two versions of the COPY.LIST command. The COPY.LIST command in IDEAL flavor accounts is compatible with Prime INFORMATION's COPY.LIST command. The one in PICK, REALITY, and IN2 flavor accounts is compatible with the generic Pick version of COPY.LIST. This entry describes the COPY.LIST command found in PICK, REALITY, and IN2 flavor accounts.

Syntax

COPY.LIST *listnames* [(*options*)]

Parameters

The following tables describes each parameter of the syntax.

Parameter	Description
<i>listnames</i>	is the list of saved Select Lists to be copied. Saved Select Lists are records in a type 1 file called &SAVEDLISTS&. If you specify more than one, they must be separated by blanks. An asterisk (*) specifies that all lists are to be copied. If lists are copied to other lists in the &SAVEDLISTS& file, the copy must be given a list name different from that of the original list.
<i>options</i>	are the same as those for the COPY command.

COPY.LIST Parameters

If the copy is to be sent to a file, the COPY processor responds with:

TO:

The reply has the syntax:

[([DICT] *filename* [)]] [*listnames*]

filename is the name of the file to which you are copying the lists.

listnames are the names of the copied lists. The file name must be enclosed in parentheses.

The right parenthesis is optional.

If you do not specify a destination file, the list is copied to another list in the &SAVEDLISTS& file by default, and you must specify *listnames* different from the original list names to prevent the copies from overwriting the original lists.

If nothing is entered at the TO: prompt, the copy is displayed on the terminal as if the T option had been specified.

COPY.LIST is a proc that invokes the COPY processor. See the COPY command for a full list of parenthetical options that can be used with the COPY.LIST command.

Example:

```
>SELECT SUN.MEMBER WITH YR.JOIN EQ 1984

2 record(s) selected to SELECT list #0.
>>SAVE.LIST 1984
2 record(s) SAVED to SELECT list "1984".
>COPY.LIST 1984
TO: (SUN.MEMBER)
    1 1984 copied to 1984 in file "SUN.MEMBER".

1 record copied.
>
```

CREATE.BFILE

Use CREATE.BFILE to create a type 1 file for BASIC programs. CREATE.BFILE creates the file dictionary (a hashed file), the data file (a nonhashed type 1 file), and the file definition record in the VOC file. CREATE.BFILE can also be used to create multiple data files in the manner of CREATE.FILE. The specifications for the file dictionary can be specified either on the command line or in response to a prompting sequence.

Syntax

CREATE.BFILE { DICT | DATA } [*filename*] [*modulo*, [*separation,type*]] [*,description*]

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
DICT	specifies only the file dictionary.
DATA	specifies only the data file.
<i>filename</i>	is the UniVerse file name. CREATE.BFILE creates the data file <i>filename</i> and a file dictionary <i>D_filename</i> . If the file name contains special characters, CREATE.BFILE performs the following transformations before naming the files: <ul style="list-style-type: none">■ / becomes ?\■ ? becomes ??■ ASCII CHAR 0 (NUL) becomes ?0■ A leading period (.) becomes ?. If the file name is longer than 12 characters, a sequencing pattern is added to the end of the file name.

CREATE.BFILE Parameters

Parameter	Description
<i>modulo</i>	is an integer between 1 and 8,388,608 defining the number of groups for the dictionary. If you do not specify modulo on the command line, you are prompted for it.
<i>separation</i>	is an integer between 1 and 8,388,608 specifying the size of the group buffer for the dictionary in 512-byte blocks. If you do not specify separation, a default of 1 is used.
<i>type</i>	can be any file type for the file dictionary. If you do not specify type, the default type of 18 is used.
<i>description</i>	is any additional information about the file. This description is put in field 1 of the VOC file entry starting in the third character position (the first two positions are reserved for the type code, which for a file definition record is F). The description is displayed as HELP for the file name when you type <i>?filename</i> .

CREATE.BFILE Parameters

CREATE.FILE

The syntax of CREATE.FILE is slightly different in PICK, REALITY, and IN2 flavor accounts from that used in INFORMATION and IDEAL flavor accounts. In PICK, REALITY, and IN2, the syntax is:

```
CREATE.FILE { DICT | DATA } filename [ ,datafile ] [ dict.modulo  
[ ,dict.separation [ ,dict.type ] ] ] [ data.modulo [ ,data.separation  
[ ,data.type ] ] ] [ description ]
```

The main difference in syntax has to do with which parameter is specified first in the command line after the file name. In PICK, REALITY, and IN2 flavor accounts, the *modulos*, which are the only required parameters, must be specified first, followed by the separation and file type (if specified). In INFORMATION and IDEAL flavor accounts, the file type must be specified first, followed by the modulo and separation.

If the modulo is not specified, you are prompted for it. If the separation is not specified, a default separation of 1 is used. If the file type is not specified, a default type of 18 is used. Type 18 files use the same hashing algorithm as other Pick systems.

Beginning at UniVerse 11.2.3, the ICREATE.FILE command is available. The ICREATE.FILE command is not flavor-dependent. It uses the IDEAL/INFORMATION flavor syntax for the CREATE.FILE command.

DC

Use the DC command to convert some or all records in a Pick dictionary to their equivalents in a UniVerse file dictionary. DC is usually invoked from the Account Conversion menu displayed by the CONVERT.ACCOUNT command.

Syntax

DC [*-options*] [*filename* [*records*]]

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>options</i>	See table below.
<i>filename</i>	is the name of the file whose dictionary is to be converted.
<i>records</i>	is the list of record IDs of the records to be converted. Separate record IDs with spaces.

DC Parameters

options can be one or more of the following specified in any order:

Option	Description
L	sends output to the printer.
M	specifies that the dictionaries are in REALITY format.
N	does not send output to the system printer.
O	specifies that the dictionaries are not in REALITY format.

DC Options

Option	Description
P	specifies that you intend to maintain Pick style dictionaries.
U	specifies that you intend to maintain UniVerse style dictionaries.
Y	sends output to the printer.

DC Options (Continued)

If you do not enter the *filename* and *records* on the command line, UniVerse prompts you for them.

If you enter DC with no arguments and there is no active select list, the following prompt appears:

```
Enter name of dictionary to convert, or '*' for all local
dictionaries?
```

Entering a file name converts a single dictionary and sends a report to the printer. Entering an asterisk(*) builds a select list of all local dictionaries and prompts you to approve each dictionary before converting it. It also sends a report to the printer. The DC command also accepts a select list of file names of dictionaries to be converted.

When a file is restored from an account on a Pick system to UniVerse, the DC command places the original Pick dictionary records in a special file called the Pick dictionary. You can list the Pick dictionary by using the PDICT keyword instead of the DICT keyword in a Retrieve command.

The dictionary converter reads in the records in the Pick dictionary and copies them to the UniVerse dictionary. After all the records have been copied, DC builds an @ phrase and invokes CD (the dictionary compiler) on the UniVerse file.

If the dictionary converter encounters any items it cannot convert, it prints the following message:

```
Error listing written to file "&SAVEDLISTS&" item
"DC.ERRORS_####_####".
```

is the time in internal format and *####* is the date in internal format.

This record contains just the file and record ID pairs that caused it trouble. You may want to examine the record with the Editor.

Once the Pick dictionary is converted to a UniVerse file dictionary, you should decide whether or not you want to maintain the Pick dictionary. The only reason for maintaining the Pick dictionary is if you have programs that refer to it with the PDICT keyword. You should change those programs to directly reference the UniVerse dictionary with the DICT keyword. A Pick dictionary requires additional processing to maintain, and it is not as versatile or efficient as a UniVerse dictionary.

The special file dictionary DICT.PICK, corresponding to DICT.DICT for UniVerse dictionaries, can be used to provide a Pick-like dictionary listing. Use it with the USING keyword, as in the following example:

```
>LIST DICT MYFILE USING DICT.PICK
```

If you decide to maintain the UniVerse dictionary, all future modifications should be made directly in the UniVerse file dictionary. All Pick dictionaries can be deleted from the account.

If you decide to maintain Pick dictionaries, then all future modifications should be made in the Pick dictionaries. Do *not* delete the UniVerse file dictionaries.

Do not make changes in both the Pick dictionary and the UniVerse dictionary, since it is likely that the converter will overwrite the change. Also, be aware that the conversion program must be executed before the dictionary items can be accessed by any UniVerse processor such as Retrieve. When REVISE or the Editor is used to make changes, it keeps track of the IDs of modified Pick dictionary records and invokes the converter at the end of the process. However, if changes are made with a proc or a UniVerse BASIC program, you must invoke the dictionary conversion program to make the changes effective.

DECATALOG

Use the DECATALOG program to delete a locally cataloged program. It deletes the object code in addition to removing the catalog entry from the user's VOC file. In a REALITY flavor account, the DECATALOG command has no use since all programs are normally cataloged.

Syntax

DECATALOG *filename* [*programs*]

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>filename</i>	is the name of the file from which the object code of <i>program</i> are to be deleted.
<i>programs</i>	are the names of the programs whose object code is to be deleted from <i>filename.O</i> . Use an asterisk (*) in place of <i>programs</i> to indicate all records in the file.

DECATALOG Parameters

In PICK and IN2 flavor accounts, a list of program names can be supplied, separated by spaces. The DECATALOG command accepts a select list to specify the programs to be decataloged.

In PICK, REALITY, and IN2 accounts, the following parenthetical options can be specified. The right parenthesis is optional.

Option	Description
#	is the same as SAMPLE.
A	is the same as ALL.MATCH.
B	is ignored.

DECATALOG Parenthetical Options For Pick, Reality, and IN2 Flavors

Option	Description
C	is the same as COL-HDR-SUPP.
D	is the same as DET-SUPP.
F	is ignored.
G	is ignored.
H	is the same as HDR-SUPP.
I	is the same as SQUAWK.
L	is the same as EXPLODE.
N	is the same as NO.MATCH.
P	is the same as LPTR.
S	is the same as NO.SELECT.
T	is ignored.

DECATALOG Parenthetical Options For Pick, Reality, and IN2 Flavors (Continued)

In PICK, REALITY, and IN2 accounts, the following parenthetical options can be specified. The right parenthesis is optional.

Option	Description
N	specifies that automatic paging should be suppressed.
P	is used to print the report on the system printer.

Parenthetical Options

GET.LIST

Use the GET.LIST command to activate a saved select list, making it available to UniVerse BASIC READNEXT statements, control level commands, data management commands, and Retrieve commands.

Syntax

GET.LIST [*filename*] [*listname*] [TO *n*]

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>filename</i>	is the name of the type 1 or type 19 file containing the list you want to recall. If you do not enter a file name, the file &SAVEDLISTS& is assumed.
<i>listname</i>	is the name of the list to be recalled. If you do not enter a list name, GET.LIST tries to recall a list with the record ID &TEMP <i>port#</i> &, where <i>port#</i> is the one- or two-digit identifier of the terminal you are logged in on.
TO <i>n</i>	is the select list number to assign the list to. If <i>n</i> is not specified, Select List 0 is used.

GET.LIST Parameters

When GET.LIST is executed, a message is displayed confirming that the list is active.

Example:

```
>SELECT SUN.MEMBER WITH LNAME EQ WILLIAMS TO 1

2 record(s) selected to SELECT list #1.
>>SAVE.LIST WILLIAMS FROM 1

2 record(s) SAVED to SELECT list "WILLIAMS".
>GET.LIST WILLIAMS

2 record(s) selected to SELECT list #0.
```

>>**LIST SUN.MEMBER**

LIST SUN.MEMBER 11:23:16am 03-18-91 PAGE 1

SUN.MEMBER	FIRST NAME	LAST NAME	YEAR JOINED	INTERESTS
7100	ALICE	WILLIAMS	1984	HANG-GLIDING WINDSURFING
4309	EDGAR	WILLIAMS	1984	FISHING SAILING

2 records listed

LOGOFF

Use the LOGOFF command without an argument to log out of UniVerse. Use LOGOFF with a process ID number to terminate a phantom process. LOGOFF is a synonym for the UniVerse LOGOUT command. You can also use the shorter synonym LO to log off.

Syntax

LOGOFF [*process.ID.#*]

The user login name for the process ID number must be the same as your login name.

When you log off, UniVerse closes all open files, releases all assigned devices, and discards the current sentence stack (unless the STACKWRITE record in the VOC file is ON).

PRINT.ERR

Use the PRINT.ERR command to display one or more records from the ERRMSG file.

Syntax

PRINT.ERR [*records* | *]

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>records</i>	Specifies explicitly which records are to be displayed. You can specify as many records as you want, separated by spaces. PRINT.ERR also accepts a Select List to specify ERRMSG records.
*	Specifies all items in the ERRMSG file.

PRINT.ERR Parameters

Note: Do not confuse the PRINT.ERR command with the BASIC statement PRINTERR.



QSELECT

PICK, REALITY, and IN2 flavor accounts accept the following syntax for QSELECT:

Syntax

QSELECT [DICT] *filename* [*records*] [(*field.#* [)]

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
<i>filename</i>	is the name of a UniVerse file.
<i>records</i>	is a list of record IDs. An asterisk (*) specifies all records.
<i>field.#</i>	is the number (AMC) of one field whose values you want to store as elements in the select list. Multivalues are stored as separate elements in the select list. If a field number is not specified, QSELECT selects all values (and subvalues) from all fields of all specified records in the file.

QSELECT Parameters

REFORMAT

The REFORMAT command differs somewhat from the one found on other Pick systems. The *field* specification is required: that is, if no output fields are specified, nothing is reformatted. REFORMAT does not use the default output specified by the @ phrase in the file dictionary, nor does it output only record IDs if there is no output specification.

At the File Name = prompt, you cannot specify the name of the source file: that is, you cannot reformat a file back onto itself (as you can on other Pick systems).

You cannot use the BY.EXP and BY.EXP.DSND keywords to reformat multivalued records into separate singlevalued records, one record per line of an exploded sort, as you can on other Pick systems.

When reformatting to tape, tape labels are written in PICK format in PICK and IN2 flavor accounts. They are written in REALITY format in REALITY flavor accounts.

In PICK, REALITY, and IN2 flavor accounts, the following parenthetical options can be specified. The right parenthesis is optional.

REFORMAT VOC FILE.NAME (P)

Option	Description
C	is the same as COL-HDR-SUPP.
D	is the same as DET-SUPP.
H	is the same as HDR-SUPP.
I	is ignored.
N	is the same as NO.PAGE.
P	is the same as LPTR.
F	is ignored.
B	is ignored.
G	is ignored.

REFORMAT Parenthetical Options

RUN

The syntax for the RUN command is:

RUN [*filename*] *programs* [(*options*[*)*]]

The file name is optional. If you do not specify the name of a program file, the file BP is used by default.

In PICK, REALITY, and IN2 accounts, the following parenthetical options can be specified. The right parenthesis is optional.

Option	Description
A	is ignored.
E	aborts to debugger instead of printing warning (nonfatal) error messages.
H	suppresses output of tape labels.
I	suppresses initialization of unnamed common.
N	suppresses automatic paging.
P	sends program output to the printer.
S	suppresses all warning (nonfatal) error messages.
T	(REALITY flavor only) suppresses output of tape labels.

RUN Parenthetical Options

RAID

There is no D option. Use the RAID command to enter the debugger before running a BASIC program.

In PICK, REALITY, and IN2 accounts, the following parenthetical options can be specified. The right parenthesis is optional.

Option	Description
#	is the same as SAMPLE.
A	is the same as ALL.MATCH.
B	is ignored.
C	is the same as COL-HDR-SUPP.
D	is the same as DET-SUPP.
F	is ignored.
G	is ignored.
H	is the same as HDR-SUPP.
I	is the same as SQUAWK.
L	is the same as EXPLODE.
N	is the same as NO.MATCH.
P	is the same as LPTR.
S	is the same as NO.SELECT.
T	is ignored.

RAID Parenthetical Options

SELECT

In PICK, REALITY, and IN2 accounts, the SELECT command accepts standard Pick syntax:

```
SELECT [ DICT ] filename [ records ] [ selection ] [ sort ] [ output ]  
[ (options[ ]) ]
```

Pick users should note that sort expressions can be used with the SELECT command.

If output fields are specified, each value and subvalue becomes a separate element in the select list. You can also create a select list from data values using the SAVING keyword. If you using a SAVING expression on a multivalued field, however, the multivalues in each field are stored as a single element in the select list, with subvalues separated by value marks.

In PICK, REALITY, and IN2 accounts, the following parenthetical options can be specified. The right parenthesis is optional.

Option	Description
#	is the same as SAMPLE.
C	is the same as COL-HDR-SUPP.
D	is the same as DET-SUPP.
H	is the same as HDR-SUPP.
I	is the same as ID-SUPP.
N	is the same as NO.PAGE.
P	is the same as LPTR.
F	is ignored.
B	is ignored.
G	is ignored.

SELECT Parenthetical Options

SET.FILE

The UniVerse SET.FILE command allows you to explicitly name the Q-pointer in the command line. It does not create a VOC entry called QFILE by default.

Syntax

SET.FILE [*account*] [*filename*] [*pointer.ID*]

If you enter SET.FILE with no arguments, you are prompted for them. For example:

```
>SET.FILE
Q name: CUSTOMERS
Account: ACCOUNTS.REC
File: CUSTOMERS
Q-pointer written to VOC file.
```

If you want SET.FILE to create a Q-pointer whose record ID is QFILE, omit the pointer name in the command line. At the name prompt, press ENTER. For example:

```
>SET.FILE PERSONNEL PAYROLL
Q name: <RETURN>
Q-pointer written to VOC file.
>
```

Options

In PICK, REALITY, and IN2 accounts, the following parenthetical options can be specified.

Option	Description
C	is the same as COL-HDR-SUPP.
D	is the same as DET-SUPP.
H	is the same as HDR-SUPP.
I	is the same as ID-SUPP.

SET.FILE Parenthetical Options

Option	Description
N	is the same as NO.PAGE.
P	is the same as LPTR.
F	is the same as FORM.FEED.
B	is ignored.
G	is ignored.

SET.FILE Parenthetical Options (Continued)

SREFORMAT

The SREFORMAT command differs somewhat from the one found on other Pick systems. The *field* specification is required: that is, if no output fields are specified, nothing is reformatted. SREFORMAT does not use the default output specified by the @ phrase in the file dictionary, nor does it output only record IDs if there is no output specification.

At the File Name = prompt, you cannot specify the name of the file specified in the SREFORMAT command. You cannot reformat a file back onto itself (as you can on other Pick systems).

You cannot use the BY.EXP and BY.EXP.DSND keywords to reformat multivalued records into separate singlevalued records, one record per line of an exploded sort, as you can on other Pick systems.

When reformatting to tape, tape labels are written in PICK format in PICK and IN2 flavor accounts. They are written in REALITY format in REALITY flavor accounts.

In PICK, REALITY, and IN2 flavor accounts, the following parenthetical options can be specified. The right parenthesis is optional.

Option	Description
C	is the same as COL-HDR-SUPP.
D	is the same as DET-SUPP.
H	is the same as HDR-SUPP.