Rocket UniVerse

Guide to RetrieVe

Version 12.1.1

June 2019
UNV-1211-RETR-1

Notices

Edition

Publication date: June 2019
Book number: UNV-1211-RETR-1
Product version: Version 12.1.1

Copyright

Trademarks

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

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.

Corporate information

Rocket Software, Inc. develops enterprise infrastructure products in four key areas: storage, networks, and compliance; database servers and tools; business information and analytics; and application development, integration, and modernization.

Website: www.rocketsoftware.com

Rocket Global Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451-1468
USA

To contact Rocket Software by telephone for any reason, including obtaining pre-sales information and technical support, use one of the following telephone numbers.

Country	Toll-free telephone number
United States	1-855-577-4323
Australia	1-800-823-405
Belgium	0800-266-65
Canada	1-855-577-4323
China	400-120-9242
France	08-05-08-05-62
Germany	0800-180-0882
Italy	800-878-295
Japan	0800-170-5464
Netherlands	0-800-022-2961
New Zealand	0800-003210
South Africa	0-800-980-818
United Kingdom	0800-520-0439

Contacting Technical Support

The Rocket Community is the primary method of obtaining support. If you have current support and maintenance agreements with Rocket Software, you can access the Rocket Community and report a problem, download an update, or read answers to FAQs. To log in to the Rocket Community or to request a Rocket Community account, go to www.rocketsoftware.com/support.

In addition to using the Rocket Community to obtain support, you can use one of the telephone numbers that are listed above or send an email to support@rocketsoftware.com.

Contents

Chapter 1: Introduction

This chapter describes simple queries you can perform with RetrieVe, the UniVerse query language that you can use to select, sort, and display information from your UniVerse databases. This simple and understandable language allows application developers to perform complex tasks easily while allowing end users to construct queries with minimum knowledge.

What is RetrieVe?

You can enter RetrieVe commands interactively, directly from your terminal, or embed RetrieVe commands in application programs, procs, and paragraphs to access data in UniVerse files and SQL tables. Although RetrieVe queries are part of UniVerse, they provide additional capabilities unavailable in other UniVerse commands. Using the RetrieVe syntax effectively, you can search the database to extract a subset of data that meets certain qualifications. Once you have retrieved the information you want, you can then use other (non-RetrieVe) UniVerse commands to process the subset of data.

Suppose you run a retail operation and you want to track your inventory. To do this, you maintain a UniVerse database that stores the product identification code, description, wholesale cost, selling price, and the number in stock for each item you carry. Using RetrieVe commands against this simple database, you could obtain a variety of information vital to conducting your business. For example, you might ask for a listing of:

- Items that are low in stock
- Items with a markup of more than 50%
- Items of a particular type
- High-ticket items
- Average markup across the board

Additionally, you can choose how this information is presented. Do you want it displayed at your terminal or output on the printer? Would you prefer to have detailed or summarized information? Would you like the listing sorted in some fashion, and should the order be ascending or descending? For instance, you can tailor a report to take advantage of a 132-character line printer, obtain total inventory value by product line, or ask for a product listing arranged in order from highest to lowest markup.

Using RetrieVe

You can use RetrieVe to generate displays and reports from a database, entering your queries as English-like sentences at the UniVerse prompt (>).

`LIST` is a common RetrieVe command that retrieves information from a file. Its simplest form is the word `LIST` followed by a file name. To get a listing of a file called INVENTORY.F, enter:

```
 >LIST
INVENTORY.F
LIST INVENTORY.F 09:40:35AM 31 May 1995    PAGE    1
ITEM.CODE   TYPE    DESCRIPTION.............     COST......
   PRICE.....
14    V      Ice Cream, Various             $80.78
   $99.36
16    R      French Fries, Frozen           $34.95
   $45.78
17    U      Nachos                         $28.61
```

```
   $42.06
.
.
.
45 records listed.
```

Because you did not name any particular fields in the query, you got a default output. Also, because you specified no selection criteria, you get all of the records in the file.

But what if you wanted to see only certain records, and only certain fields in those records? As an example, you might ask to see the quantity on hand (QOH) and a description for those products in inventory with a QOH greater than 100:

```
 >LIST INVENTORY.F QOH DESCRIPTION WITH QOH > 100

LIST INVENTORY.F QOH DESCRIPTION WITH QOH > 100 08:55:12AM 31 May
1995  PAGE      1
INVENTORY.F    QOH..    DESCRIPTION.............
14                       154    Ice Cream, Various
17                       140    Nachos
2                        102    Cotton Candy
28                       174    Cookies
29                       158    Paper Plates
.
.
.
32 records listed.
```

To narrow your focus, you might refine the previous query to find all products with a QOH over 100 and with a cost greater than $75:

```
>LIST INVENTORY.F DESCRIPTION QOH COST WITH QOH > 100
  AND COST > $75

LIST INVENTORY.F DESCRIPTION QOH COST WITH QOH > 100 AND COST >
$75 09:02:03AM 31 May 1995  PAGE    1
INVENTORY.F    DESCRIPTION.............    QOH..    COST......

14             Ice Cream, Various            154       $80.78
28             Cookies                       174       $98.32
1              Beer                          127       $76.92
10             Franks                        151       $99.92
12             Mustard                       125       $91.52
25             Pretzels                      135       $87.22
31             Programs                      143       $79.78
37             Dog Chow                      131       $96.36
42             Cheese Slices                 169       $88.21
43             Sawdust                       181       $90.48
21             Sea Snails                    154       $91.17
5              Cola                          185      $102.83

12 records listed.
```

The list is not in any particular order, so you again refine the command to present the records in order by description:

```
 >LIST INVENTORY.F DESCRIPTION QOH COST WITH QOH > 100
        AND COST > $75 BY
```

```
DESCRIPTION
LIST INVENTORY.F DESCRIPTION QOH COST WITH QOH >   100 AND COST >
$75 BY DESCRIPTION 09:03:43AM  31 May 1995  PAGE    1
INVENTORY.F   DESCRIPTION.............     QOH..   COST......

1             Beer                          127    $76.92
42            Cheese Slices                 169    $88.21
5             Cola                          185    $102.83
28            Cookies                       174    $98.32
37            Dog Chow                      131    $96.36
10            Franks                        151    $99.92
14            Ice Cream, Various            154    $80.78
12            Mustard                       125    $91.52
25            Pretzels                      135    $87.22
31            Programs                      143    $79.78
43            Sawdust                       181    $90.48
21            Sea Snails                    154    $91.17

12 records listed.
```

These examples (and the remaining examples in this manual) use a database representing the operative business data for a traveling circus. For a description of this Circus database and the files it contains, see the next section and The sample database, on page 169.

Two unique attributes, 9998 and 9999, are available for backwards compatibility purposes with other multivalue databases.

- 9998 is a special attribute identifier used to produce a sequential counter for items being processed by the Retrieve sentence.
- 9999 is a special attribute-defining function that returns the size (in bytes) of the item being processed by Retrieve.

The sample database

UniVerse comes with a sample database called Circus that you can use to explore many of the features of RetrieVe.

This sample database reflects the activities of a traveling circus and consists of 10 UniVerse files, which are described in The sample database, on page 169.

Circus can be installed as either an account of UniVerse files or as a schema of SQL tables. The two forms of this data are distinguished from one another by a suffix in the name: .F identifies the UniVerse file version, and .T identifies the SQL table version. Thus, INVENTORY.F is the UniVerse file version of the inventory data, and INVENTORY.T is the SQL table version.

Examples in this manual use the UniVerse file version of the database. Remember that you can issue RetrieVe commands against SQL tables, and you can issue SQL statements against UniVerse files. However, the results may look slightly different, depending on which you use.

Installing the sample database

The following UniVerse commands generate and remove the Circus database:

Command	Action
MAKE.DEMO.FILES	Creates and loads the Circus database files into the current account. The files must not already exist in this account. The file names have an .F suffix, and the contents of the files are the same as those of the corresponding .T tables.
REMOVE.DEMO.FILES	Deletes the Circus database files from the current account.
SETUP.DEMO.SCHEMA *username*	Registers *username* as an SQL user (if not one already) and makes the current UniVerse account into a schema called DEMO_*username* which is owned by *username*. This command can be run only by an SQL user who is a DBA.
MAKE.DEMO.TABLES	Creates and loads the Circus database table into the current account, making the current user the owner of the tables. The user must be a registered SQL user, the account must be an SQL schema, and the tables must not already exist in this schema. The resultant tables have a .T suffix.
REMOVE.DEMO.TABLES	Drops the Circus database tables from the current schema. The user must be either a registered SQL user who is the owner of the tables or a DBA.

For example, to create and load the Circus database as UniVerse files into your current account, enter:

```
>MAKE.DEMO.FILES

Creating file "LOCATIONS.F" as Type 2, Modulo 2, Separation 2.
Creating file "D_LOCATIONS.F" as Type 3, Modulo 1, Separation 2.

Creating file "LIVESTOCK.F" as Type 2, Modulo 2, Separation 2.
Creating file "D_LIVESTOCK.F" as Type 3, Modulo 1, Separation 2.
.
.
.
WARNING: A file will be created with a truncated name.
Creating file "CONCESSIO0000" as Type 2, Modulo 2, Separation 2.
WARNING: A file will be created with a truncated name.
Creating file "D_CONCESSIO0000" as Type 3, Modulo 1, Separation 2.

WARNING: A file will be created with a truncated name.
Creating file "ENGAGEMEN000" as Type 2, Modulo 2, Separation 2.
WARNING: A file will be created with a truncated name.
Creating file "D_ENGAGEMEN000" as Type 3, Modulo 1,Separation 2.

Compiling "ANIMALS".
TRANS ( LIVESTOCK.F , ANIMAL.ID , NAME , X )
Compiling "EQUIPMENT".
TRANS ( EQUIPMENT.F , EQUIP.CODE , DESCRIPTION , X )
Compiling "OP.NAME".
TRANS ( PERSONNEL.F , OPERATOR , NAME , X )
Compiling "ANIMALS".
.
.
.
All demo files initialized.
```

Note the warnings about the file names for CONCESSIONS and ENGAGEMENTS being truncated. This applies only to their names within the operating system file system, not to their UniVerse file names, so within UniVerse, you can still reference them as CONCESSIONS.F and ENGAGEMENTS.F.

Uninstalling the sample database

To uninstall the database, use either `REMOVE.DEMO.FILES` (if the database is the UniVerse file version) or `REMOVE.DEMO.TABLES` (if the database is the UniVerse SQL table version).

For example, to remove the UniVerse file version of the Circus database, enter:

```
>REMOVE.DEMO.FILES

DELETEd file "ACTS.F", Type 2, Modulo 2.
DELETEd file "D_ACTS.F", Type 3, Modulo 1.
DELETEd file definition record "ACTS.F" in the VOC file.
.
.
.
DELETEd file "VENDORS.F", Type 2, Modulo 2.
DELETEd file "D_VENDORS.F", Type 3, Modulo 1.
DELETEd file definition record "VENDORS.F" in the VOC file.

DELETEd file "ENGAGEMEN000", Type 2, Modulo 2.
DELETEd file "D_ENGAGEMEN000", Type 3, Modulo 1.
DELETEd file definition record "ENGAGEMENTS.F" in the VOC file.

All demo files removed.
```

Multivalued fields

Before getting deeper into the use of RetrieVe, you should be familiar with a unique UniVerse attribute, multivalued fields. UniVerse uses a three-dimensional file structure, called a nonfirst-normal-form data model, to store multiple values for a field in a single record. Such fields are known as multivalued fields. Multivalued fields enable a record to contain information that would otherwise be scattered among several interrelated files.

UniVerse allows two or more multivalued fields to be defined in the file dictionary as being associated with one another, in such a way that the first value of the first multivalued field is associated with the first value of another field, the second value of one with the second value of the other, and so forth. Such associations are useful in situations where a group of multivalued fields forms an array or nested table within a file. Each association has an association name.

The sample database uses many multivalued fields to store information such as vaccination types and dates for each animal, information about the dependents of each employee, and the staff and animals used in each act.

Most of these multivalued fields are defined as belonging to associations, in which the first value in one multivalued field is related to the first value in each of the other multivalued fields in the association, the second value is related to all the other second values, and so forth. An example of an association in the sample database would be VAC.ASSOC, which is a vaccination array in which each vaccination type is related to a vaccination date, a next-scheduled vaccination date, and a vaccination certificate number.

How multivalued fields are displayed

When you ask for a multivalued field as part of your list, each value in that field is displayed or printed on a separate line. For example, the INVENTORY.F file has two associated multivalued columns,

VENDOR.CODE and ORDER.QTY, which store the vendor sources and quantities ordered from each vendor. Asking for a list of these values produces the following output:

```
>LIST INVENTORY.F DESCRIPTION VENDOR.CODE ORDER.QTY

LIST INVENTORY.F DESCRIPTION VENDOR.CODE ORDER.QTY 10:04
May 1995  PAGE    1
INVENTORY.F    DESCRIPTION.............    VENDOR.CODE
ORDER.QTY

14             Ice Cream, Various                  140
500
                                                    95
500
                                                    85
100
                                                   228
700
                                                   184
100
                                                   227
500
                                                    12
500
                                                    58
900
16             French Fries, Frozen                116
600
17             Nachos                              116
200
                                                    83
900
                                                   105
800
28             Cookies                              38
500
                                                    69
600
29             Paper Plates                        230
300
.
.
.
45 records listed.
```

You can use association names in queries just as you do field names, so that you could rephrase the previous query as follows:

```
>LIST INVENTORY.F DESCRIPTION ORDERS.ASSOC

LIST INVENTORY.F DESCRIPTION ORDERS.ASSOC 10:07:06AM 31 May 1995
PAGE    1
INVENTORY.F    DESCRIPTION.............    VENDOR.CODE
ORDER.QTY

14             Ice Cream, Various                  140
500
                                                    95
500
                                                    85
100
                                                   228
700
                                                   184
100
                                                   227
500
                                                    12
500
                                                    58
900
16             French Fries, Frozen                116
600
17             Nachos                              116
200
                                                    83
900
                                                   105
800
28             Cookies                              38
500
                                                    69
600
29             Paper Plates                        230
300
.
.
.
45 records listed.
```

Making simple queries

You can start using RetrieVe queries once you know the appropriate command and the name of the file you want to query. The simplest form of a query requires only a verb and file name. A verb is any of the RetrieVe commands, such as `COUNT`, `LIST`, `SORT`, `SEARCH`, and `SELECT`. Complete descriptions of RetrieVe verbs are in the *UniVerse User Reference*.

For example, to count the records in the EQUIPMENT.F file, enter `COUNT` and the file name:

```
>COUNT EQUIPMENT.F

62 records counted.
```

`COUNT` differs from `LIST` in that it displays only the total number of records, rather than data from individual fields.

Note that when typing long commands that exceed a single line on your terminal, the command automatically wraps to the next line. Do not press ENTER until you have finished typing the command. If you want to force the start of a new line, type an underscore (_) at the end of the current line and press ENTER. A plus sign (+) appears at the beginning of the next line to indicate continuation.

File dictionaries

Each UniVerse file has an associated file dictionary that describes each field in the file. The kind of queries you can enter against a file and the output returned depend on the file dictionary for the file.

Note: The following discussion assumes that you are familiar with UniVerse and that you understand the concept of a file dictionary, the metadata that defines the structure and content of a UniVerse data file.

Record IDs

The file dictionary generated by a `CREATE.FILE` command defines only the record ID. The D-descriptor that defines the record ID is named @ID. A record ID uniquely identifies each record in a file; it is sometimes referred to as field 0 and is separated from the rest of the record's data. Everything else has to be manually added to the file dictionary later.

Thus, if you query a file having only this rudimentary dictionary, record IDs are all you see displayed. For example, if the EQUIPMENT.F file had such a rudimentary dictionary, a simple `LIST` command produces the following results:

```
 >LISTEQUIPMENT.F

LIST EQUIP1 @ID 03:31:47PM 31 May 1995     PAGE    1
EQUIPMENT.F

44
28
32
16
4
60
.
.
.
61 records listed.
```

Data descriptors

But in reality the EQUIPMENT.F file, like all files in the Circus database, comes with a complete set of data descriptors. These descriptors define all the fields in the file. The dictionary also includes a special @ phrase that tells which fields to display if no fields are named in the query. For example, the @ phrase for the EQUIPMENT.F file specifies that the EQUIP.CODE, DESCRIPTION, and PURCHASE.DATE fields be displayed by default. Asking for a listing that specifies no fields produces the following output:

```
>LIST EQUIPMENT.F
LIST EQUIPMENT.F 10:24:42AM 31 May 1995    PAGE            1
EQUIP.CODE    DESCRIPTION.............          PURCHASE.DATE

14              Coffee/cookies Stand              12/16/91
```

```
16              Wild West Photo Stand            08/22/92
17              Glamor Photo Stand               08/21/93
 2              Hot Dog Stand                    03/29/89
28              Truck 897 M X X                  06/07/90
.
.
.
61 records listed.
```

A file dictionary can include a number of phrases that can be useful in RetrieVe queries. In addition to the @ phrase, these include @LPTR, @REVISE, and user-defined phrases.

Sorting the output

SORT arranges the output in some defined order. The simplest form of the SORT command is SORT followed by the file name, which sorts the output in order by record ID. To sort the output of the EQUIPMENT.F file, enter:

```
>SORT EQUIPMENT.F

SORT EQUIPMENT.F 10:23:29AM 31 May 1995  PAGE    1
EQUIP.CODE    DESCRIPTION.............    PURCHASE.DATE

         1    Souvenir Stand                   06/04/90
        10    Popcorn Cart                     09/03/90
        11    Sausage-on-a-stick Stand         08/29/90
        12    Beer Keg Stand                   07/24/89
        .
        .
        .
61 records listed.
```

You can also sort the output by any other field in the file. For example, to sort the previous output in descending order by purchase date, enter:

```
>SORT EQUIPMENT.F BY.DSND PURCHASE.DATE

SORT EQUIPMENT.F BY.DSND PURCHASE.DATE 02:44:59PM 31 May 1995
PAGE    1
EQUIP.CODE    DESCRIPTION.............    PURCHASE.DATE

        29    Merry-Go-Round                   01/15/95
        61    1992 Mack Truck Model            12/15/94
              4500L
        62    Calliope                         10/28/94
        33    Truck 588 R W J                  12/16/93
        17    Glamor Photo Stand               08/21/93
        46    Computer                         06/28/93
        .
        .
        .
61 records listed.
```

Displaying specific fields

Notice that the fields listed in the output in the previous examples came from a default list stored in the file dictionary (see Using the @ phrase, on page 31). This default listing is what you get if you specify only the file name in your query.

To display specific fields in the file, enter their field names. For example, enter the field name DESCRIPTION to display the description of products in the INVENTORY.F file:

```
 >LIST
INVENTORY.F DESCRIPTION
LIST INVENTORY.F DESCRIPTION 10:48:42AM 31 May 1995    PAGE  1
INVENTORY.F    DESCRIPTION.............

14             Ice Cream, Various
16             French Fries, Frozen
17             Nachos
28             Cookies
29             Paper Plates
3              Imported Ale
30             Balloons
.
.
.
45 records listed.
```

Note that RetrieVe displays the record ID (the column heading of which is assigned the file name by default—in this case, INVENTORY.F) even though you did not name it in the query. Because the record ID uniquely identifies each record in a UniVerse file, RetrieVe assumes you want to see it unless you indicate otherwise. In UniVerse, record IDs are treated differently from other fields and can affect the output of a query in specific ways. Refer to About record IDs, on page 25 for further information.

Most of the time, your query names specific fields rather than use the default. For example, you might need to produce monthly inventory reports listing the record ID, description, and quantity on hand. Because the record ID is included by default, you need to name only the latter two fields in the query as follows:

```
>LIST INVENTORY.F DESCRIPTION QOH

LIST INVENTORY.F DESCRIPTION QOH 10:51:57AM 31 May 1995  PAGE    1

INVENTORY.F    DESCRIPTION.............    QOH..

14             Ice Cream, Various            154
16             French Fries, Frozen           51
17             Nachos                        140
28             Cookies                       174
29             Paper Plates                  158
3              Imported Ale                   83
30             Balloons                       77
.
.
.
45 records listed.
```

If you ask for a multivalued field, each value in the field is output on a separate line. For example, when you ask for a list of personnel and their dependents' names and dates of birth from the PERSONNEL.F file, the information for each dependent is displayed as shown:

```
>LIST PERSONNEL.F NAME DEP.NAME DEP.DOB BY NAME

LIST PERSONNEL.F NAME DEP.NAME DEP.DOB BY NAME 10:58:25AM 31 May 1995  PAGE
1
PERSONNEL.F    NAME.....................    DEP.NAME..    DEP.DOB...

107            Anderson, Suzanne            Guy             01/24/64
                                            Paul            06/09/88
46             Astin, Jocelyn
7              Bacon, Roger
176            Bailey, Cheryl               Barry           02/16/58
                                            Allen           06/17/82
94             Bennett, Nicholas
67             Bowana, Keltu
190            Brooks, Mary                 Charlie         04/21/47
                                            Suzanne         06/10/72
20             Burrows, Alan
60             Carr, Marion
35             Carr, Stephen                Evelyn          12/06/66
33             Carter, Joseph               Michelle        12/23/57
                                            Joe             05/26/77
53             Clark, Kelly
34             Clark, Lisa
84             Cooper, Peter                Rebecca         01/22/68

187            Dickinson, Alan
162            Dickinson, Cecilia           Stephen         01/23/54
.
.
.
132 records listed.
```

Customizing the output

UniVerse provides many ways for you to customize your output, which are discussed in Customizing query output, on page 60. These include EVAL expressions, aggregate functions, breakpoints, field qualifiers, and report qualifiers.

EVAL expressions can be used in a query to perform ad hoc calculations and produce other values not directly obtainable from the database. For example, a record in the inventory file contains price and cost fields, but not a profit field, so you have to calculate the profit yourself, as shown in the following query:

```
>LIST INVENTORY.F PRICE COST EVAL "(PRICE - COST)" BY ITEM.CODE

LIST INVENTORY.F PRICE COST EVAL "(PRICE - COST)" BY ITEM.CODE
11:02:29AM 31 May 1995  PAGE    1
INVENTORY.F    PRICE.....    COST......    (PRICE-COST)

1                $116.92        $76.92          $40.00
2                 $75.83        $65.94           $9.89
3                 $20.13        $13.51           $6.62
4                 $20.25        $14.57           $5.68
.
.
.
45 records listed.
```

You can also use aggregate functions to do such things as totaling a field, finding averages, and obtaining the highest or lowest value in a field. For example, to total the quantity on hand in inventory, enter:

```
>LIST INVENTORY.F DESCRIPTION TOTAL QOH BY DESCRIPTION

LIST INVENTORY.F DESCRIPTION TOTAL QOH BY DESCRIPTION 11:39:32AM
31 May 1995  PAGE    1
INVENTORY.F    DESCRIPTION.............    QOH..

30             Balloons                      77
1              Beer                         127
38             Bird Seed                     94
45             Bunting                      199
9              Candy Selection               94
.
.
.
40             Ticket Stock                  96
                                            =====
                                             5853
45 records listed.
```

You can also organize output by the values in one or more fields and produce subtotals for each control break. In the INVENTORY.F file, items are categorized by type, and you can ask for the total quantity on hand for each type:

```
>LIST INVENTORY.F BREAK.ON TYPE TOTAL QOH BY TYPE DET.SUP

LIST INVENTORY.F BREAK.ON TYPE TOTAL QOH BY TYPE DET.SUP
11:49:29AM  31 May 1995  PAGE    1
TYPE    QOH..

         5450
B         280
C         289
D         299
.
.
.
Z         223
        =====
         5853

45 records listed.
```

Finally, you can use field qualifiers to tailor the way in which RetrieVe displays an output field, and report qualifiers to control the overall format of the output listing. FMT is an example of a field qualifier and is frequently used to increase or decrease the number of characters allowed for display of a field in the output.

For example, if you wanted to reduce the space allowed for displaying the description in INVENTORY.F from 25 screen columns (characters) to 12, enter:

```
>LIST INVENTORY.F DESCRIPTION FMT 12T TOTAL QOH BY DESCRIPTION

LIST INVENTORY.F DESCRIPTION FMT 12T TOTAL QOH BY DESCRIPTION
03:06:49PM  31 May 1995  PAGE    1
INVENTORY.F    DESCRIPTION.    QOH..

30             Balloons          77
1              Beer             127
38             Bird Seed         94
45             Bunting          199
.
.
.
45 records listed.
```

Compare this output to that of two examples back, and notice that the DESCRIPTION column is significantly narrower. CONV is another commonly used field qualifier and is frequently used to mask field output, such as when you want to add commas, monetary symbols, and other characters to a value. To insert a comma every third integer position in QOH, you would use an MD, conversion code:

```
>LIST INVENTORY.F BREAK.ON TYPE TOTAL QOH CONV MD,
   BY TYPE DET.SUP

LIST INVENTORY.F BREAK.ON TYPE TOTAL QOH CONV MD, BY TYPE DET.SUP
03:13:14PM  31 May 1995  PAGE    1
TYPE    QOH..

B         280
C         289
D         299
F          82
.
.
.
X         131
Z         223
        =====
        5,853

45 records listed.
```

The DET.SUP keyword in the previous query is an example of a report qualifier. Other report qualifiers suppress report and column headings, sample records, set margins, and perform other report functions.

Using select lists

Essentially, a select list is a list of record IDs that represent a subset of records on which you want to perform an operation or a sequence of operations. Once you have obtained a select list, you can save it so you do not have to search through the entire file again for the same records. Select lists are generated by RetrieVe's `SELECT`, `SSELECT`, and `SEARCH` commands and can be used by most RetrieVe commands.

Select lists are commonly used to narrow a search by taking a file, selecting a subset of records, and then further selecting or manipulating that subset. In the Circus database, the LIVESTOCK.F file contains a field that categorizes each circus animal by its use: Z = zoo animals, R = ride animals, and

P = petting animals. If you want to work with only those records belonging to zoo animals, you could make a select list of all zoo animals by entering:

```
>SELECT LIVESTOCK.F WITH USE = Z

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

Now you have a select list that contains the record IDs of all the zoo animals, and you can use this select list as you would a file. For example, you might want to list those zoo animals whose country of origin is Canada. To do this, you enter the query as though you were addressing the LIVESTOCK.F file. Because a select list is active (as indicated by the **>>** prompt), the query uses it as a list of pointers to records in the LIVESTOCK.F file.

```
>>LIST LIVESTOCK.F USE ORIGIN NAME DESCRIPTION
   WITH ORIGIN = Canada

LIST LIVESTOCK.F USE ORIGIN NAME DESCRIPTION WITH ORIGIN = Canada
12:33:30PM   31 May 1995  PAGE    1
LIVESTOCK.F    USE    ORIGIN......    NAME......    DESCRIPTION

2              Z      Canada          Birnin        Mink
75             Z      Canada          Koumra        Fox
6              Z      Canada          Isa           Mink
76             Z      Canada          Kyabe         Weasel
57             Z      Canada          Lawra         Fox
68             Z      Canada          Mopti         Wolverine
85             Z      Canada          Makanza       Skunk
55             Z      Canada          Parakom       Wolverine
82             Z      Canada          Gemena        Wolverine

9 records listed.
```

The use of select lists is further covered in Creating and using select lists, on page 94.

Printing reports

So far, the commands you have seen display their output on the screen. Alternatively, you can redirect the output to a printer by using the LPTR keyword in the query. For example, to list the INVENTORY.F file on the printer, enter:

```
>LIST INVENTORY.F LPTR
```

This query prints on logical print channel 0. To use another print channel, enter the print channel number. For example, to print on print channel 10, enter:

```
>LIST INVENTORY.F LPTR 10
```

Just as you use the @ phrase for convenience when displaying data on the screen, it might also be convenient to print field values without having to specifically name the fields in your query. You could use the @ phrase for this, but because printers can fit more fields on a line than a terminal, using the @ phrase for printing limits your flexibility. Just as the @ phrase defines fields to be displayed when you do not name the fields in the query, the @LPTR phrase defines the fields to be printed when the query does not supply them.

Suppose you decide that the @phrase for the INVENTORY.F file contains too many fields to be displayed on a screen. Using the UniVerse ReVise facility or the UniVerse Editor, you can redefine the @ phrase to include only the item code, cost, price, and quantity on hand, as follows:

```
    @
001 PH
002 ITEM.CODE COST PRICE QOH ID.SUP
```

You could then define the @LPTR phrase to include those fields originally in the @ phrase as follows:

```
         @LPTR
001 PH
002 ITEM.CODE ITEM.TYPE DESCRIPTION QOH COST PRICE VENDOR.CODE -
ORDER.QTY ID.SUP
```

If you do not specify an @LPTR phrase, the @ phrase is used to determine what fields should be output to the printer. If neither an @LPTR phrase nor an @ phrase exists for the file, RetrieVe only prints the record IDs.

Output can also be redirected to another file or to tape. These topics are covered in Redirecting output, on page 106.

Making queries readable

You can use throwaway keywords to make queries more readable. Throwaway keywords are keywords you can use in any query to make the sentence more like English. Throwaway keywords do not affect the query, so you can use them anywhere in the sentence.

The following query uses the words THE, FOR, and FILE without affecting the meaning of the command:

```
>LIST THE ITEM.CODE DESCRIPTION PRICE FOR THE INVENTORY.F FILE
```

The throwaway keywords are:

A	ARE	FILE
FOR	INVISIBLE	OF
PRINT	THAN	THE

Do not use AND or OR for readability because—rather than being throwaway keywords as you might assume—they have special meaning in RetrieVe commands. For example, entering the following command would be incorrect:

```
>LIST ITEM.CODE DESCRIPTION AND PRICE FOR THE INVENTORY.F FILE
```

RetrieVe and SQL

UniVerse also offers a second query language, a UniVerse version of SQL, which performs functions similar to RetrieVe. As an industry standard, SQL provides access to a wide range of existing applications and databases on different servers. UniVerse SQL offers the full complement of security, integrity, and transaction processing of standard SQL languages.

Using SQL, application developers can write interfaces to open systems computing. Because SQL is a relational database language, it allows you to work easily with multiple tables at the same time. You can use RetrieVe to access files created as SQL tables, and use SQL statements to access UniVerse files,

with only minor differences between the results. For more information about UniVerse SQL, see the *UniVerse SQL User Guide* and *UniVerse SQL Reference*.

Chapter 2: Constructing queries

This chapter explains how to query a UniVerse database to retrieve information, organize it, and produce a display or report. This chapter shows how to construct more complicated queries. For example, you will learn about how RetrieVe commands handle record IDs, how you can use phrases in the file dictionary as shortcuts, the different ways you can search for records in a file, and how to sort the output.

Query syntax

Before you can use queries effectively, you need to know how they are structured. RetrieVe queries contain the following elements:

```
verb [DICT] filename [records | FROM list#][selection.expression]
[sort.expression][output.expression] [output.limiter]
[report.qualifiers]
```

These elements determine what the query does and the records and fields the query operates on. Remember, only *verb* and *filename* are required. The following list summarizes each element in the syntax:

Element	Description
verb	Specifies the action to be performed (see RetrieVe verbs, on page 24).
DICT	Queries the dictionary of the file you specify in *filename* instead of the data file. If specified, DICT must precede *filename* in the query.
filename	The name of the file. *filename* is required and can be almost anywhere in the query.
records	A list of record IDs that specifies the records on which the query operates. Enclose the record IDs in single quotation marks.
FROM *list#*	A number, from 0 through 10, of an active select list that contains record IDs. The query operates on those records whose record IDs are in the select list.
selection.expression	Specifies the conditions data in a record must meet in order for the record to be selected for the query.
sort.expression	Specifies the order in which records are listed.
output.specification	Specifies either the names of the fields for output, or one or more EVAL expressions. An output expression can also include special keywords that direct the processing of field values for output.
output.limiter	The WHEN clause, used to limit the output of multivalued fields.
report.qualifiers	Special keywords used in formatting reports.

RetrieVe verbs

To give you an overview of the things you can do with RetrieVe, here is a list of RetrieVe verbs:

Verb	Description
CHECK.SUM	Gets statistical information on the internal storage of values in a particular field for one or more records in a file.
COUNT	Counts the records in a file (see Making simple queries, on page 14).
ESEARCH	Same as SEARCH.
LIST	Searches for and displays data from records in a file (see Introduction, on page 8, Constructing queries, on page 24, and Customizing query output, on page 60).
LIST.ITEM	Displays full listings of selected records (see Getting an internal view of your data, on page 57).
LIST.LABEL	Displays records in a format suitable for mailing labels and other block listings (see Creating mailing labels, on page 91).
REFORMAT	Redirects RetrieVe output to a file or to tape (see Redirecting output, on page 106).
SEARCH	Creates a select list of records that contain an occurrence of one or more specified strings (see Using SEARCH to select on character strings, on page 97).
SELECT	Creates a list of records that meet specified selection criteria (see Introduction, on page 8 and Creating and using select lists, on page 94).
SORT	Lists selected records in sorted order (refer to Sorting records by record IDs, on page 53).
SORT.ITEM	Displays full listings of selected records in sorted order (see Getting an internal view of your data, on page 57).
SORT.LABEL	Displays items in a format suitable for mailing labels and other block listings (see Creating mailing labels, on page 91).
SREFORMAT	Redirects RetrieVe output to a file or to tape, with records sorted by record ID (see Redirecting output, on page 106).
SSELECT	Creates a sorted list of records that meet specified selection criteria (see Creating and using select lists, on page 94).
STAT	Displays numeric statistics for fields in a file (see Customizing query output, on page 60).
SUM	Adds numeric values in fields of records that meet specified selection criteria (see Customizing query output, on page 60).
T.DUMP	Copies records from disk to tape (see Redirecting output, on page 106).
T.LOAD	Copies records from tape to disk (see Loading T.DUMP files from tape to disk, on page 111).

Before getting into more detail about constructing RetrieVe queries, you need to understand two important UniVerse concepts: record IDs and phrases.

About record IDs

In UniVerse, the record ID is treated differently from other fields in a file. One way to visualize record IDs and how they are used is to think of a record in a UniVerse file as having two parts, an index tab

(the record ID) and the page (the fields containing the data of the record). Just as a tab uniquely identifies a page and can be used to find that page, so too a record ID uniquely identifies a record and can be used to search for that record. Because the record ID is special, it is sometimes referred to as field 0, and it is separated from the remainder of the record by a special character called an item mark. In hashed files, the record ID is used in the hashing algorithm that distributes records across groups.

The record ID value can be sequentially assigned numbers, used for the sole purpose of satisfying the requirements of UniVerse file conventions, or they can be meaningful data such as employee badge numbers, part numbers, or account numbers. In any event, the record ID for each record must be unique, and it cannot be a null value.

In the file dictionary there are potentially three types of entry associated with the record ID: the @ID entry, one or more synonyms for the @ID entry, and an @KEY phrase.

The @ID entry

The @ID entry is mandatory in all UniVerse file dictionaries and, in fact, is the only entry defined in the file dictionary by the `CREATE.FILE` command (all other field descriptors must be defined later). The name of the record ID descriptor is @ID, its column name is automatically set to the name of the file, and its output format defaults to 10L (10 screen columns, left-justified). A typical @ID definition looks something like this:

```
    @ID
001 D Default record ID for RetrieVe
002 0
003
004 PERSONNEL.F
005 10L
006 S
```

@ID synonyms

@ID synonyms are optional entries you may find in some file dictionaries, particularly in table dictionaries or in UniVerse files converted from tables through the CONVERT.SQL utility. Defining an @ID synonym lets you refer to the record ID by some name other than @ID. An @ID synonym definition, like the @ID entry, has a field number of 0. In the Circus database, all files have @ID synonyms with names like ITEM.CODE, BADGE.NUMBER, LOCATION.CODE, and so forth. Also, defining an @ID synonym lets you specify a customized conversion code, column heading, and formatting for displaying or printing the record ID. In the PERSONNEL.F file, the @ID and @ID synonym definitions (in this example, the synonym is BADGE.NO) appear in the dictionary as:

```
    @ID
001 D Default record   ID for RetrieVe
002 0
003
004 PERSONNEL.F
005 10L
006 S
        BADGE.NO
001 D
002 0
003 MD0
004
005 5R
006 S
```

You can get the same effect by manually editing the @ID definition to look like the @ID synonym definition.

The @KEY phrase

Another optional entry is the @KEY phrase. The SQL statement CREATE TABLE generates the @KEY phrase in the dictionary of every table. @KEY contains a space-separated list of the column names that make up the table's primary key. The names are in the order of occurrence in the multiple-column primary key.

Displaying record IDs

The record ID is always listed in RetrieVe reports unless you explicitly suppress it using the ID.SUP option. If you enter:

```
LIST filename
```

and there is no @ phrase defined in the file dictionary, you get a list of record IDs. If there is no @ID synonym, the values are headed and formatted according to the definition of the @ID field in the dictionary. For example:

```
>LIST PERSONNEL.F

LIST PERSONNEL.F  10:53:54AM  31 May 1995     PAGE  1
PERSONNEL.F
124
140
159
175
.
.
.
132 records listed.
```

However, because dictionary files for the Circus database do contain @ID synonyms, the list is formatted according to the definition of the synonym:

```
>LIST PERSONNEL.F
LIST PERSONNEL.F  10:53:54AM  31 May 1995     PAGE  1
BADGE.NO
     124
     140
     159
     175
       .
       .
       .
 132 records listed.
```

If you actually ask for @ID in a query, you may be confused by the output because you get two columns of data. The first column is the @ID field (or its synonym field, if one is defined) by default, and the second column is the @ID field you asked for:

```
>LIST PERSONNEL.F @ID

LIST PERSONNEL.F @ID 09:52:37AM    31 May 1995     PAGE   1
BADGE.NO     PERSONNEL.F

124          124
```

```
140          140
159          159
175          175
.
.
.
132 records listed.
```

Alternately, if you use the name of the @ID synonym (BADGE.NO in this example) or @KEY, you again get two columns. This time they are formatted according to the definition of the @ID synonym:

```
>LIST PERSONNEL.F BADGE.NO

LIST PERSONNEL.F BADGE.NO 11:10:16AM  31 May 1995 PAGE  1
PERSONNEL.F            BADGE.NO

124                         124
140                         140
159                         159
175                         175
191                         191
.
.
.
132 records listed.
```

Displaying only record IDs

To list only the record IDs, use the ONLY keyword (or its synonym ID.ONLY):

```
>LIST INVENTORY.F ONLY

LIST INVENTORY.F ONLY 10:06:05AM   31 May 1995 PAGE 1
INVENTORY.F

14
16
17
2
28
29
.
.
.
45 records listed.
```

Using ONLY is equivalent to the default list of record IDs displayed when both of the following conditions are true:

- No @ phrase exists in the file dictionary.
- You do not specifically name fields in the query.

Suppressing record IDs in output

Because many record IDs are no more than sequential record numbers that uniquely identify individual records, you do not always want to list them in a report. For example, even though the following query asks only for the description field, you get a record ID column as well:

```
>LIST INVENTORY.F DESCRIPTION

LIST INVENTORY.F DESCRIPTION 09:59:06AM  31 May 1995  PAGE    1
INVENTORY.F    DESCRIPTION.............

14             Ice Cream, Various
16             French Fries, Frozen
17             Nachos
2              Cotton Candy
28             Cookies
29             Paper Plates
.
.
.
45 records listed.
```

Use the ID.SUP keyword to avoid listing the record IDs. For example, enter:

```
>LIST INVENTORY.F DESCRIPTION ID.SUP

LIST INVENTORY.F DESCRIPTION ID.SUP 10:01:43AM 31 May 1995    PAGE 1
DESCRIPTION.............

Ice Cream, Various
French Fries, Frozen
Nachos
Cotton Candy
Cookies
Paper Plates
.
.
.
45 records listed.
```

Multipart record IDs and multiple-column primary keys

You can create multipart record IDs in UniVerse files. The values making up the multipart record IDs are stored in the @ID field (field 0), with some special character (such as an asterisk) separating the values. To list the values of such multipart record IDs as separate columns in RetrieVe reports, create I-descriptors in the dictionary that uses the `FIELD` function to extract the values you want.

One of the files in the Circus database, ENGAGEMENTS.F, has a multipart record ID.

In a table, a primary key (the SQL equivalent of a record ID) can comprise more than one column (field), the only stipulation being that the combined values from all the primary key columns must be unique for each row. As with multipart record IDs, the values from all the primary key columns are packed into field 0 with a special character separating the values. Such tables must always be updated using SQL statements or a program, because you cannot enter the data into such a field using the Editor or ReVise.

If you want to use SQL statements such as INSERT and UPDATE against UniVerse files that are not tables and that contain multipart record IDs, create an @KEY phrase in the file dictionary, using the following syntax:

```
@KEY
0001: PH
0002: key1 key2 [ keyn ] …
```

By default, the column names specified in field 2 of the @KEY phrase are separated in the record ID by text marks, but you can specify another character as separator. To do this, create an X-descriptor called @KEY_SEPARATOR, using the following syntax:

```
@KEY_SEPARATOR

0001: X

0002: char
```

char is any single character that belongs to the 7-bit character set. It cannot be ASCII NUL (CHAR(0)) or the null value.

An example is the ENGAGEMENTS.F file, which has a record ID comprising the LOCATION.CODE and DATE fields. In the dictionary for ENGAGEMENTS.F, you see that the @ID field is defined as usual, but the two fields that constitute the record ID are defined separately as the first and second segments of @ID:

```
    @ID
001 D Default record ID for RetrieVe
002 0
003
004 ENGAGEMENTS.F
005 10L
006 S
LOCATION.CODE
001 I
002 FIELD(@ID,"*",1)
003
004
005 7L
006 S
    DATE
001 I
002 FIELD(@ID,"*",2)
003 D2/
004
005 10R
006 S
```

This approach leads to some interesting displays that might be confusing. If you request the @ID field in ENGAGEMENTS.F, either by default or directly in a query, the LOCATION.CODE and DATE fields are concatenated and displayed in their raw form with the date unconverted:

```
>LIST ENGAGEMENTS.F ADVANCE

LIST ENGAGEMENTS.F ADVANCE 04:44:37PM  31 May 1995  PAGE  1
ENGAGEMENTS.F          ADVANCE.....

CIAH001*10139            $6,975.00
CIAH001*10611            $7,709.00
WPHX001*10084            $6,134.00
CDFW001*9114
CIND001*9547
CMIL001*9195
EHAR001*9213
WSEA001*10280            $9,768.00
EATL001*9575
WSFO001*9364
WVGA001*10636            $9,403.00
CDFW001*10275            $12,068.00
CMIL001*10102            $3,416.00
CSPR001*9048
```

```
ENYC001*10565            $3,636.00
EPHI001*9814
CDFW001*10150            $11,971.00
.
.
.
252 records listed.
```

Using phrases

Typing a long list of field names or expressions in every query can be tedious. To avoid this, you can use phrases as a kind of shorthand. A phrase is stored in the file dictionary and can contain any elements of a RetrieVe command except the verb and file name.

If you want to refer to a group of fields by a single term—for example, use the term MAILING.INFO to refer to the NAME, ADR1, ADR2, and ADR3 fields in the PERSONNEL.F file—add the dictionary entry:

```
    MAILING.INFO
001 PH
002 NAME ADR1 ADR2 ADR3
```

Then, whenever you want to include these fields in your output, enter:

```
>LIST PERSONNEL.F MAILING.INFO
```

A phrase can specify such things as a list of output field names, often-used expressions, or associated fields. Probably the most commonly used phrase is the @ phrase.

Using the @ phrase

The quickest way to list fields in a database is by using an @ phrase. An @ phrase specifies the fields to display by default if you do not specify them in the query. An @ phrase is defined in the file dictionary. Only one @ phrase can exist for a file.

If you do not know whether an @ phrase exists, enter:

```
LIST filename
```

If all you get is a listing of record IDs, then no @ phrase was defined for the file.

Another way to check for the @ phrase is to list the file dictionary. To get a vertical display of the dictionary entries, enter:

LIST.ITEM DICT *filename*

To get a horizontal display of the dictionary entries, enter:

LIST DICT *filename*

For example, to get a vertical display of the file dictionary for the EQUIPMENT.F file, enter:

```
>LIST.ITEM DICT EQUIPMENT.F

DICT EQUIPMENT.F    01:44:48PM  31 May 1995  Page    1

    @ID
001 D Default record ID for RetrieVe
002 0
004 EQUIPMENT.F
005 10L
006 S
    EQUIP.CODE
001 D
002 0
005 5R
006 S

.
.
.
    @
001 PH
002 ID.SUP EQUIP.CODE DESCRIPTION PURCHASE.DATE
    @REVISE
001 PH
002 VENDOR.CODE VENDOR.REF DEPRECIATION DESCRIPTION COST
    USE.LIFE TAX.LIFE VOLTS PURCHASE.DATE
```

In this case, the @ phrase lists the names of only three of the fields in the file:

```
002 ID.SUP EQUIP.CODE DESCRIPTION PURCHASE.DATE
```

These fields are listed by default if your query does not indicate which specific fields you want to see. So if you simply enter:

```
LIST EQUIPMENT.F
```

you get a listing of these three fields even though you did not name them in the query.

Even if a file has an @ phrase defined, you can override it by specifying one or more fields in your query. For example, the following query lists the record ID (ITEM.CODE), quantity on hand, and price, and ignores the @ phrase entirely:

```
>LIST INVENTORY.F QOH PRICE
LIST INVENTORY.F QOH PRICE 12:03:32PM 31 May 1995  PAGE    1
INVENTORY.F    QOH..    PRICE.....

14               154       $99.36
16                51       $45.78
17               140       $42.06
2                102       $75.83
28               174      $143.55
29               158       $76.51
.
.
.
45 records listed.
```

Besides the @ phrase just discussed, there are two other special phrases that you may encounter in a file dictionary. If you look back at the file dictionary listing, you see an @REVISE phrase, which is the default whenever you use a ReVise command without specifying fields. There is also an @LPTR phrase, which specifies the default list of fields to be used when output is sent to a printer.

You can define these phrases for a file when you create its file dictionary or you can add it later. For example, the @ phrase for INVENTORY.F contains the item code, type, description, cost, and price, and could have been inserted when the dictionary was being built initially or added later when needed:

```
    @
001 PH
002 ID.SUP ITEM.CODE TYPE DESCRIPTION COST PRICE
```

The ID.SUP at the beginning simply suppresses automatic display of the record ID, since you have included ITEM.CODE as one of the fields.

Using other phrases

Besides the special phrases (@ phrase, @LPTR, and @REVISE), there are also user-defined phrases that can provide handy shortcuts when querying.

Suppose that when you ask to see cost you most likely also want to see quantity on hand and price. To avoid having to enter COST, QOH, and PRICE in your queries, you could define a phrase in the file dictionary that encompasses all three and assign it a label, say CQP:

```
    CQP
001 PH
002 ID.SUP COST QOH PRICE
```

You could then use CQP as shorthand in a query to display the three fields, as follows:

```
>LIST INVENTORY.F CQP

LIST INVENTORY.F CQP 09:43:15AM 31 May 1995  PAGE    1
ITEM.CODE    COST......    QOH..    PRICE.....

       14        $80.78      154        $99.36
       16        $34.95       51        $45.78
       17        $28.61      140        $42.06
       28        $98.32      174       $143.55
       29        $48.73      158        $76.51
        3        $13.51       83        $20.13
       30        $43.81       77        $64.40
        .
        .
        .
45 records listed.
```

Note: Naming a phrase in a query overrides the @ phrase, just as naming a field does.

You can combine phrases and field names in the same query. For example, in addition to CQP you could also ask for the item type:

```
>LIST INVENTORY.F CQP TYPE

LIST INVENTORY.F CQP TYPE 09:52:53AM 31 May 1995  PAGE    1
ITEM.CODE    COST........    QOH..    PRICE.......    TYPE

       14          $80.78      154          $99.36    V
       16          $34.95       51          $45.78    R
       17          $28.61      140          $42.06    U
        2          $65.94      102          $75.83    P
       28          $98.32      174         $143.55    R
         .
         .
         .
45 records listed.
```

Besides grouping fields, user-defined phrases in the file dictionary can also define associations (two or more related multivalued fields).

Selecting records

The simplest way to view records is to ask for all the records in the file. Usually, when you look up information in the database, you do not want to see all the records in a file. Instead, you would rather see a subset of the file—records that have certain characteristics. For example, you might want to see a list of inventory items with particular item codes, engagements scheduled for the fourth quarter, acts that last no longer than 20 minutes, or every tenth vendor record.

Using RetrieVe, you can select records in any of the following ways:

- By record ID
- By sampling records
- By finding records that meet certain criteria
- By using select lists (refer to Introduction, on page 8 and Creating and using select lists, on page 94)

Selecting records by record ID

An easy way to select a record is by using the record ID to indicate the record you want to see. Be sure to enclose the record ID value in single quotation marks. In UniVerse, single quotation marks are conventionally used to enclose record ID values, and double quotation marks are used in all other cases where quotation marks are required.

For example, to call up the record for item code 24 in INVENTORY.F, enter:

```
>LIST INVENTORY.F '24'

LIST INVENTORY.F "24" 10:44:12AM  31 May 1995  PAGE    1
ITEM.CODE    TYPE    DESCRIPTION.............    COST......
PRICE.....

      24    P       Jerky                          $48.90
$64.55

1 records listed.
```

Because in some ways record IDs are like other fields, you can use any of the relational operators (EQ, NE, GE, LT, LE, or GT) to select on record IDs.

If you want to retrieve several records, you can list several record IDs in the query, as shown:

```
>LIST INVENTORY.F '23' '11' '14'

LIST INVENTORY.F "23" "11" "14" 09:02:41AM 31 May 1995  PAGE    1
ITEM.CODE    TYPE    DESCRIPTION.............    COST......    PRICE.....

       25    M       Pretzels                       $87.22       $126.47
       11    G       Hot Dog Buns                   $35.33        $41.34
       14    V       Ice Cream, Various             $80.78        $99.36

3 records listed.
```

If you want RetrieVe to prompt you for record IDs, use the INQUIRING keyword. The following query asks you for the record IDs of the records you want to see, one at a time:

```
>LIST INVENTORY.F INQUIRING

LIST INVENTORY.F INQUIRING 09:04:49AM 31 May 1995  PAGE    1
ITEM.CODE    TYPE    DESCRIPTION.............    COST......    PRICE.....

Record =  21

LIST INVENTORY.F INQUIRING 09:05:15AM 31 May 1995  PAGE    1
ITEM.CODE    TYPE    DESCRIPTION.............    COST......    PRICE.....

       21    O       Sea Snails                     $91.17       $100.29

Record =  <Return>

1 records listed.
>
```

To view another record, enter its record ID at the Record = prompt. Pressing ENTER alone, as shown, ends the prompting and returns you to the UniVerse prompt (>).

Selecting records by sampling

Another useful way to select a number of records is by sampling them. For instance, you may want to print several records to test the formatting of a report without processing the entire file. You can sample:

- The first *n* records of a file by using the SAMPLE keyword
- Every *n*th record of a file by using the SAMPLED keyword

To sample the animal ID, name, and description of the first five records in the LIVESTOCK.F file, use SAMPLE:

```
>LIST LIVESTOCK.F NAME DESCRIPTION SAMPLE 5

LIST LIVESTOCK.F NAME DESCRIPTION SAMPLE 5 09:08:51AM 31 May 1995
PAGE    1
LIVESTOCK.F    NAME......    DESCRIPTION

14             Zungeru       Mongoose
2              Birnin        Mink
28             Auchi         Puma
29             Okene         Lion
3              Argungu       Otter

Sample of 5 records listed.
```

Notice that the previous example listed the first five records in LIVESTOCK.F. By contrast, the following query, which samples every fifth record in the 87-record file, lists 17 records (87 divided by 5):

```
>LIST LIVESTOCK.F NAME DESCRIPTION SAMPLED 5

LIST LIVESTOCK.F NAME DESCRIPTION SAMPLED 5 09:10:43AM 31 May 1995  PAGE
1
LIVESTOCK.F    NAME......    DESCRIPTION

3              Argungu       Otter
60             Dabola        Dhole
22             Koko          Stoat
67             Sokolo        Jaguar
18             Jebba         Shetland
34             Baro          Elephant
5              Sokoto        Shetland
64             Morie         Kinkajou
79             Zongo         Cacomistle
15             Kontagora     Shetland
42             Gashua        Lion
68             Mopti         Wolverine
84             Lisala        Sable
32             Ekiti         Shetland
63             Foula         Shetland
39             Shendam       Ferret
71             Bousso        Shetland

Sample of 17 records listed.
```

Selecting records by string comparisons

Using the powerful SEARCH (or its synonym ESEARCH) command, you can find all the records in a file that contain a particular string, regardless of what field the string is in. SEARCH produces a select list which you can then use as input to another command to display those records found to contain the string.

Select lists, and the commands associated with them, are discussed in Creating and using select lists, on page 94.

Selecting records by field values

Most often, you do not know the record IDs of the records you want included in the query, because record IDs may be no more than numbers used for the internal structuring (bookkeeping) of the file. Instead, you want to select records based on whether they contain particular values.

SEARCH (or ESEARCH) selects records containing a specified string value and it is useful when you do not know (or do not care) in what field the value is found. But if you want to focus on a specific field, you need to select records by field values using a WITH clause to define the selection criteria. For example, you might want to modify the previous example and list only those inventory items with the text Frank in the CONTACT field:

```
>LIST VENDORS.F COMPANY CONTACT WITH CONTACT LIKE ...Frank...

LIST VENDORS.F COMPANY CONTACT WITH CONTACT LIKE ...Frank...
09:12:58AM 31 May 1995  PAGE    1
VENDORS.F.     COMPANY.................     CONTACT.............
218            Global Traders              Kroll, Frank
150            Kozy Enterprises            Bennett, Frank
184            Urban Marketing             Ellsworth, Frank
3 records listed.
```

Note: The three periods, or ellipsis, (…) before and after Frank are wildcards, indicating that you want to select all records in which the CONTACT field contains the character string “Frank” regardless of whether other characters precede or follow the string.

A selection expression retrieves only those records in which one or more specific fields contain specific values. Once that has been done, the remainder of the query can process the selected records. For example, you might want to find those records with a contact name containing Frank, sort them alphabetically by company, and print them—all in the same query:

```
>LIST VENDORS.F COMPANY CONTACT WITH CONTACT LIKE ...Frank...
BY COMPANY LPTR
```

A selection expression specifies the criteria that a field must meet in order for the record to be processed. In this case WITH CONTACT LIKE ...Frank... is the selection expression. A selection expression compares values from one or more fields in the file with a value that you specify. The keyword WITH introduces a selection expression, and relational operators compare the values. A selection expression lets you work on a portion of the file without processing the file in its entirety.

Here is another example of a selection expression, one that selects records of employees from PERSONNEL.F who were born before 1950 and lists their badge number, name, and date of birth:

```
>LIST PERSONNEL.F NAME DOB WITH DOB LT 01/01/50

LIST PERSONNEL.F NAME DOB WITH DOB LT 01/01/50 09:18:28AM 31 May 1995  PAGE
1
PERSONNEL.F     NAME.....................     DOB.......

53              Clark, Kelly                    01/28/47
184             Hill, Sandra                    03/16/49
33              Carter, Joseph
7               Bacon, Roger                    03/12/39
.
.
.
43              Wood, Donna                     01/24/45

19 records listed.
```

The relational operator you use in a selection expression depends on the type of data you are comparing: numeric or string.

Data type	Description
Numeric	A field containing strictly numeric data. Comparing their value takes into account the concept of signed values.
String	A field containing a series of characters. The characters are compared according to their code values, depending on the collating sequence being used.

Also, as seen in Selecting records where values are unknown, on page 43, you can compare either type of data to nulls, that is, whether or not a field contains a null value.

The following table shows which relational operators you can use for each type of data.

Data type	Relational operator	Synonym	Description
Numeric fields	EQ	=	Equal to
	NE	#, <>, ><	Not equal to
	GE	>=, =>	Greater than or equal to
	GT	>, AFTER	Greater than
	LT	<, BEFORE	Less than
	LE	=<, >=	Less than or equal to
String fields	LIKE	MATCHES MATCHING	Matches a pattern or text
	UNLIKE	NOT.MATCHING	Does not match a pattern or text
	SAID	SPOKEN, ~ (tilde)	Sounds like
	EQ	=	Equal to
	NE	#, <>, ><	Not equal to
	GE	>=, =>	Greater than or equal to
	GT	>, AFTER	Greater than
	LT	<, BEFORE	Less than
	LE	=<, >=	Less than or equal to
Null values	IS.NULL		Tests whether the field contains a null value
	IS.NOT. NULL		Tests whether the field does not contain a null value

The previous table shows that there are many ways to search for values in a file. Selecting records by field values can be done by using:

- Values in numeric fields
- Values in string (nonnumeric) fields
- Null values in fields
- Comparison of one field to another
- Negation
- Combination of two or more selection expressions
- Secondary indexes

Selecting records based on values in numeric fields

You can select records based on the contents of a numeric field. For example, to list those inventory items that are priced above $100, enter the following:

```
>LIST INVENTORY.F DESCRIPTION PRICE WITH PRICE > $100

LIST INVENTORY.F DESCRIPTION PRICE WITH PRICE > $100 11:10:40AM 31 May
1995  PAGE    1
INVENTORY.F    DESCRIPTION.............    PRICE.....

28                    Cookies                                   $143.55
1                     Beer                                          $116.92
10                    Franks                                        $110.91
12                    Mustard                                   $135.45
22                    Egg Rolls                                 $159.85
25                    Pretzels                                  $126.47
31                    Programs                                  $126.85
38                    Bird Seed                                 $101.03
43                    Sawdust                                   $130.29
21                    Sea Snails                                $100.29
5                     Cola                                          $149.10
11 records listed.
```

Selecting records based on values in string (nonnumeric) fields

Comparing strings offers more options than comparing numeric fields. When comparing strings, you can:

- Compare a string to literal text
- Find text within a string
- Compare strings to a pattern of characters
- Find empty strings
- Find values that sound alike

Comparing a string to literal text

To compare a string field to a literal value, use the relational operators (EQ, NE, GE, LT, LE, or GT). Enter the value to be compared exactly (paying particular attention to upper- and lowercase letters) or you will not get the expected results. We recommend that you enclose the literal text in double quotation marks; this is mandatory if the string includes spaces, punctuation, or special characters.

For example, to find an inventory item with a description of Bird Seed, enclose Bird Seed within quotation marks because it contains a space:

```
>LIST INVENTORY.F ITEM.CODE COST WITH DESCRIPTION
   EQ "Bird Seed"

LIST INVENTORY.F ITEM.CODE COST WITH DESCRIPTION EQ "Bird Seed" 09:59:55AM
31 May 1995  PAGE    1
INVENTORY.F    ITEM.CODE    COST......

38                  38        $84.19

1 records listed.
```

Use GT (or AFTER) and GE to find strings that come after a specified value alphabetically and use LT (or BEFORE) or LE to find strings that come before a specified value. For example, to get an alphabetical listing of all personnel with last names that come after Miller, enter:

```
>LIST PERSONNEL.F NAME WITH NAME AFTER "Miller" BY NAME

LIST PERSONNEL.F NAME WITH NAME AFTER "Miller" BY NAME 10:01:22AM 31 May
1995  PAGE    1
PERSONNEL.F    NAME.....................

54             Milosz, Charles
126            Milosz, James
29             Morse, Carol
192            Morse, Leonard
.
.
.
50             Young, Carol
114            Young, Joan
165            Young, Pamela

68 records listed.
```

Finding text within a string

Because last names appear at the beginning of the NAME field, using a simple comparison worked well in the previous example. Sometimes, though, the text you are looking for can appear anywhere in a field. Use the keyword LIKE or its synonym MATCH or MATCHING to match text within a string.

As you saw in the example of searching for text string Frank in the CONTACT field, you can use three periods as wildcard characters in the search value. As another example of using wildcards, the two sets of three periods in the value in the following query finds the text <space>CA<space> (California) anywhere in the ADR3 field (including at the beginning or end):

```
>LIST LOCATIONS.F ADR3 WITH ADR3 LIKE "... CA ..."

LIST LOCATIONS.F ADR3 WITH ADR3 LIKE "... CA ..." 10:03:25AM 31 May 1995
PAGE    1
LOCATIONS.F    ADR3.....................

WSDO001        SAN DIEGO CA 91914
WSFO001        SAN FRANCISCO CA 94025
WLAX001        LOS ANGELES CA 91025

3 records listed.
```

You can use wildcards in several different combinations to search for a string at the beginning, end, or middle of a field, as shown in the following list:

To match a string...	Specify...	Example
At the beginning of the field	*string*…	WITH NAME LIKE W...
At the end of the field	…*string*	WITH NAME LIKE ...Mary
In any position	…*string*…	WITH NAME LIKE ...Mary...
At the beginning and end of the field	*string1*…*string2*	WITH NAME LIKE W...Mary

Here are two more examples:

```
>LIST PERSONNEL.F NAME WITH NAME LIKE ...Ma...

LIST PERSONNEL.F NAME WITH NAME LIKE ...Ma... 10:04:38AM 31 May 1995  PAGE
1
PERSONNEL.F     NAME.....................

124             Schultz, Mary Lou
191             Jones, Mark
45              Wagner, Mary Beth
.
.
.
60              Carr, Marion
12              Martinez, Suzanne
32              Mahoney, Elizabeth

15 records listed.

>LIST PERSONNEL.F NAME WITH NAME MATCHING M...Elizabeth

LIST PERSONNEL.F NAME WITH NAME MATCHING M...Elizabeth 10:05:50AM 31 May
1995  PAGE    1
PERSONNEL.F     NAME.....................

10              Martinez, Elizabeth
32              Mahoney, Elizabeth

2 records listed.
```

Use the UNLIKE (or NOT.MATCHING) keyword to determine if a string does not contain a text value. For example, to select locations outside of California, enter:

```
>LIST LOCATIONS.F ADR3 WITH ADR3 NOT.MATCHING "... CA ..."

LIST LOCATIONS.F ADR3 WITH ADR3 NOT.MATCHING "... CA ..." 10:08:27AM 31 May
1995  PAGE    1
LOCATIONS.F     ADR3.....................

WREN001         RENO NV 89401
CCLE001         CLEVELAND OH 44110
CDET001         DETROIT  MI 48110
.
.
.
29 records listed.
```

Comparing strings to a character pattern

Another way to select records is to examine a string field for a particular pattern of characters. You can use the MATCH keyword along with a pattern code to examine a field value for such things as a certain number of alphabetic characters, any number of numeric characters, any number of any characters, or some combination thereof. For example, if you wanted to select records containing non-U.S. zip codes, which include alphabetic characters, you could test the zip code field for the presence of at least one alphabetic character.

The following list shows the character codes you can enter to test for different character patterns:

To test for a pattern of...	Enter this character code...
Any number of any characters (including none)	… or 0X
n number of any characters	*n*X
Any number of alphabetic characters (including none)	0A
n number of alphabetic characters	*n*A
Any number of numeric characters (including none)	0N
n number of numeric characters	*n*N
Any literal string	"exact text"

Alphabetic characters are the letters A through Z; characters are any codable character including spaces. When you are looking for a pattern of a specific number of characters, precede the character code with the number of characters you are searching for. For example, to find all personnel with names of exactly 10 characters of any type, enter:

```
>LIST PERSONNEL.F NAME WITH NAME MATCHING '10X'

LIST PERSONNEL.F NAME WITH NAME MATCHING "10X" 10:10:09AM 31 May 1995  PAGE
1
PERSONNEL.F    NAME.....................

70             Ford, Hope
25             Ford, Paul
71             Hill, Kate
40             Tuo, Chang

4 records listed.
```

Just as in searching for text within a field, you use wildcards (**...**) to specify that the pattern can be found anywhere in the field. In the earlier example of selecting records containing non-U.S. zip codes, you would enter:

```
...WITH ZIP MATCHING ...1A...
```

Use the tilde (~) to indicate that you want the negative match of a pattern. For example, to find all street addresses with *no* numbers, enter:

```
>LIST PERSONNEL.F BADGE.NO ADR1 WITH ADR1 MATCHING "~0N"

LIST PERSONNEL.F BADGE.NO ADR1 WITH ADR1 MATCHING "~0N" 10:18:17AM 31 May
1995  PAGE    1
PERSONNEL.F    BADGE.NO    ADR1.....................

163                163    Bradford Arms - Main
                          Street
182                182    Caroline Street

2 records listed.
```

Comparing field values to empty strings

You can also select records on the basis of a field containing an empty string. An empty string is a string of 0 length that is known to have no value. An empty string is represented as " ". Do not confuse an empty string with a null value (which stands for an unknown value) or with a field filled with all blanks (which has a length).

Testing for empty strings can be useful to find data that is known to have no value. For example, you can use an empty string to find products with no price, as follows:

```
>LIST INVENTORY.F DESCRIPTION PRICE WITH PRICE = ""

LIST INVENTORY.F DESCRIPTION PRICE WITH PRICE = "" 10:45:19AM 31 May 1995
PAGE    1
INVENTORY.F    DESCRIPTION.............    PRICE.....

17             Nachos

1 records listed.
```

Using an empty string can help you distinguish among products whose price is unknown or null, products whose price is filled with zeros or blanks, and products that have no price (empty string).

Unlike a null value or a string of blanks, an empty string has no internal representation.

Selecting records where values are unknown

You can also select records by checking a field for the null value. A null value represents data whose value is unknown and is distinct from an empty string or blanks. For example, the PRICE field of a record in the INVENTORY.F file might contain a null value because the price is unknown at the moment.

If you sort a field without first extracting the null values, the null values are included in the output. A better strategy is to sort only those values that are not null. Use the IS.NOT.NULL keyword to select records with field values that are other than null. For example, to find all inventory items with a known price, enter:

```
>LIST INVENTORY.F DESCRIPTION PRICE WITH PRICE IS.NOT.NULL

LIST INVENTORY.F DESCRIPTION PRICE WITH PRICE IS.NOT.NULL 10:50:33AM 31 May
1995  PAGE    1
INVENTORY.F    DESCRIPTION.............    PRICE.....

14             Ice Cream, Various              $99.36
16             French Fries, Frozen            $45.78
17             Nachos
2              Cotton Candy                    $75.83
28             Cookies                        $143.55
29             Paper Plates                    $76.51
3              Imported Ale                    $20.13
30             Balloons                        $64.40
32             Handbills                       $57.33
33             Elephant Chow                   $16.61
.
.
.
43 records listed.
```

Note that the value in the PRICE field for Nachos is an empty string, not null.

To find inventory items with unknown prices (that is, records with a null value in PRICE), use the IS.NULL keyword as follows:

```
>LIST INVENTORY.F DESCRIPTION PRICE WITH PRICE IS.NULL

LIST INVENTORY.F DESCRIPTION PRICE WITH PRICE IS.NULL 10:48:39AM 31 May
1995  PAGE    1
INVENTORY.F    DESCRIPTION.............    PRICE.....

37             Dog Chow
42             Cheese Slices

2 records listed.
```

Note that, even though here the null values in PRICE appear as blanks, nulls may be represented in other ways on other terminals and printers.

Comparing field values to homonyms

If you are unsure of the spelling of the value you are searching for, you can compare the field to a homonym (a “sound alike” value). For example, you think that an employee’s name is something like Gustino, but you are not sure, so instead of entering `LIST PERSONNEL.F WITH NAME EQ 'Gustino'`, you enter:

```
>LIST PERSONNEL.F NAME WITH NAME SAID GUSTINO

LIST PERSONNEL.F NAME WITH NAME SAID GUSTINO 10:57:58AM 31 May 1995  PAGE
1
PERSONNEL.F    NAME.....................

101              Giustino, Susan
4                    Giustino, Carol

2 records listed.
```

In order to be considered a match, the homonym and the value in the field must begin with the same letter.

Selecting records by comparing one field to another

Up to now, the examples have compared a field's contents to a literal or homonym, or looked for null values. You can also select records based on comparing one field in a record to another. For example, to verify that the selling price of each inventory item is always greater than the cost, you could ask for a list of inventory items that have a price less than or equal to their cost:

```
>LIST INVENTORY.F DESCRIPTION WITH PRICE LE COST

LIST INVENTORY.F DESCRIPTION WITH PRICE LE COST 10:59:23AM 31 May 1995
PAGE     1
INVENTORY.F        DESCRIPTION..............
17         Nachos

1 records listed.
```

Nachos is listed because its price contains an empty string (meaning "no price") and an empty string is considered to be equal to 0. As another example, suppose you wanted a listing of all equipment that has an estimated useful life greater than or equal to its depreciation life:

```
>LIST EQUIPMENT.F DESCRIPTION USE.LIFE TAX.LIFE WITH USE.LIFE GE
   TAX.LIFE

LIST EQUIPMENT.F DESCRIPTION USE.LIFE TAX.LIFE WITH USE.LIFE GE TAX.LIFE
11:01:10AM  31 May 1995  PAGE    1
EQUIPMENT.F    DESCRIPTION..............    USE.LIFE    TAX.LIFE

14             Coffee/cookies Stand                7           5
16             Wild West Photo Stand               5           3
17             Glamor Photo Stand                  6           6
.
.
.
33 records listed.
```

Selecting records by using negation

Another way to express selection criteria is by using negation, or asking for the opposite of what the selection expression describes. To negate a selection expression, just precede it with the keyword NOT.

Negation does not do anything you could not express in some other way, although it does allow you to phrase a query in a more natural manner. For example, earlier when you wanted to find any inventory items that had a price less than or equal to their cost, you entered:

```
>LIST INVENTORY.F DESCRIPTION WITH PRICE LE COST
```

If you think of searching for items where price is not greater than cost, then it might be more natural for you to phrase your query as follows:

```
>LIST INVENTORY.F DESCRIPTION WITH NOT PRICE GT COST
```

Combining selection expressions (compound expressions)

You can combine selection expressions by using the logical operators AND or OR. Taking just two selection expressions, *expressionA* and *expressionB*, the effect of combining them with AND and OR is as follows:

If you enter this expression...	The record is selected only if....
expressionA AND *expressionB*	Both *expressionA* and *expressionB* are true.
expressionA OR *expressionB*	Either *expressionA* or *expressionB* is true (or both are true).

If you repeat the WITH keyword at the beginning of each selection expression, you need not use the AND operator; multiple WITH clauses are connected with AND by default.

To select inventory items that have both a quantity on hand of over 150 and a price of less than $100, enter:

```
>LIST INVENTORY.F DESCRIPTION QOH PRICE WITH QOH > 150 AND
   PRICE < $100

LIST INVENTORY.F DESCRIPTION QOH PRICE WITH QOH > 150 AND PRICE < $100
11:04:33AM  31 May 1995  PAGE    1
INVENTORY.F    DESCRIPTION.............    QOH..    PRICE.....

14             Ice Cream, Various            154       $99.36
29             Paper Plates                  158       $76.51
32             Handbills                     154       $57.33
.
.
.
6              Ice Bags                      193       $92.08

13 records listed.
```

But if instead you want to see a list of inventory items that have a quantity on hand of over 150 or a price of under $100, enter:

```
>LIST INVENTORY.F DESCRIPTION QOH PRICE WITH QOH > 150 OR
   PRICE < $100

LIST INVENTORY.F DESCRIPTION QOH PRICE WITH QOH > 150 OR PRICE < $100
11:05:50AM 31 May 1995  PAGE    1
INVENTORY.F    DESCRIPTION.............    QOH..    PRICE.....

14             Ice Cream, Various            154       $99.36
16             French Fries, Frozen           51       $45.78
17             Nachos                        140
28             Cookies                       174      $143.55
29             Paper Plates                  158       $76.51
3              Imported Ale                   83       $20.13
30             Balloons                       77       $64.40
.
.
.
```

You can use shortcuts in the following examples of compound selection expressions:

- If the field name in the second selection expression is the same as the field name in the first one, you can omit the second instance of the name. For example, to find items with a quantity on hand greater than 50 but less than 75, enter:

  ```
  >LIST INVENTORY.F WITH QOH GT 25 AND LT 50
  ```

  instead of entering:

  ```
  >LIST INVENTORY.F WITH QOH GT 25 AND QOH LT 50
  ```

- You can omit OR in a selection expression which compares a field with two different values. For example, the query:

  ```
  >LIST INVENTORY.F DESCRIPTION QOH LT 75 OR GT 150
  ```

  could be shortened to:

  ```
  >LIST INVENTORY.F DESCRIPTION QOH LT 75 GT 150
  ```

Selecting records with secondary indexes

A field in a selection expression might have a secondary index associated with it. A secondary index, also known as an alternate key, is an index structure based on a field that is commonly used as a key for accessing records in a file.

Typically, you would not build a secondary index on the record ID, because a record ID is, in itself, a type of index. But if you frequently selected inventory records on the basis of item type, then the **TYPE** field would be a suitable candidate for the creation of a secondary index.

In general, letting RetrieVe use a secondary index for record selection speeds up the process, because RetrieVe can read a secondary index more quickly than it can scan an entire file. RetrieVe uses a secondary index whenever possible and the underlying mechanism is largely transparent to the user.

It is important to be aware of secondary indexes because:

- Secondary indexes might need to be updated periodically.
- If your query uses a secondary index that has not been updated, the report you produce might not match the data actually in the file.

To avoid this problem, you can require that the query not use the index.

Before you refer to an indexed field in selection criteria, you should first run `LIST.INDEX` to determine whether an index is up-to-date or not yet built. For example, assuming that the **NAME** field had a secondary index, the following display indicates that this index is up-to-date:

```
>LIST.INDEX PERSONNEL.F NAME

Alternate Key Index Summary for file PERSONNEL.F

File.......... PERSONNEL.F
Indices........ 1 (0 A-type, 0 C-type, 1 D-type, 0 I-type, 0 S-type)
Index Updates.. Enabled, No updates pending

Index name      Type  Build    Nulls  In DICT  S/M  Just Unique Field
num/I-type

NAME             D    Not Reqd  Yes    Yes      S    L    N      3
```

If the secondary index is not built, or has not yet been updated, you need to specify that the selection not use the secondary index. To do so, use the keyword `NO.INDEX` to specify that the secondary index not be used.

For example: If you want to select records based on NAME, but you do not want RetrieVe to use the secondary index on NAME, include `NO.INDEX` in your query:

```
>LIST PERSONNEL.F NAME WITH NAME LIKE T... NO.INDEX

LIST PERSONNEL.F NAME WITH NAME LIKE T... NO.INDEX 11:11:06AM  31 May 1995
PAGE    1
PERSONNEL.F    NAME.....................

80             Torres, Stephen
38             Tucker, Alfred
89             Tanaka, Donna
41             Tucker, Joe
40             Tuo, Chang
137            Torres, Ernest
106            Tanaka, Joe

7 records listed.
```

By default (and if available), a secondary index is used in almost all instances. If you want the secondary index on NAME to be used, you do not have to do anything. There are some rare cases where the secondary index is not used by default and you have to include `REQUIRE.INDEX` as follows:

```
>LIST PERSONNEL.F NAME WITH NAME LIKE T... REQUIRE.INDEX
```

Indexes are not used with calculated values

In query, indexes are only used with literal references.

Example 1

The following would not use the index assuming DATE was an indexed field:

```
SELECT file WITH DATE = EVAL "@DATE-2"
```

This is by design, since the value of a calculated file can change during the course of execution.

Example 2

In this example, the value has the potential to change with each record processed.

```
SELECT file WITH DATE = SHIP.DATE
```

Limiting multivalued output with WHEN

Although you can use a selection expression to select records in a query, the output may not always be meaningful when it contains multivalued fields. For example, using WITH to list inventory items that have been ordered in quantities greater than or equal to 900, lists all items whose multivalued ORDER.QTY field contains at least one value greater than or equal to 900:

```
>LIST INVENTORY.F DESCRIPTION VENDOR.CODE ORDER.QTY WITH ORDER.QTY
   GE 900

LIST INVENTORY.F DESCRIPTION VENDOR.CODE ORDER.QTY WITH ORDER.QTY GE 900
11:12:26AM 31 May 1995  PAGE    1
INVENTORY.F    DESCRIPTION.............    VENDOR.CODE    ORDER.QTY

14             Ice Cream, Various                 140          500
                                                   95          500
                                                   85          100
                                                  228          700
                                                  184          100
                                                  227          500
                                                   12          500
                                                   58          900
17             Nachos                             116          200
                                                   83          900
                                                  105          800
32             Handbills                           11          600
                                                   61          200
.
.
.
7              Popcorn                            152          500
                                                  120          500
                                                  155          800
                                                  128          800
                                                  203          900
                                                  218          500

18 records listed.
```

This listing displays all of the multivalues in the selected records. Usually, all you want to see are only those vendors and order quantities that meet the selection criterion of ORDER.QTY GE 900. To limit the display to just those values, use an output limiter. Do not confuse an output limiter with a selection expression: a selection expression selects records, an output limiter both selects records based on a multivalued field and limits display of multivalues within the selected records.

An output limiter begins with the WHEN keyword instead of WITH. For example, to display only those vendors and order quantities where ORDER.QTY is greater than or equal to 900, use WHEN as follows:

```
>LIST INVENTORY.F DESCRIPTION VENDOR.CODE ORDER.QTY WHEN ORDER.QTY
   GE 900

LIST INVENTORY.F DESCRIPTION VENDOR.CODE ORDER.QTY WHEN ORDER.QTY GE 900
11:14:35AM  31 May 1995  PAGE    1
INVENTORY.F    DESCRIPTION.............    VENDOR.CODE    ORDER.QTY

14             Ice Cream, Various                  58          900
17             Nachos                              83          900
32             Handbills                           82          900
33             Elephant Chow                       17          900
1              Beer                                48          900
10             Franks                              81          900
11             Hot Dog Buns                        67          900
22             Egg Rolls                          196          900
24             Jerky                               38          900
25             Pretzels                           113          900
36             Horse Feed                         125          900
37             Dog Chow                           124          900
40             Ticket Stock                       194          900
                                                  139          900
41             T-shirts                            56          900
43             Sawdust                             39          900
20             Crabcakes                           45          900
5              Cola                                46          900
7              Popcorn                            203          900

18 records selected.  19 values listed.
```

The output lists all records with multivalues greater than or equal to 900, and prints a line for each such value, eliminating all order quantities (and their associated vendor codes) that do not meet the criterion. Note that each multivalue listed is counted in the total of 19 values in the summary line at the bottom of the report.

Using WHEN is equivalent to using WHERE and WHEN in a UniVerse SQL statement. For example, the previous query is the same as the following SQL statement:

```
>SELECT ITEM.CODE, DESCRIPTION, VENDOR.CODE, ORDER.QTY
SQL+FROM INVENTORY.F
SQL+WHERE ORDER.QTY GE 900
SQL+WHEN ORDER.QTY GE 900;
```

You can use WHEN with the `LIST`, `SELECT`, `SORT`, `SSELECT`, `STAT`, and `SUM` commands.

Sorting data

When you retrieve records from a file, records are listed in the order in which they are stored in the file. For example, the following query lists records in the order they physically exist in the INVENTORY.F file:

```
>LIST INVENTORY.F DESCRIPTION QOH COST

LIST INVENTORY.F DESCRIPTION QOH COST 11:18:24AM 31 May 1995  PAGE    1
INVENTORY.F    DESCRIPTION.............    QOH..    COST......

14             Ice Cream, Various            154       $80.78
16             French Fries, Frozen           51       $34.95
17             Nachos                        140       $28.61
.
.
.
45 records listed.
```

However, listing records in their physical order is almost never the best approach. A major function of a query language is the ability to sort data before displaying it. You can sort records by record IDs, by fields, or by some combination of both. In the previous example, a likely order would be by description:

```
>LIST INVENTORY.F DESCRIPTION QOH COST BY DESCRIPTION

LIST INVENTORY.F DESCRIPTION QOH COST BY DESCRIPTION 11:20:26AM  31 May
1995  PAGE    1
INVENTORY.F    DESCRIPTION.............    QOH..    COST......

30             Balloons                     77        $43.81
1              Beer                        127        $76.92
38             Bird Seed                    94        $84.19
.
.
.
45 records listed.
```

The UniVerse sort order or collating sequence is based on one of the following:

- The default UniVerse collating sequence for ASCII 7-bit data
- If you use UniVerse in NLS mode, the local convention's collating sequence

If NLS is disabled, UniVerse sorts characters by their byte value. If you enter data with NLS enabled and a map of NONE, this is equivalent to the Unicode sort order. If the data is mapped, for example, Korean, KSC5601 (double-byte), the byte order of the character set determines the sort order.

For ways to customize the collating sequence, see the *UniVerse NLS Guide*.

Sorting and field justification

When you use a sort expression, data is sorted from either left or right, depending on how the field's output format is defined in the file dictionary. If sorting on a field produces unpredictable results, you should check the definition of the field in the file dictionary.

You need to understand how fields are defined to get the sort order you want. The justification of the field as specified in its dictionary entry determines the type of sort performed when you specify a sort expression. For example, dates and most numeric fields should be stored as right-justified to be sorted correctly. Nonnumeric (string) fields generally should be left-justified in order to be sorted properly.

On a left-justified field, sorting is performed left to right according to the collating sequence. On a right-justified field, sorting is right-justified and compares substrings. Thus, if a right-justified field contains a mix of numeric and nonnumeric characters, the numeric portion is sorted numerically, and the nonnumeric characters are sorted left to right. Unexpected results may occur when sorting decimals unless all values have the same number of places to the right of the decimal point.

Much could be said about the sorting algorithm used, but an example is worth a thousand words. So, to better understand sorting, take the following sample data (all examples use the ASCII-7 character set):

```
FIELD.RJ    FIELD.LJ    FIELD.DEC...

     AB1    AB1              -50.25
    AB20    AB20               1.00
     A1A    A1A                0.30
     5AB    5AB                2.25
     125    125               -0.30
     12A    12A               -2.25
    1250    1250            1000.00
    CD20    CD20               0.00
     A1C    A1C              999.00
      AA    AA                -9.99
    AB11    AB11               1.00
     1A1    1A1              -10.00
```

FIELD.RJ is a right-justified string field, FIELD.LJ is a left-justified string field, and FIELD.DEC is a right-justified numeric field.

Compare the differences between sorting on the left-justified string field versus the right-justified string field:

Justification differences	
FIELD.LJ	FIELD.RJ
125	1A1
1250	5AB
12A	12A
1A1	125
5AB	1250
A1A	A1A
A1C	A1C
AA	AA
AB1	AB1
AB11	AB11
AB20	AB20
CD20	CD20

As you can see, sorting on a left-justified field proceeds from left to right (that is, the values are ordered by the leftmost position, and then within that by the next position to the right, and so forth), and numerics come before alphabetics in the ASCII collating sequence. In the example of the right-justified field, due to the mix of numeric and nonnumeric characters, each substring is treated separately, with the numeric characters sorted numerically and the nonnumeric characters sorted left to right. Consequently 1*xx* comes before 5*xx*, which comes before 12*x* which comes before 125, and so on, and 1250 comes before A1A because numbers precede alphabetics.

Now take a look at how a right-justified numeric field is sorted:

```
FIELD.DEC...

     -50.25
     -10.00
      -9.99
      -2.25
```

```
    -0.30
     0.00
     0.30
     1.00
     1.00
     2.25
   999.00
  1000.00
```

As you would expect, right-justified fields are sorted in what might be called arithmetic order, because negative values are taken into consideration. Thus, –50.25 comes before –10.00 in the sort order because –50 is smaller than –10.

Sorting records by record IDs

To sort records by record IDs, use the `SORT` command without including any sort expression (BY @ID is the sort default). For example, to list the records in the INVENTORY.F file sorted by record IDs, enter:

```
>SORT INVENTORY.F DESCRIPTION QOH COST

SORT INVENTORY.F DESCRIPTION QOH COST 11:24:47AM 31 May 1995  PAGE
1
INVENTORY.F     DESCRIPTION.............     QOH..     COST......

1               Beer                         127         $76.92
10              Franks                       151         $99.92
11              Hot Dog Buns                 123         $35.33
12              Mustard                      125         $91.52
.
.
.
19              Fried Clams                  174         $66.31
2               Cotton Candy                 102         $65.94
20              Crabcakes                     87         $28.53
21              Sea Snails                   154         $91.17
.
.
.
45 records listed.
```

Because the record ID is defined by default as being left justified, the order is not quite what you might expect. In this case, it would be better to use the @ID synonym (ITEM.CODE) to get the order you wanted.

You could have used a `LIST` command with a BY @ID phrase instead of `SORT`, or added a superfluous BY @ID phrase to the previous query.

Sorting records by field values

There are times when you want to sort records by a field (or fields) other than record ID (for example, by employee name or inventory item description), and for that you need to use a sort expression. A sort expression starts with the keyword BY or BY.DSND for singlevalued fields, or BY.EXP or BY.EXP.DSND for multivalued fields. When you include a BY phrase in your query, it does not matter whether you use `LIST` or `SORT` as the verb because a sort always results.

BY and BY.EXP both sort in ascending order; BY.DSND and BY.EXP.DSND both sort in descending order. For example, to sort records in the PERSONNEL.F file in ascending order by NAME, enter:

```
>LIST PERSONNEL.F NAME BY NAME ID.SUP
LIST PERSONNEL.F NAME BY NAME ID.SUP 11:30:54AM    31 May 1995     PAGE
1
NAME.....................

Anderson, Suzanne
Astin, Jocelyn
Bacon, Roger
Bailey, Cheryl
Bennett, Nicholas
Bowana, Keltu
Brooks, Mary
Burrows, Alan
.
.
.
132 records listed.
```

Note that to display the sort field, you have to specify its name twice, once in the *sort.expression* and once in the *output.specification*.

To sort the same file in descending order of the NAME field, enter:

```
>LIST PERSONNEL.F NAME BY.DSND NAME ID.SUP

LIST PERSONNEL.F NAME BY.DSND NAME ID.SUP 11:32:07AM   31 May 1995
PAGE            1
NAME.....................

Young, Pamela
Young, Joan
Young, Carol
Yamaguchi, Mary
Wood, Donna
Wood, Debbie
Wilkins, Alan
Whitcomb, Stephanie
Weinstein, Henry
Weinberg, Jeffrey
Wang, Isabel
.
.
.
132 records listed.
```

You can sort on more than one field, and even specify ascending order on some and a descending order on others. For example, to see a list of inventory, ordered by QOH (ascending) and COST (descending), enter:

```
>LIST INVENTORY.F QOH COST DESCRIPTION BY QOH BY.DSND COST

LIST INVENTORY.F QOH COST DESCRIPTION BY QOH BY.DSND COST
01:22:16PM  31 May 1995  PAGE    1
INVENTORY.F    QOH..    COST......    DESCRIPTION..............

16                51        $34.95    French Fries, Frozen
7                 57        $34.15    Popcorn
44                61        $23.60    Onion Rings
13                71        $36.94    Ketchup
8                 71        $11.64    Taffy
30                77        $43.81    Balloons
22                82       $103.80    Egg Rolls
3                 83        $13.51    Imported Ale
20                87        $28.53    Crabcakes
38                94        $84.19    Bird Seed
9                 94        $61.45    Candy Selection
40                96        $57.13    Ticket Stock
.
.
.
45 records listed.
```

As the output shows, the costs for the items having a quantity on hand of 71 are listed in descending order; the same holds true for the costs of the two items with a quantity on hand of 94.

You cannot simply list sort fields one after the other, as you do when specifying fields to be displayed. You must precede each field name with either BY or BY.DSND.

Sorting data with multivalues

When you sort records on a multivalued field, you need to explode the values. Although you could sort a multivalued field without exploding its values, the output is usually not meaningful.

For example, assume that you want to list all scheduled vaccinations for the livestock, in order by date, so that you can mark them on your calendar. Scheduled dates are stored in the multivalued field VAC.NEXT, and the type of vaccination is stored in an associated multivalued field called VAC.TYPE. Sorting on VAC.NEXT without exploding it produces a list of records sorted by the first date found in VAC.NEXT in each record, rather than by all dates, as shown:

```
>LIST LIVESTOCK.F DESCRIPTION VAC.NEXT VAC.TYPE ID.SUP
   BY VAC.NEXT

LIST LIVESTOCK.F DESCRIPTION VAC.NEXT VAC.TYPE ID.SUP BY VAC.NEXT
11:37:05AM  31 May 1995  PAGE    1
DESCRIPTION    VAC.NEXT..    VAC.TYPE

Shetland       06/03/95      R
               09/21/96      P
               04/07/96      L
Horse          06/05/95      R
               08/27/96      P
               08/03/96      L
Horse          06/08/95      R
               01/21/95      P
               02/07/96      L
Linsang        06/12/95      R
               03/27/95      P
               03/30/95      L
Ferret         06/16/95      R
.
.
.
87 records listed.
```

Exploding on a multivalued field effectively creates an individual record for each value found in the field, so that the multivalues can be treated as separate entities. When you explode the multivalued VAC.NEXT field, each date value in the field forms a separate record and those records are then sorted:

```
>LIST LIVESTOCK.F DESCRIPTION VAC.NEXT VAC.TYPE ID.SUP
   BY.EXP VAC.NEXT

LIST LIVESTOCK.F DESCRIPTION VAC.NEXT VAC.TYPE ID.SUP BY.EXP
VAC.NEXT 01:17:48PM  31 May 1995  PAGE    1
DESCRIPTION    VAC.NEXT..    VAC.TYPE

Ferret         05/22/95      L
Otter          05/24/95      L
Hyena          05/24/95      P
Civet          05/26/95      P
Shetland       06/02/95      P
Shetland       06/03/95      L
Shetland       06/03/95      R
.
.
.
87 records listed.
```

Note that each multivalue is displayed on a separate line, and all dates are in ascending order. Any data from the singlevalued fields in the record is repeated in each exploded pseudo-record.

As with singlevalued fields, you can also sort multivalued fields in descending order. To do this, use the keyword BY.EXP.DSND. For example, to list the multivalued field QOH from the INVENTORY.F file in descending order, enter:

```
>LIST INVENTORY.F QOH ITEM.CODE DESCRIPTION ID.SUP
   BY.EXP.DSND QOH

LIST INVENTORY.F QOH ITEM.CODE DESCRIPTION ID.SUP BY.EXP.DSND QOH
01:20:36PM  31 May 1995  PAGE    1
QOH..    ITEM.CODE    DESCRIPTION..............

 5450           50    Soda
  825           46    Corn Dogs
  199           45    Bunting
  197           26    Coffee
  193            6    Ice Bags
  186           39    Film
  185            5    Cola
  181           43    Sawdust
  174           28    Cookies
  174           19    Fried Clams
  171           41    T-shirts
  169           42    Cheese Slices
  165           33    Elephant Chow
  162           23    Sausages
  158           29    Paper Plates
   .
   .
   .
45 records listed.
```

Getting an internal view of your data

Up to now you retrieved data and output it in a conventional report form, with column headings, converted dates, and the like. But there are times when you may need to see something closer to how the data is internally stored. RetrieVe provides two commands for doing this: `LIST.ITEM` and `SORT.ITEM`.

`LIST.ITEM` presents this data unsorted, and `SORT.ITEM` orders the data by record ID. `LIST.ITEM` and `SORT.ITEM` are handy when you want to look at the “raw” data, just as it is internally stored, unconverted, and see the “hidden” character codes that are used to separate multivalues and fields. They are also convenient for getting a listing of a file dictionary in a more easily readable form. Because both commands retrieve the entire record, you cannot specify individual fields. However, you can include selection criteria, sort criteria, and headers and footers in the command. Values in multivalued fields are shown with the value (and subvalue) marks that separate them.

For example, if you used `LIST.ITEM` to look at certain fields in the INVENTORY.F file, you would get the following output:

```
>LIST.ITEM INVENTORY.F BY ITEM.CODE WITH COST < $50

LIST.ITEM INVENTORY.F BY ITEM.CODE WITH COST < $50 04:09:42PM  31
May 1995     PAGE            1

    3
001 U
002 Imported Ale
003 83
004 1351
005 2013
006 152v139v207v14v157v206
007 600v200v500v400v800v400
```

```
    4
001 J
002 Lemonade
003 153
004 1457
005 2025
006 93v199v21v48v161v222v204v72v128
007 600v300v100v500v600v700v200v200v200
.
.
.
```

If you compare this with the layout of the INVENTORY.F file, you see that the record ID is shown as field 0. Following this are ITEM.TYPE (field 1), DESCRIPTION (field 2), QOH (field 3), COST (field 4), PRICE (field 5), VENDOR.CODE (field 6, a multivalued field), and ORDER.QTY (field 7, another multivalued field). Notice that in the documentation a value mark is shown as *v* and a subvalue mark is shown as *s*. How these and other system delimiters appear on a terminal screen or printer depends on the type of terminal or printer and how it is configured.

You can also use either `LIST.ITEM` or `SORT.ITEM` against the dictionary of a file, as shown in the following example:

```
>LIST.ITEM DICT INVENTORY.F
DICT INVENTORY.F  04:13:11PM  31 May 1995     PAGE 1

    @ID
001 D Default record   ID for RetrieVe
002 0
004 INVENTORY.F
005 10L
006 S

    ITEM.CODE
001 D
002 0
005 5R
006 S

    TYPE
001 D
002 1
005 1L
006 S

    DESCRIPTION
001 D
002 2
005 25T
006 S
QOH
001 D
002 3
005 5R
006 S
.
.
.
    @REVISE
001 PH
002 TYPE DESCRIPTION QOH COST PRICE VENDOR.CODE ORDER.QTY
      @
001 PH
```

```
002 ID.SUP ITEM.CODE TYPE DESCRIPTION COST PRICE
```

Chapter 3: Customizing query output

This chapter shows you how to control the way in which the output from a query is displayed or printed.

An output specification identifies the information you want included in the output of a query. The simplest form of output specification is a list of the individual fields to be included in the query output. For example, to list the description, quantity on hand, and cost for records in the INVENTORY.F file, enter the following:

```
>LIST INVENTORY.F DESCRIPTION QOH COST
```

If you do not include an output specification in the query, the fields from the @ phrase (if any) in the file dictionary are displayed. If the file dictionary has no @ phrase defined, only the record IDs are listed. Using an output specification allows you to override the @ phrase (or the default record ID display when no @ phrase is present).

Many of the earlier examples used an output specification to display particular fields, rather than default to the fields named in the @ phrase. But output specifications are designed for much more than this. Use an output specification, to do the following:

- Specify virtual fields for output by creating EVAL expressions. For example, you could create a profit field in INVENTORY.F computed from the price less the cost. Refer to Using virtual fields, on page 60 for a discussion of EVAL expressions.
- Perform calculations, such as average, percent, and total, on fields by using field modifiers. Refer to Performing totals, counts, and averages, on page 65 for a discussion of aggregate computations.
- Create breaks in a report by specifying breakpoints. Refer to Using breakpoints and subtotals, on page 69 for a discussion of using breakpoints and subtotals.
- Format the way data is displayed by using field qualifiers. Refer to Fine-tuning output with field qualifiers, on page 76 for a discussion of controlling output formatting and conversion.

Using virtual fields

You can create a virtual field to display or print the result of some operation (such as a calculation) specified by the current query. Virtual fields are also called EVAL expressions or I-descriptors (I-types). Strictly speaking, the term virtual field refers to the concept, I-descriptor or I-type refers to the definition of a virtual field as stored in a file dictionary, and EVAL expression refers to a dynamically executed I-descriptor (that is, an I-descriptor taking the form of an EVAL expression in an interactive RetrieVe query). In fact, you specify an EVAL expression in a query just as you would define it in an I-descriptor in the file dictionary.

An I-descriptor defines a virtual field whose contents are derived from constants, fields in the associated data file, or fields from other files. I-descriptors or I-type expressions are evaluated every time the RetrieVe sentence referencing or containing them is executed. The I-descriptor can contain field names, operators (arithmetic, logical, or conditional), constants, variables, substring extraction expressions, the `TRANS` function (for file translation), the `TOTAL` function, and UniVerse BASIC functions and subroutines.

Using EVAL expressions for ad hoc calculations

If an operation is repeated frequently, you probably want to store its definition in the file dictionary as an I-descriptor so that it can be accessed when it is required. But sometimes you may be entering

a query and suddenly realize that you need to perform a calculation that has not been defined previously. This is one reason for including an EVAL expression in a query.

For example, you might want to show profit margin as part of your output, and include an EVAL expression as shown:

```
>LIST INVENTORY.F PRICE COST EVAL "(((PRICE - COST)/
   COST) * 100)" CONV MD2 BY ITEM.CODE
( ( ( PRICE - COST ) / COST ) * 100 )
LIST INVENTORY.F PRICE COST EVAL "(((PRICE - COST)/COST) * 100)" CONV MD2
BY ITEM.CODE 09:03:32PM  31 May 1995  PAGE    1
                                        (((PRICE - COST)/COS
INVENTORY.F    PRICE.....    COST......    T) * 100).........

1                $116.92       $76.92                  0.52
2                 $75.83       $65.94                  0.15
3                 $20.13       $13.51                  0.49
4                 $20.25       $14.57                  0.39
5                $149.10      $102.83                  0.45
6                 $92.08       $69.23                  0.33
.
.
.
45 records listed.
```

UniVerse executes the EVAL expression and lists the result as a field in the output. This result is not stored for later use. The CONV MD2 is a field qualifier that formats the profit margin value appropriately; field qualifiers are discussed later in this chapter.

There are three guidelines you must observe when creating EVAL expressions:

- You need write privileges to the file dictionary to create an EVAL expression (because the EVAL expression is temporarily stored in the file dictionary for the duration of the query).
- Enclose the EVAL expression in double quotation marks.
- Use parentheses to indicate the precedence of operations. In the previous example, there are three sets of parentheses in the EVAL expression. The first set indicates that COST is to be subtracted from PRICE first, and then the result of the subtraction is to be divided by COST, and finally the result of the division is to be multiplied by 100.

You can use an EVAL expression in a query anywhere you would use a field name, and, in fact, an EVAL expression may be the output specification. For example, to see what a 15% increase in price would look like, enter the following query:

```
>LIST INVENTORY.F DESCRIPTION ID.SUP EVAL "PRICE * 1.15"
   BY DESCRIPTION
PRICE * 1.15

LIST INVENTORY.F DESCRIPTION ID.SUP EVAL "PRICE * 1.15" BY DESCRIPTION
09:08:40AM  31 May 1995  PAGE    1
DESCRIPTION.............    PRICE * 1.15

Balloons                        $74.06
Beer                           $134.46
Bird Seed                      $116.18
Bunting                         $24.76
.
.
.
45 records listed.
```

The display includes the result of the computation defined in the EVAL expression, that is, the present prices multiplied by 115%. These newly calculated prices exist only for the output of the query and are not stored in the file (or anywhere else). Notice that the column heading for an EVAL expression is the expression itself; if you want to change this heading to something else, you must use a COL.HDG field qualifier. Refer to Customizing column headings, on page 82 for a discussion of COL.HDG.

You can also use EVAL expressions in selection and sort expressions. For example, to list only those items whose newly calculated price is now higher than $150, you could use an EVAL expression as the selection criterion:

```
>LIST INVENTORY.F DESCRIPTION ID.SUP EVAL "PRICE * 1.15"
   BY DESCRIPTION WITH EVAL "PRICE * 1.15" > 150
PRICE * 1.15
PRICE * 1.15

LIST INVENTORY.F DESCRIPTION ID.SUP EVAL "PRICE * 1.15" BY DESCRIPTION WITH
EVAL "PRICE * 1.15" > 150 09:12:01AM  31 May 1995  PAGE    1
DESCRIPTION.............    PRICE * 1.15

Cola                              $171.47
Cookies                           $165.08
Egg Rolls                         $183.83
Mustard                           $155.77

4 records listed.
```

To sort the output in descending order by the newly calculated price, enter the sort expression as an EVAL expression:

```
>LIST INVENTORY.F DESCRIPTION ID.SUP EVAL "PRICE * 1.15"
   BY EVAL "PRICE * 1.15"
PRICE * 1.15
PRICE * 1.15
LIST INVENTORY.F DESCRIPTION ID.SUP EVAL "PRICE * 1.15" BY EVAL "PRICE *
1.15" 09:14:40AM  31 May 1995  PAGE    1
DESCRIPTION.............    PRICE * 1.15

Nachos                             $0.00
Taffy                             $17.14
Film                              $17.81
Elephant Chow                     $19.10
Imported Ale                      $23.15
Lemonade                          $23.29
Large Cat Chow                    $23.67
Bunting                           $24.76
.
.
.
45 records listed.
```

Using EVAL expressions to access other files

Besides using an EVAL expression for an ad hoc calculation, you can also use it to access data from another file. Used with the `TRANS` (translation) function, an EVAL expression can get data from another file.

`TRANS` is a UniVerse BASIC function that returns the contents of a field or record in a UniVerse file. A simplified form of its syntax looks like this:

`TRANS (`*`filename`*`, `*`record.ID`*`, `*`field#`*`, `*`control.code`*`)`

filename is the name of the UniVerse file from which you want to access the data, *record.ID* is an expression that is the source for the record IDs of the records to be accessed, *field#* is the field you want to access, and *control.code* specifies what action to take if the data is not found or is the null value. TRANS is more commonly used in definitions in a file dictionary than in interactive queries, but is shown here in the latter context to illustrate its use.

For example, the VENDORS.F file contains information about the suppliers from whom the circus purchases its inventory. Although the INVENTORY.F file includes the vendor codes of these suppliers, you might want to see the actual company names, information which can be found only in the VENDORS.F file. To list all the items in the INVENTORY.F file and include the company names of their suppliers from the VENDORS.F file, enter:

```
>LIST INVENTORY.F DESCRIPTION ID.SUP EVAL "TRANS(VENDORS.F,
   VENDOR.CODE, COMPANY, 'X')" FMT 25T BY DESCRIPTION
TRANS ( VENDORS.F , VENDOR.CODE , COMPANY , X )
LIST INVENTORY.F DESCRIPTION ID.SUP EVAL "TRANS(VENDORS.F, VENDOR.CODE,
COMPANY, "X")" FMT 25T BY DESCRIPTION 09:39:44AM  31 May 1995  PAGE    1
                           TRANS(VENDORS.F, VEN.....
                           DOR.CODE, COMPANY, '.....
DESCRIPTION.............   X').....................

Balloons                   Community Processors
                           Singapore Logistics
                           River Energy
                           Mountain Providers
                           Continental Mart
                           Pilgrim Processors
Beer                       Emerald Brothers
                           Aspire Innovations
                           Northwestern Academy
                           Urban Marketing
                           Illinois Operations
                           Reliable Merchandise
                           Prime Automation
                           Colonial Equipment
                           Pennsylvania Trading
Bird Seed                  Bayou Manufacturers
.
.
.
Ticket Stock               Sunrise Logistics
                           Platinum Promotions

45 records listed.
```

Essentially this query takes each record in the INVENTORY.F file, uses its VENDOR.CODE field to find corresponding records in the VENDORS.F file (that is, records that have a record ID value corresponding to each VENDOR.CODE value), and extracts the value in the COMPANY field. The X control code specifies that an empty string is to be returned if a corresponding record does not exist or its COMPANY field is null. FMT 25 sets the size of the output field to 25 characters. Note that the quotation marks you use within the TRANS function must differ from the quotation marks you use to specify the EVAL expression (this is a requirement of the UniVerse BASIC language).

You have just seen an example of translating one field from another file. Here's an example of translating multiple fields from a second file, asking for a list of all rides (from the RIDES.F file) together with both the description and vendor code for all the equipment (from the EQUIPMENT.F file) that goes with each ride:

```
>LIST RIDES.F
   EVAL "TRANS(EQUIPMENT.F, EQUIP.CODE, DESCRIPTION, 'X')"
   FMT 25T EVAL "TRANS(EQUIPMENT.F, EQUIP.CODE, VENDOR.CODE,
   'X')"
TRANS ( EQUIPMENT.F , EQUIP.CODE , DESCRIPTION , X )
TRANS ( EQUIPMENT.F , EQUIP.CODE , VENDOR.CODE , X )

LIST RIDES.F EVAL "TRANS(EQUIPMENT.F, EQUIP.CODE, DESCRIPTION, "X")" FMT
25T EVAL "TRANS(EQUIPMENT.F, EQUIP.CODE, VENDOR.CODE, "X")" 09:33:53AM  31
May 1995  PAGE    1
              TRANS(EQUIPMENT.F, E.....    TRANS(EQUIPMENT.F, E
              QUIP.CODE, DESCRIPTI.....    QUIP.CODE, VENDOR.CO
RIDES.F...    ON, 'X').................    DE, 'X').............

1             Sausage-on-a-stick Stand                     207
              Cooling System                               189
              Subsidiary Tent Frame                         19
              Truck 821 N H Y                               22
10            Main Tent                                    188
              Popcorn Cart                                  90
              Desk Credenza Sets                           163
11            Cash Register                                 69
.
.
.
8             V C R                                        201
              Security System                               29
              Wild West Photo Stand                        192
9             Subsidiary Reserve Tent                      118

15 records listed.
```

Rather than translating just a few fields of data, you can translate the data in all the fields (except for the record ID) in the second file by specifying @RECORD (or its equivalent, –1) as the field. For example, if you want to list all the details about the equipment for each ride, rather than just the description and cost, enter:

```
>LIST RIDES.F EVAL "TRANS(EQUIPMENT.F, EQUIP.CODE, @RECORD,
   'X')" FMT 25T
TRANS ( EQUIPMENT.F , EQUIP.CODE , @RECORD , X )

LIST RIDES.F EVAL "TRANS(EQUIPMENT.F, EQUIP.CODE, @RECORD, "X")" FMT 25T
09:47:44AM  31 May 1995  PAGE    1
              TRANS(EQUIPMENT.F, E.....
              QUIP.CODE, @RECORD, .....
RIDES.F...    'X').....................

1             207
              133030182
              G
              Sausage-on-a-stick Stand
              2109673
              4
              4
              110
              8277
              189
              908597064
              E
              Cooling System
              7059165
              4
              7
              110
              7880
              19
              230353709
              M
              Subsidiary Tent Frame
              5758161
              6
              5
              440
              8830
              .
              .
              .
15 records listed.
```

Note that, unlike the standard detail listing, the fields in the second file are not labeled, so you would have to be familiar enough with the file to know that the first line is the vendor code, the second is the vendor reference code, the third is the depreciation code, the fourth is the description, and so forth. Also note that the record ID of the EQUIPMENT.F file is not automatically included as part of the data display, because in UniVerse the record ID is not considered data.

Performing totals, counts, and averages

Rather than just list individual records, you may want to perform certain operations on values in all the records as a whole. For this purpose, RetrieVe provides aggregate computations through the following field modifier keywords:

Keyword	Synonym	Description
AVG	AVERAGE	Computes the average of a field.
BREAK.ON	BREAK-ON	Specifies breakpoints in a report.
BREAK.SUP		Specifies breakpoints in a report, but suppresses the BREAK.ON display.

Keyword	Synonym	Description
CALC	CALCULATE	Computes I-descriptor fields that use the `TOTAL` function.
ENUM	ENUMERATE	Counts and displays the total number of values for a field.
MAX		Lists the highest value found in a field.
MIN		Lists the lowest value found in a field.
PCT	% PERCENT PERCENTAGE	Computes the field as a percentage of the total.
TOTAL		Totals the values in a field.
TRANSPORT		Lists the last value for a field.

As a simple example, if you want to know the average duration of all the acts you currently stage, enter:

```
>LIST ACTS.F DESCRIPTION ID.SUP AVG DURATION BY DESCRIPTION

LIST ACTS.F DESCRIPTION ID.SUP AVG DURATION BY DESCRIPTION 09:53:52AM  31
May 1995  PAGE    1
DESCRIPTION.............    DURATION

Aerial Extravaganza                5
Animals on Parade                  6
Clownarama                        11
Grande Finale                      6
Rock Around the Big Top            5
Salute to the Circus              12
                                ========
                                    7.5

6 records listed.
```

Or perhaps you want to know the highest price charged for an item in inventory:

```
>LIST INVENTORY.F MAX PRICE DET.SUP

LIST INVENTORY.F MAX PRICE DET.SUP 09:56:24AM  31 May 1995  PAGE    1
MAX PRICE.

==========
   $159.85

45 records listed.
```

Note that you must specify `DET.SUP` (Detail Suppress), or you get a listing showing each individual inventory item and its price, followed by the highest value.

If you want to know how many different vendors you purchase each inventory item from (without actually listing the vendor codes), enter:

```
>LIST INVENTORY.F BREAK.ON ITEM.CODE ENUM VENDOR.CODE DET.SUP
   BY ITEM.CODE

LIST INVENTORY.F  BREAK.ON ITEM.CODE ENUM VENDOR.CODE DET.SUP ID.SUP BY
ITEM.CODE 10:02:52AM  31 May 1995  PAGE    1
ITEM.CODE    ENUM VENDOR.CODE

        1                   9
        2                   2
        3                   6
        4                   9
        5                   8
        6                   4
        .
        .
        .
       43                   5
       44                   1
       45                   2
            ================
                         221

45 records listed.
```

Here ENUM VENDOR.CODE performs a count on the values in the VENDOR.CODE field, BREAK.ON ITEM.CODE sorts the output and totals this count by ITEM.CODE, and `DET.SUP` suppresses the listing of the individual records. Refer to Using breakpoints and subtotals, on page 69 for more information on using breakpoints.

As another example, you might want to find what percentage of the whole each inventory item represents in terms of quantity on hand:

```
>LIST INVENTORY.F DESCRIPTION ID.SUP QOH PCT QOH BY DESCRIPTION

LIST INVENTORY.F DESCRIPTION ID.SUP QOH PCT QOH BY DESCRIPTION 10:08:23AM
31 May 1995  PAGE    1
DESCRIPTION.............    QOH..    QOH...

Balloons                     77      1.32
Beer                        127      2.17
Bird Seed                    94      1.61
Bunting                     199      3.40
Candy Selection              94      1.61
Cheese Slices               169      2.89
.
.
.
T-shirts                    171      2.92
Taffy                        71      1.21
Ticket Stock                 96      1.64
                                   ======
                                   100.00

45 records listed.
```

The report lists a percentage value for each record in the file, and therefore the listing shows that the quantity on hand for balloons represents 1.32% of the total inventory, the quantity on hand for taffy represents 1.21% of the inventory, and so forth. Note that if you want to actually show the quantity on hand, you must specify QOH twice in the query: the first mention specifies that the field be listed, the second mention identifies the field on which to calculate the percent. Also note that the total percentage is automatically computed and displayed as part of the `PCT` function, but the report does not list the total quantity on hand. Use the `TOTAL` field modifier to list the total quantity as well. For example:

```
>LIST INVENTORY.F DESCRIPTION ID.SUP TOTAL QOH PCT QOH
   BY DESCRIPTION
LIST INVENTORY.F DESCRIPTION ID.SUP TOTAL QOH PCT QOH BY DESCRIPTION
10:11:39AM  31 May 1995  PAGE    1
DESCRIPTION.............    QOH..    QOH...

Balloons                     77      1.32
Beer                        127      2.17
Bird Seed                    94      1.61
Bunting                     199      3.40
Candy Selection              94      1.61
Cheese Slices               169      2.89
Coffee                      197      3.37
.
.
.
T-shirts                    171      2.92
Taffy                        71      1.21
Ticket Stock                 96      1.64
                           =====    ======
                            5853    100.00

45 records listed.
```

Notice that because TOTAL QOH is an output specification, it automatically lists the field being totaled so you do not need to specify QOH a third time in order to have it appear in the listing.

Field modifiers perform functions similar to those of some RetrieVe commands. Earlier, you asked for the average performance time by using the AVG modifier. You could instead have used the RetrieVe `STAT` command to get the average, but without a detail listing of the individual records. The report also lists the total time of all acts and the record count for the file:

```
>STAT ACTS.F DURATION

STAT ACTS.F DURATION 10:14:08AM 31 May 1995  PAGE    1
ACTS.F....    DURATION

              ========
TOTALS              45

              ========
AVERAGES           7.5

              ========
COUNTS              6

6 records summed.
```

The `SUM` command is analogous to the TOTAL modifier, in that it totals a field. Earlier, you asked for the TOTAL QOH and you got a detailed listing of inventory items with a total QOH at the end. If you want only the total QOH, just enter:

```
>SUM INVENTORY.F QOH COST

SUM INVENTORY.F QOH COST 10:15:43AM  31 May 1995  PAGE    1
INVENTORY.F    QOH..    COST......

                      =====     ==========
TOTALS                5853       $2,394.82
45 records summed.
```

Note: If a multivalued field is named in a SUM phrase, all the values are added together. For example, if you summed the multivalued field ORDER.QTY, you would get a total of all the multivalues in the field:

```
>SUM INVENTORY.F ORDER.QTY

SUM INVENTORY.F ORDER.QTY 09:43:26AM 01 May 1995    PAGE            1
INVENTORY.F            ORDER.QTY

                       =========
TOTALS                    115000

45 records summed.
```

Using breakpoints and subtotals

You can organize your output by the values in one or more fields, and produce subtotals for each different value, by using breakpoints. Breakpoints are merely another kind of field modifier, and should be familiar to anyone who has worked with financial reports. You can create breakpoints to perform calculations, such as average, percent, and total, when values in another field change.

When you want to organize a file by the content of a field and then produce a subtotal each time the value of that field changes, use the BREAK.ON keyword and specify a sort on that field.

As an example of using BREAK.ON, assume that you want to see the detail cost for each inventory item along with the total advance per type. To obtain this, include both a BREAK.ON and a BY (sort) on TYPE and perform a TOTAL calculation on COST:

```
>LIST INVENTORY.F BREAK.ON TYPE DESCRIPTION TOTAL COST
   BY TYPE BY DESCRIPTION ID.SUP

LIST INVENTORY.F BREAK.ON TYPE DESCRIPTION TOTAL COST BY TYPE BY
DESCRIPTION ID.SUP 02:45:17PM  31 May 1995  PAGE    1
TYPE    DESCRIPTION.............    COST......

B       Candy Selection                $61.45
B       Film                           $10.76
**                                  ----------
B                                      $72.21

C       Frozen Yogurt, Various         $23.68
C       Sawdust                        $90.48
**                                  ----------
C                                     $114.16

D       Fried Clams                    $66.31
.
.
.
Z       Large Cat Chow                 $17.59
Z       Ticket Stock                   $57.13
**                                  ----------
Z                                      $74.72

                                    ==========
                                     $2,394.82

45 records listed.
```

Note that a grand total is provided automatically at the end of the listing.

If the break field were multivalued, you would use BY.EXP instead of BY for the sort. For example, to take the INVENTORY.F file and calculate the cost of the quantity on hand by vendor, use a BY.EXP and a BREAK.ON on VENDOR.CODE:

```
>SORT INVENTORY.F BY.EXP VENDOR.CODE BREAK.ON VENDOR.CODE
   DESCRIPTION TOTAL EVAL "COST * QOH" FMT 15R ID.SUP
COST * QOH
SORT INVENTORY.F BY.EXP VENDOR.CODE BREAK.ON VENDOR.CODE DESCRIPTION TOTAL
EVAL "COST * QOH" FMT 15R ID.SUP 03:13:35PM  31 May 1995  PAGE    1
VENDOR.CODE    DESCRIPTION.............    COST * QOH.....

          2    Coffee                            $5,222.47
**                                          ---------------
          2                                      $5,222.47

          6    Dog Chow                         $12,623.16
**                                          ---------------
          6                                     $12,623.16

          9    Sausages                          $6,190.02
**                                          ---------------
          9                                      $6,190.02

         10    Large Cat Chow                    $2,233.93
**                                          ---------------
         10                                      $2,233.93

         11    Handbills                         $6,588.12
**                                          ---------------
         11                                      $6,588.12

         12    Ice Cream, Various               $12,440.12
         12    Pretzels                         $11,774.70
**                                          ---------------
         12                                     $24,214.82
.
.
.
        231    Hot Dog Buns                      $4,345.59
**                                          ---------------
        231                                      $4,345.59

                                            ===============
                                             $1,638,334.64

221 records listed.
```

Labeling grand totals and subtotals

Use the GRAND.TOTAL keyword to label the grand total line of a report that includes breakpoints. For example, adding GRAND.TOTAL to the earlier query where you asked for total advances results in the labeling of the grand total line as shown:

```
>LIST INVENTORY.F BREAK.ON TYPE DESCRIPTION TOTAL COST
   BY TYPE BY DESCRIPTION ID.SUP GRAND.TOTAL "TOT:"

LIST INVENTORY.F BREAK.ON TYPE DESCRIPTION TOTAL COST BY TYPE BY
DESCRIPTION ID.SUP GRAND.TOTAL "TOT:" 02:57:51PM  31 May 1995  PAGE    1
TYPE    DESCRIPTION.............    COST......

B       Candy Selection                $61.45
B       Film                           $10.76
**                                  ----------
B                                      $72.21

C       Frozen Yogurt, Various         $23.68
C       Sawdust                        $90.48
**                                  ----------
C                                     $114.16

D       Fried Clams                    $66.31
.
.
.
Z       Large Cat Chow                 $17.59
Z       Ticket Stock                   $57.13
**                                  ----------
Z                                      $74.72

TOT:                                ==========
                                    $2,394.82

45 records listed.
```

In addition to labeling the grand total, you can also label the subtotal lines, via the text option in BREAK.ON. For example, to add the label "Type" to the previous output, enter:

```
>LIST INVENTORY.F BREAK.ON "Type" TYPE DESCRIPTION
   TOTAL COST BY TYPE BY DESCRIPTION ID.SUP
   GRAND.TOTAL "TOT:"

LIST INVENTORY.F BREAK.ON "Type" TYPE DESCRIPTION TOTAL COST  BY TYPE BY
DESCRIPTION ID.SUP GRAND.TOTAL "TOT:" 03:00:52PM  31 May 1995  PAGE    1
TYPE    DESCRIPTION.............    COST......

B       Candy Selection                $61.45
B       Film                           $10.76
Type                                ----------
B                                      $72.21

C       Frozen Yogurt, Various         $23.68
C       Sawdust                        $90.48
Type                                ----------
C                                     $114.16

D       Fried Clams                    $66.31
.
.
.
Z       Large Cat Chow                 $17.59
Z       Ticket Stock                   $57.13
Type                                ----------
Z                                      $74.72

TOT:                                ==========
                                    $2,394.82

45 records listed.
```

Note that, when labeling grand totals and subtotals, the width of your text is determined by the number of characters allotted to the data in the first column, which in the previous example was the four characters occupied by the column heading TYPE. If your grand total or subtotal label exceeds that, it is truncated, unless you specify a FMT field qualifier to extend the width of the column. Refer to Formatting the output of fields and EVAL expressions, on page 78 for a discussion of formatting output.

For example, if you want to label the subtotals "Total for Type:", you need to add FMT 15L to the BREAK.ON phrase in the query:

```
>LIST INVENTORY.F BREAK.ON "Total for Type:" TYPE FMT 15L
        DESCRIPTION TOTAL COST BY TYPE BY DESCRIPTION ID.SUP
        GRAND.TOTAL "TOT:"
```

Suppressing detail lines

In many earlier examples, the output consisted of a detail line for each record, followed by a subtotal for each break, and a grand total at the end. To see only the subtotals and suppress the individual detail lines, use the DET.SUP keyword, as follows:

```
>LIST INVENTORY.F BREAK.ON "Type" TYPE TOTAL
   COST BY TYPE GRAND.TOTAL "TOT:" DET.SUP

LIST INVENTORY.F BREAK.ON "Type" TYPE TOTAL COST BY TYPE GRAND TOTAL "TOT:"
DET.SUP 10:24:12am 31 May 1995  PAGE    1
TYPE    COST......

B           $72.21
C          $114.16
D          $157.83
F          $103.80
G           $69.48
J           $14.57
K          $171.13
M          $256.37
N          $197.59
O          $151.71
P          $184.56
Q           $49.60
R          $203.59
T           $42.78
U           $42.12
V          $226.30
W          $165.94
X           $96.36
Z           $74.72
         $2,394.82

45 records listed.
```

When you use DET.SUP, any labels you have specified for breakpoints or grand total are ignored, and the default output of record IDs is suppressed as well.

Specifying multiple breakpoints

Breakpoints can be even more powerful if you use several breakpoints in the same query. Using multiple breakpoints organizes the information by several different fields while also performing calculations on those fields. Breakpoints are specified in order from left to right.

For example, if you want to know your total livestock investment, broken down by use and origin, enter:

```
>SORT LIVESTOCK.F BY USE BY ORIGIN BREAK.ON USE BREAK.ON
   ORIGIN TOTAL COST ID.SUP DET.SUP

SORT LIVESTOCK.F BY USE BY ORIGIN BREAK.ON USE BREAK.ON ORIGIN TOTAL COST
ID.SUP DET.SUP 03:29:03PM  31 May 1995  PAGE    1
USE     ORIGIN......     COST........

P       Austria            $24,006.00
P       Brazil             $10,697.00
P       England            $18,427.00
P       India               $6,867.00
P       Kenya              $59,023.00
P       Pound               $6,340.00
P       Tahiti              $8,960.00
P                         $134,320.00
R       Egypt              $29,881.00
R       England            $61,536.00
R       India               $6,529.00
R                          $97,946.00
.
.
.
Z       Peru                $8,016.00
Z       Russia              $6,134.00
Z       Siberia            $10,985.00
Z       Texas              $16,516.00
Z       Ukraine            $13,807.00
Z                          $305,608.00
                           ============
                           $537,874.00

87 records listed.
```

Note that you get an intermediate total for each country of origin, a major total for each use, and a grand total.

Suppressing breakpoint lines, fields, and subtotals

There are several ways to suppress unneeded or unwanted output when using breakpoints.

For instance, in a previous example that listed the items purchased from each vendor, several vendors supplied only one item apiece, and having both a detail line and a breakpoint line is somewhat superfluous. One example of this is vendor 2, who supplies only coffee, and so the detail line and the total line show the same information:

```
        2           Coffee          $5,222.47
**                                ---------------
    2                                $5,222.47
```

If this happens often, you might want to suppress the printing of the detail line and show only the breakpoint line where there is only one detail line for a specific breakpoint value. To do this, use the D option of the BREAK.ON keyword as follows:

```
>SORT INVENTORY.F BY.EXP VENDOR.CODE BREAK.ON "'D'"
   VENDOR.CODE DESCRIPTION TOTAL EVAL "COST * QOH"
   FMT 15R ID.SUP
COST * QOH

SORT INVENTORY.F BY.EXP VENDOR.CODE BREAK.ON ""D""  VENDOR.CODE DESCRIPTION
TOTAL EVAL "COST * QOH" FMT 15R ID.SUP 03:38:56PM  31 May 1995  PAGE    1
VENDOR.CODE    DESCRIPTION.............    COST * QOH.....

          2    Coffee                            $5,222.47

          6    Dog Chow                         $12,623.16

          9    Sausages                          $6,190.02

         10    Large Cat Chow                    $2,233.93

         11    Handbills                         $6,588.12

         12    Ice Cream, Various               $12,440.12
         12    Pretzels                         $11,774.70
                                                ----------
         12                                     $24,214.82
          .
          .
          .
45 records listed.
```

Now there is only one output line for each of these vendors, and that is the breakpoint line.

In rare instances, you may not want to display the field on which you are breaking, perhaps for reasons of confidentiality. For example, if you wanted to produce a salary survey report by department but wanted to hide the identity of the departments, you could break on the department but use BREAK.SUP instead of BREAK.ON to suppress the display of the department designation.

In still other cases, you may want to break on a field, but not produce subtotals. You can do this by using the L option with the BREAK.ON keyword. The output skips a line to indicate the breakpoint, but no text or totals appear on the line. For example, to list the inventory items by vendor, break on a change in vendor, but not do any totaling, enter:

```
>LIST INVENTORY.F BY.EXP VENDOR.CODE BREAK.ON "'L'"
   VENDOR.CODE DESCRIPTION ID.SUP

LIST INVENTORY.F BY.EXP VENDOR.CODE BREAK.ON ""L"" VENDOR.CODE DESCRIPTION
ID.SUP 11:41:39AM 31 May 1995  PAGE    1
VENDOR.CODE    DESCRIPTION..............

          2    Coffee

          6    Dog Chow

          9    Sausages

         10    Large Cat Chow

         11    Handbills

         12    Pretzels
         12    Ice Cream, Various

         13    Hot Dog Buns
         13    Ice Bags
         13    Horse Feed
         .
         .
         .
45 records listed.
```

Paging on breakpoints

Use the P option inside the text of BREAK.ON to force reports to start a new page with each breakpoint. For example, the following query produces a report that starts on a new page with each break on VENDOR.CODE:

```
>LIST INVENTORY.F BREAK.ON "Vend Total 'P'" VENDOR.CODE
   BY.EXP VENDOR.CODE TOTAL EVAL "COST * QOH"
   GRAND.TOTAL "FINAL:" ID.SUP
LIST INVENTORY.F BREAK.ON "Vend Total "P"" VENDOR.CODE BY.EXP VENDOR.CODE
TOTAL EVAL "COST * QOH" GRAND.TOTAL "FINAL:" ID.SUP 11:45:07AM  31 May 1995
PAGE    1
VENDOR.CODE    COST * QOH

          2     $5,222.47
Vend Total      ----------
          2     $5,222.47


Press any key to continue...

LIST INVENTORY.F BREAK.ON "Vend Total "P"" VENDOR.CODE BY.EXP VENDOR.CODE
TOTAL EVAL "COST * QOH" GRAND.TOTAL "FINAL:" ID.SUP 11:46:16AM  31 May 1995
PAGE    2
VENDOR.CODE    COST * QOH

          6    $12,623.16
Vend Total      ----------
          6    $12,623.16




Press any key to continue...
.
.
.
45 records listed.
```

Fine-tuning output with field qualifiers

Many times, these field qualifiers are included in a field's definition in the file dictionary. But sometimes you want to change field characteristics "on the fly," that is, without permanently changing the file dictionary. However, because UniVerse temporarily writes entries to the dictionary when you use these qualifiers, you must have write privileges on the dictionary.

Field qualifiers serve several purposes:

- Control output formatting and converting of data (FMT and CONV)
- Define the column heading for a field (COL.HDG)
- Create aliases for field names and assign names to virtual fields (AS)
- Redefine a field's singlevalued/multivalued status or its association with other fields (ASSOC, ASSOC.WITH, MULTI.VALUE, and SINGLE.VALUE)
- Copy the display characteristics of another field (DISPLAY.LIKE)

The field qualifier keywords are summarized in the following table:

Keyword	Synonym	Description
AS		Specifies a name for an EVAL expression or a synonym for a field name.
ASSOC	ASSOCIATION	Associates a field expression with an association of multivalued fields.

Keyword	Synonym	Description
ASSOC.WITH	ASSOCIATED	Associates a field expression with another multivalued field.
COL.HDG	DISPLAY.NAME DISPLAYNAME	Defines a column heading.
CONV	CONVERSION	Defines a conversion.
DISPLAY.LIKE	DISPLAYLIKE	Sets display characteristics.
FMT	FORMAT	Defines output formatting.
MULTI.VALUE	MULTIVALUED	Specifies a multivalued field expression.
SINGLE.VALUE	SINGLEVALUED	Specified a single-valued field expression.

A field qualifier in a query overrides any similar field qualifiers in the dictionary definition of the field, and is in effect only during execution of the query. In constructing a query, place the field qualifiers immediately after the field name or temporary field definition (EVAL expression) to which they apply.

Controlling output formatting and conversion

By default, the form in which a field is displayed is determined by how the field is defined in the file dictionary. Also by default, the form in which output from an aggregate function or virtual field is displayed is that of the first or only field named in the modifier or expression.

For example, if you use the phrase TOTAL PRICE in a query, the form used to display the total defaults to the form defined for PRICE. Likewise, in the previous example, where an EVAL expression (COST * QOH) was used to calculate the total value of quantity on hand by vendor, the output format of the result defaults to the format of the COST field.

There are three field qualifiers that can be used to temporarily apply formatting and conversion characteristics to a field:

- FMT
- CONV
- DISPLAY.LIKE

Generally, formatting (FMT) is used to specify the width of a display column, the type of justification, the format of numeric data, and masking and padding characters. Conversion (CONV) applies any BASIC conversion code available to the `ICONV` and `OCONV` functions; date and time conversions are typical examples. DISPLAY.LIKE is used to set a field's display characteristics to be the same as those of another field.

Before getting into the details, there are three overall concepts that you should understand about formatting and conversion:

- If a query specifies a FMT or CONV for a field, that specification totally overrides any corresponding FMT or CONV supplied for the field in the file dictionary.
- Conversion (CONV) is performed immediately before the output process, after all selection, sorting, and calculations have all been completed, while formatting (FMT) is part of the output process and is the last operation performed on the data before it is written to the screen, printer, or tape drive.
- If more than one FMT or CONV is supplied for a field in a query, the formatting or conversion processes are performed in sequence, such as the output from the first CONV is input to the second CONV, and so forth.

Formatting the output of fields and EVAL expressions

FMT (or FORMAT) is the field qualifier that you use to change the appearance—length and justification—of a field or EVAL expression in the output. It can also specify masking and padding characters and scaling. In the absence of any FMT specification in either the file dictionary definition or the query, the default is FMT 10L.

Syntax

The syntax for FMT is:

FMT *width* [*background*] *justification* [*edit*] [*mask*]

For example, FMT "10L" positions the data left-justified in a 10-character-wide column. FMT "5R" positions the data right-justified in a five-character-wide column. FMT "15T" positions the data left-justified in a 15-character-wide column and breaks a line at a space rather than in the middle of a word. As a more complex example, FMT "10*R2$" positions the data right-justified in a 10-character-wide column, pads the data with asterisks (*), allows two positions to the right of the decimal point, and places a $ to the immediate left of the leftmost significant digit: ***$102.83.

There are many reasons for using FMT in a query. For one, you might want to increase or decrease the width allowed for displaying a field so that it fits better on a line. For example, if you ask for a listing of names and addresses from the PERSONNEL.F file, you get a vertical listing because both fields are defined with a format of 25T in the file dictionary and they cannot fit across the page:

```
>LIST PERSONNEL.F NAME ADR1 ADR2

LIST PERSONNEL.F NAME ADR1 ADR2 11:47:12AM  31 May 1995  PAGE    1

PERSONNEL.F. 124
NAME........ Schultz, Mary Lou
ADR1........ 6520 Bomar Street
ADR2........ Happy  TX 79042
.
.
.
132 records listed.
```

You can make them all fit horizontally by using FMT to shorten their output length:

```
>LIST PERSONNEL.F NAME FMT 18L ADR1 FMT 25L ADR2 FMT 22L
LIST PERSONNEL.F NAME FMT 18L ADR1 FMT 25L ADR2 FMT 22L ID.SUP 11:50:51AM
31 May 1995  PAGE    1
NAME..............    ADR1.....................    ADR2...................

Schultz, Mary Lou     6520 Bomar Street           Happy  TX 79042
Powers, Jean          9690 Bell Street            Carefree  AZ 85719
Elliott, Warren       2700 Mason Street           Thankful  NC 28683
Mcmanus, Joe          3050 Dowling Street         Ideal  SD 57541
Jones, Mark           3670 Bastrop Street         Access  OH 43901
Morse, Carol          550 Hadley Street           Equity  OH 43749
.
.
.
132 records listed.
```

Because justification affects sorting, you might also use FMT to cause a left-justified field to be sorted as though it were right-justified or vice versa. Thus in a situation where a numeric field is defined as left-justified in the file dictionary, which would cause some unexpected results if you tried to sort it, you would use a FMT qualifier to right-justify it before sorting.

Another common use of FMT is to display EVAL results with appropriate scaling. Because EVAL expressions are not defined in the file dictionary, the only FMT (or CONV) specifications they have are either the default specifications or the ones you supply in the query.

Specifying output conversions

Related to FMT is the CONV field qualifier. Because FMT and CONV functionality sometimes overlaps, it can be difficult to distinguish between these two. FMT specifies output formatting of the data that affects only how it is displayed. CONV specifies a conversion process to be performed on data as it is extracted from the field or stored (through non-RetrieVe means) into the field.

Dates are a good example of the use of CONV. Internally, all dates are stored as an integer representing the number of days relative to Day 0 (in UniVerse, Day 0 is December 31, 1967). When a date is retrieved from this field, it must be converted to a conventional form that is recognizable to the user.

You define the conversion through a conversion specification consisting of one or more conversion codes. If you use multiple conversion codes, the codes are applied from left to right, so that the leftmost code is applied to the original value, then the next conversion code to the right is applied to the result of the first conversion, and so on.

Generally, conversion codes are identical to those used in the `ICONV` and `OCONV` functions of BASIC.

You can tell the type of conversion from the conversion code:

- Date conversions begin with D.
- Time conversions begin with MT.
- Character conversions begin with MC (masked character conversion).
- Numeric conversions begin with MD (masked decimal conversion), ML (masked left conversion), or MR (masked right conversion).

There are also codes for converting data stored as packed decimal, hexadecimal, octal, and binary. Some typical conversions are shown in the following tables, but you should experiment with various combinations to get a sense of what you can do with CONV.

Date conversions

A value of July 2, 1995 in the DATE field of the ENGAGEMENTS.F file is used as the source value.

Conversion code	Explanation	Example
D4/	Month, day, and four-digit year, separated by slashes	07/02/1995
DWAL	Day of the week in initial caps	Sunday
DQ	Quarter	3
DMA[3]	Name of month, abbreviated to three characters	JUL
DYA	Name of Chinese calendar year	SHEEP
DDMY2[Z,A3] or DDMY[Z,A3,2]	Day, month, and two-digit year, with leading zeros in day suppressed, and month abbreviated to three characters	2 JUL 95
DYJ	Year and day number within the year	1995 188

Time conversions

A value of 5 pm in the TIME field of the ENGAGEMENTS.F file is used as the source value.

Conversion code	Explanation	Example
MTH	12-hour format, with a suffix of am or pm	05:00pm
MTS	24-hour format showing seconds	17:00:00
MTS.	24-hour format showing seconds, with elements separated by. (period)	17.00.00

Numeric conversions

Numeric conversions can be masked character (MC), masked decimal (MD), formatting numbers, and so on.

Conversion code	Explanation	Example
MCU	Converts all lowercase letters to uppercase.	CLARK, KELLY
MC/A	Masks all non-alphabetic characters in a field.	7880 (from a street address)
MC/N	Masks all nonnumeric characters in a field.	Stratford Street (from a street address)
MD22	Allows two decimal positions, and scales the stored data by moving its decimal point two places to the left.	5.3465 becomes .05
MD2,$C	Allows two decimal positions, inserts a comma every three digits to the left of the decimal position, prefaces the output with a $, and shows all negative values with a suffix of cr.	45.23- becomes $45.23cr
MR2,D$*14	Allows two decimal positions, inserts a comma every three digits to the left of the decimal position, suffixes all positive values with db, prefaces the output with a $, and right-justifies the entire result within a 12-character field with a left fill of asterisks.	23582.49 becomes **$23,582.49db right-justified

Some ways to use CONV

One common way to use CONV is for dollars-and-cents amounts, for example, to display a value such as COST with a dollar sign, two decimal places, and commas every third position to the left of the decimal position. Such a conversion code has already been supplied for COST in its file dictionary definition, but if it had not been, you could use CONV MD$2,:

```
>SORT INVENTORY.F BY ITEM.CODE ITEM.CODE EVAL "COST * QOH"
   CONV MD$2, ID.SUP
COST * QOH
SORT INVENTORY.F BY ITEM.CODE ITEM.CODE EVAL "COST * QOH" CONV MD2$, ID.SUP
09:07:47AM  31 May 1995  PAGE    1
ITEM.CODE    COST * QOH..

        1       $9,768.84
        2       $6,725.88
        3       $1,121.33
        4       $2,229.21
        .
        .
        .
45 records listed.
```

Sometimes the results of an EVAL expression need a CONV specification to cause the appropriate number of significant digits to be displayed. For example, if you calculate markup as (PRICE – COST)/COST and do not specify a conversion, you would get the following, because the conversion for the EVAL expression defaults to that defined for PRICE in the file dictionary (which happens to be MD22):

```
>LIST INVENTORY.F PRICE COST EVAL "(PRICE - COST)/COST"

LIST INVENTORY.F PRICE COST EVAL "(PRICE - COST)/COST" 01:12:14PM 31 May
1995  PAGE    1
INVENTORY.F    PRICE.....    COST......    (PRICE - COST)/COST

14                $99.36        $80.78                $0.00
16                $45.78        $34.95                $0.00
17                              $28.61               $-0.01
28               $143.55        $98.32                $0.00
29                $76.51        $48.73                $0.01
3                 $20.13        $13.51                $0.00
30                $64.40        $43.81                $0.00
32                $57.33        $42.78                $0.00
.
.
.
45 records listed.
```

To correct this, specify a conversion of MD40 (four decimal places, with a scale of 0, which overrides the dictionary's scale of 2):

```
>LIST INVENTORY.F PRICE COST EVAL "(PRICE - COST)/COST" CONV MD40

LIST INVENTORY.F PRICE COST EVAL "(PRICE - COST)/COST" CONV MD40 01:15:00PM
31 May 1995  PAGE    1
INVENTORY.F    PRICE.....    COST......    (PRICE - COST)/COST

14                $99.36        $80.78                0.2300
16                $45.78        $34.95                0.3099
17                              $28.61               -1.0000
28               $143.55        $98.32                0.4600
29                $76.51        $48.73                0.5701
3                 $20.13        $13.51                0.4900
30                $64.40        $43.81                0.4700
32                $57.33        $42.78                0.3401
.
.
.
45 records listed.
```

As another example of the use of CONV, the `LIST` command in the following query displays the DATE field in the ENGAGEMENTS.F file and then displays it again as the day of the week in combined lowercase and uppercase format:

```
>LIST ENGAGEMENTS.F LOCATION.CODE DATE DATE CONV "DWAL"
   BY LOCATION.CODE BY DATE ID.SUP

LIST ENGAGEMENTS.F LOCATION.CODE DATE DATE CONV "DWAL" BY LOCATION.CODE BY
DATE ID.SUP 01:18:38PM  31 May 1995  PAGE    1
LOCATION.CODE    DATE.....    DATE.....

CCLE001            02/19/94      Saturday
CCLE001            02/20/94        Sunday
CCLE001            08/19/94        Friday
CCLE001            08/20/94      Saturday
CCLE001            12/15/95        Friday
CCLE001            12/16/95      Saturday
CCLE001            03/25/96        Monday
CCLE001            03/26/96       Tuesday
CDET001            08/15/94        Monday
CDET001            08/16/94       Tuesday
CDET001            04/22/95      Saturday
.
.
.
252 records listed.
```

If a conversion code has been supplied for a field in its file dictionary definition, and you want that conversion to be ignored (that is, you want to display the data without conversion), include a CONV " " for that field in your query.

Customizing column headings

When you define a field, you can also define a column heading for it in the file dictionary. If no column heading is specified, then the heading defaults to the field name (actually, the record ID of the field's definition in the file dictionary).

You can override either of these by using the COL.HDG field qualifier in your query. For example, if the column heading for the COST field is defined by default as "Cost" and you want to make it "Amount Paid" for this query only, enter:

```
>LIST INVENTORY.F DESCRIPTION QOH COST COL.HDG "Amount'L'Paid"
   BY DESCRIPTION ID.SUP

LIST INVENTORY.F DESCRIPTION QOH COST COL.HDG "Amount"L"Paid" BY
DESCRIPTION ID.SUP 01:22:26PM  31 May 1995  PAGE    1
                                    Amount....
DESCRIPTION.............    QOH..   Paid......

Balloons                     77       $43.81
Beer                        127       $76.92
Bird Seed                    94       $84.19
.
.
.
T-shirts                    171       $49.60
Taffy                        71       $11.64
Ticket Stock                 96       $57.13

45 records listed.
```

The 'L' in the input specifies that the column heading is to be broken into two lines at that point.

The column heading for a virtual field is the EVAL expression itself, which is probably not what you really want as the heading. For example, if you include in a query an EVAL expression that calculates the cost of each of the products on hand by multiplying the quantity available by the cost, the output looks like this:

```
LIST INVENTORY.F DESCRIPTION EVAL "COST * QOH" BY DESCRIPTION ID.SUP
01:25:47PM  31 May 1995  PAGE    1
DESCRIPTION.............    COST * QOH

Balloons                     $3,373.37
Beer                         $9,768.84
Bird Seed                    $7,913.86
Bunting                      $3,570.06
Candy Selection              $5,776.30
Cheese Slices               $14,907.49
.
.
.
T-shirts                     $8,481.60
Taffy                          $826.44
Ticket Stock                 $5,484.48

45 records listed.
```

To change the heading to read Reorder Cost, add a COL.HDG to the query:

```
>LIST INVENTORY.F DESCRIPTION EVAL "COST * QOH"
   COL.HDG "Reorder Cost" BY DESCRIPTION ID.SUP
COST * QOH

LIST INVENTORY.F DESCRIPTION EVAL "COST * QOH" COL.HDG "Reorder Cost" BY
DESCRIPTION ID.SUP 01:26:58PM  31 May 1995  PAGE    1
DESCRIPTION.............    Reorder Cost

Balloons                     $3,373.37
Beer                         $9,768.84
Bird Seed                    $7,913.86
Bunting                      $3,570.06
Candy Selection              $5,776.30
Cheese Slices               $14,907.49
.
.
.
T-shirts                     $8,481.60
Taffy                          $826.44
Ticket Stock                 $5,484.48

45 records listed.
```

Creating an alias

The AS field qualifier specifies a synonym, or alias, for a field or virtual field. The main reason for using it is to shorten what would otherwise be a cumbersomely long name, as in the following situations:

- Field names: If a field name is overly long, assigning it an alias avoids having to reenter the long string if you want to refer to the field later in the same query.
- EVAL expressions: If you used an EVAL expression and must refer to it later in the same query, assigning it an alias allows you to use that alias instead of reentering the entire expression.

A good example of how an alias saves typing is when you included an EVAL expression in the query and you want to sort on the results produced by it. Without an alias, sorting a report on the basis of an EVAL expression forces you to type the EVAL expression twice:

```
>LIST INVENTORY.F PRICE COST EVAL "(PRICE - COST) * QOH"
   BY.DSND EVAL "(PRICE - COST) * QOH"
( PRICE - COST ) * QOH
( PRICE - COST ) * QOH

LIST INVENTORY.F PRICE COST EVAL "(PRICE - COST) * QOH" BY.DSND EVAL
"(PRICE - COST) * QOH" 01:33:03PM  31 May 1995  PAGE    1
INVENTORY.F    PRICE.....    COST......    (PRICE-COST)*QOH

  42                          $88.21
  37                          $96.36
  5              $149.10      $102.83          $8,559.95
  28             $143.55       $98.32          $7,870.02
  43             $130.29       $90.48          $7,205.61
  31             $126.85       $79.78          $6,731.01
  .
  .
  .
  8               $14.90       $11.64              $231.46
  44              $25.96       $23.60              $143.96
  17                           $28.61            $-4,005.40

  45 records listed.
```

By assigning an alias to the first occurrence of the EVAL expression, you can use the alias in place of the second occurrence:

```
>LIST INVENTORY.F PRICE COST EVAL "(PRICE - COST) * QOH" AS PROFIT
   BY.DSND PROFIT
( PRICE - COST ) * QOH

LIST INVENTORY.F PRICE COST EVAL "(PRICE - COST) * QOH" AS PROFIT BY.DSND
PROFIT 01:38:05PM  31 May 1995  PAGE    1
INVENTORY.F    PRICE.....    COST......    (PRICE - COST) * QOH

42                          $88.21
37                          $96.36
5              $149.10      $102.83          $8,559.95
28             $143.55       $98.32          $7,870.02
43             $130.29       $90.48          $7,205.61
31             $126.85       $79.78          $6,731.01
.
.
.
45 records listed.
```

Defining temporary associations and structures

ASSOC, ASSOC.WITH, SINGLE.VALUE, and MULTI.VALUE are a few more field qualifiers that you may find handy in certain situations. These qualifiers can be used to change the singlevalued/multivalued nature of a field and the fields with which it is associated for the purpose of controlling output layout.

Using ASSOC and ASSOC.WITH for better report layouts

ASSOC associates any field with an existing association of multivalued fields, causing the field to be treated as part of the association for the duration of the query. One use of this facility is to produce more readable reports. As an example, compare the following two outputs:

```
>LIST PERSONNEL.F BY BADGE.NO WITH NAME LIKE T... VERT
   NAME DEP.NAME

LIST PERSONNEL.F BY BADGE.NO WITH NAME LIKE T... VERT NAME DEP.NAME
02:17:24PM  31 May 1995  PAGE    1

PERSONNEL.F. 38
NAME........ Tucker, Alfred
DEP.NAME.... Isabel
           . Nancy

PERSONNEL.F. 40
NAME........ Tuo, Chang
DEP.NAME....

PERSONNEL.F. 41
NAME........ Tucker, Joe
DEP.NAME.... Beverly
           . Brenda

PERSONNEL.F. 80
NAME........ Torres, Stephen
DEP.NAME.... Patricia
           . Cecilia
.
.
.
7 records listed.
```

Because the format is vertical (VERTICALLY is a report qualifier that specifies a vertical orientation), and NAME and DEP.NAME are unrelated, each appears on its own line. But if you create an association between NAME and DEP.ASSOC (the association of which DEP.NAME is a part), both can share the same line, giving a more compact and pleasing appearance and saving space:

```
>LIST PERSONNEL.F BY BADGE.NO WITH NAME LIKE T... VERT
   NAME ASSOC "DEP.ASSOC" DEP.NAME

LIST PERSONNEL.F BY BADGE.NO WITH NAME LIKE T... VERT NAME ASSOC
"DEP.ASSOC" DEP.NAME 02:19:59PM 31 May 1995  PAGE    1

PERSONNEL.F. 38
NAME..................... DEP.NAME..
Tucker, Alfred           Isabel
                         Nancy

PERSONNEL.F. 40

 NAME..................... DEP.NAME..
 Tuo, Chang

 PERSONNEL.F. 41
 NAME..................... DEP.NAME..
 Tucker, Joe              Beverly
                          Brenda

 PERSONNEL.F. 80
 NAME..................... DEP.NAME..
 Torres, Stephen          Patricia
                          Cecilia
 .
 .
 .
 7 records listed.
```

ASSOC.WITH is similar, except that you use it to associate a field with a multivalued field instead of an entire association.

Using SINGLE.VALUE and MULTI.VALUE for positioning fields

Using the SINGLE.VALUE field qualifier specifies that a multivalued field is to be treated as singlevalued for this query only. Using the MULTI.VALUE field qualifier does just the opposite. Generally, both field qualifiers (particularly MULTI.VALUE) are used with ASSOC and ASSOC.WITH as a further means of positioning fields in displays and reports. For technical details on using these qualifiers, see the *UniVerse User Reference*.

Copying display characteristics from other fields

If the characteristics (format, conversion, association, single- or multivalued indicator) you want to use for outputting a field are already described for some other field, you can use those characteristics without reentering their specifications. Note that column headings are not included.

Use the DISPLAY.LIKE field qualifier to set the characteristics of a field to those of another field. For example, to display the COST field in the same way that the QOH field is displayed, enter:

```
>LIST INVENTORY.F COST DISPLAY.LIKE QOH COL.HDG "Cost in Cents"

LIST INVENTORY.F COST DISPLAY.LIKE QOH COL.HDG "Cost in Cents" 02:24:10PM
31 May 1995  PAGE    1
INVENTORY.F    Cost in Cents

14                      8078
16                      3495
17                      2861

28                      9832
29                      4873
.
.
.
45 records listed.
```

Because QOH has a default definition of MD0 (numeric field, no decimal places), COST is also displayed as an integer, and so now the value 98.32 is displayed as 9832 cents, as the user-supplied column heading indicates.

Formatting reports with report qualifiers

Using report qualifiers, you can tailor the layout of the entire report by setting up headers and footers on each page, adjusting margins and spacing, and determining output orientation (horizontal or vertical). In addition, there are two RetrieVe commands, `LIST.LABEL` and `SORT.LABEL`, that enable you to format and sort mailing labels.

Using report qualifier keywords

Report qualifiers provide a variety of ways to control and refine the overall format of a report. ID.SUP, DET.SUP, LPTR, SAMPLE, and SAMPLED are report qualifiers you saw in previous examples. For complete information about which report qualifiers apply to specific queries, consult the *UniVerse User Reference*. The following list summarizes the most commonly used report qualifiers:

Keyword	Synonyms	Description
AUX.PORT		Sends output to the printer connected to the terminal's auxiliary port.
COL.HDR.SUPP	COL-HDR-SUPP	Suppresses the default report and column headings.
COL.SPCS	COL.SPACES	Changes the default spacing between columns.
COL.SUP	COL-SUPP	Suppresses the default column heading.
COUNT.SUP		Suppresses the count displayed at the bottom of a report.
DBL.SPC	DBL-SPC	Double-spaces the report.
DET.SUP	DET-SUPP	Displays only the breakpoint lines. Use with BREAK.ON.
FOOTING	FOOTER	Sets the report footing.
GRAND.TOTAL	GRAND-TOTAL	Sets the text for a grand total line. Use with BREAK.ON.

Keyword	Synonyms	Description
HDR.SUP	HDR-SUPP SUPP	Suppresses the default report heading.
HEADING	HEADER	Uses the report header you specify in the query rather than the default heading.
ID.ONLY	ONLY	Displays only record IDs.
ID.SUP	ID-SUP ID-SUPP	Suppresses the display of record IDs.
LPTR		Sends the output to the printer.
MARGIN		Defines the margin for the report.
NO.SPLIT		Starts a record on a new page if it does not fit on the current page.
NOPAGE	NO.PAGE	Specifies that the report is automatically scrolled on the terminal.
SAMPLE	FIRST	Displays the first *n* records.
SAMPLED		Displays every *n*th record.
VERTICALLY	VERT	Displays the report in vertical format, with one field on each line.

The DET.SUP, GRAND.TOTAL, ID.ONLY, ID.SUP, LPTR, SAMPLE, SAMPLED, and VERTICALLY report qualifiers were covered earlier. The following sections deal with some of the other commonly used report qualifiers.

Specifying report headings and footings

A report heading is displayed at the top of each page of a report; the footing is displayed at the bottom of each page. As you have seen in the examples, the standard (default) heading is the text of the query you enter, time, date, and page number, as follows:

```
>LIST INVENTORY.F SAMPLE 3 DESCRIPTION QOH

LIST INVENTORY.F SAMPLE 3 DESCRIPTION QOH 02:48:38PM 31 May 1995  PAGE    1
INVENTORY.F    DESCRIPTION.............    QOH..

14             Ice Cream, Various            154
16             French Fries, Frozen           51
17             Nachos                        140

Sample of 3 records listed.
```

If you do not like the standard header, you can use the HEADING report qualifier to design your own. Besides supplying the text of the header, you can also include one or more options that provide additional features. A full list is in the *UniVerse User Reference*, but a few of the more commonly used ones include:

Option	Description
C[*n*]	Centers the heading in a field of *n* spaces.
D	Inserts the current date.
G	Inserts gaps in the format to spread the heading across the entire width of the screen or page.

Option	Description
I[*n*] or R[*n*]	Inserts the current record ID, left-justified, in a field of *n* spaces.
P[*n*]	Inserts the page number, left-justified, in a field of *n* spaces.
T	Inserts the current time and date.

Here is one example of a customized report header, where you ask that the heading be centered (C) on the page and that the time (T) be included:

```
>LIST INVENTORY.F SAMPLE 3 DESCRIPTION QOH
   HEADING "Inventory On Hand 'C' 'T'"
                  Inventory On Hand  02:57:42PM 31 May 1995
INVENTORY.F    DESCRIPTION.............    QOH..

14             Ice Cream, Various            154
16             French Fries, Frozen           51
17             Nachos                        140

Sample of 3 records listed.
```

You can make the heading more readable by inserting descriptive text to label elements such as date and page number:

```
>LIST INVENTORY.F SAMPLE 3 DESCRIPTION QOH
   HEADING "Inventory On Hand   Date: 'D'   Page 'P3'"

Inventory On Hand   Date: 31 May 1995  Page   1
INVENTORY.F    DESCRIPTION.............    QOH..

14             Ice Cream, Various            154
16             French Fries, Frozen           51
17             Nachos                        140

Sample of 3 records listed.
```

You may even want to omit the header entirely. To suppress the display or printing of a header, use the HDR.SUP report qualifier as shown:

```
>LIST INVENTORY.F SAMPLE 3 DESCRIPTION QOH HDR.SUP
ITEM.CODE    DESCRIPTION.............    QOH..

INVENTORY.F    DESCRIPTION.............    QOH..

14             Ice Cream, Various            154
16             French Fries, Frozen           51
17             Nachos                        140

Sample of 3 records listed.
```

Footers, like headers, can also be customized and have options that are almost identical to those for headers.

Listing reports vertically

Normally, RetrieVe tries to fit the output to the medium, so that it adopts a horizontal format if all the fields can fit on one line, or switches to a vertical format if that is not the case. However, you can force the use of a vertical format by using the VERTICALLY report qualifier to help make reports readable

when many fields are specified in the output. For example, the following query shows the output when all the fields are listed on the same line:

```
>LIST EQUIPMENT.F DESCRIPTION COST DEPRECIATION FMT 3L COL.HDG "DEP"
   PURCHASE.DATE

LIST EQUIPMENT.F DESCRIPTION COST DEPRECIATION FMT 3L COL.HDG "DEP"
PURCHASE.DATE 03:02:14PM  31 May 1995  PAGE    1
EQUIPMENT.F   DESCRIPTION.............   COST........   DEP   PURCHASE.DATE

14            Coffee/cookies Stand          $67,521.49   C          12/16/91
16            Wild West Photo Stand          $5,016.53   D          08/22/92
17            Glamor Photo Stand            $41,324.40   Z          08/21/93
2             Hot Dog Stand                 $58,555.15   J          03/29/89
28            Truck 897 M X X               $66,700.54   F          06/07/90
29            Merry-Go-Round               $125,000.00   P          01/15/95
3             Horseshoes Stand              $28,662.27   I          05/28/89
.
.
.
61 records listed.
```

Because this produces a very wide report, which might not be readable on a small screen, you could use a vertical orientation, as follows:

```
>LIST EQUIPMENT.F DESCRIPTION COST DEPRECIATION FMT 3L
   COL.HDG "DEP" PURCHASE.DATE VERTICALLY

LIST EQUIPMENT.F DESCRIPTION COST DEPRECIATION FMT 3L COL.HDG "DEP"
PURCHASE.DATE VERTICALLY 03:04:30PM  31 May 1995  PAGE    1

EQUIPMENT.F... 14
DESCRIPTION... Coffee/cookies Stand
COST.........   $67,521.49
DEP.......... C
PURCHASE.DATE.   12/16/91

EQUIPMENT.F... 16
DESCRIPTION... Wild West Photo Stand
COST.........    $5,016.53
DEP.......... D
PURCHASE.DATE.   08/22/92

EQUIPMENT.F... 17
DESCRIPTION... Glamor Photo Stand
COST.........   $41,324.40
DEP.......... Z
PURCHASE.DATE.   08/21/93
.
.
.
61 records listed.
```

Associated fields in a vertical listing are shown on separate lines. In the following example, ITEM.CODE and LEAD.TIME are both multivalued fields, and their individual values are listed on separate lines:

```
>LIST VENDORS.F ITEM.CODE LEAD.TIME VERTICALLY

LIST VENDORS.F ITEM.CODE LEAD.TIME VERTICALLY 03:05:46PM  31 May 1995  PAGE
1

VENDORS.F. 159
ITEM.CODE LEAD.TIME
       18        66
       38        37
       44        31
        5        83
       16        43
       30        42
        6        84
       43        32
                 46

VENDORS.F. 175
ITEM.CODE LEAD.TIME
       39        20
       33        47
        1        39
       10        75
        .
        .
        .
232 records listed.
```

Creating mailing labels

Use the LIST.LABEL and SORT.LABEL commands to format and sort mailing labels. These queries work like LIST and SORT, except that the output is organized in a label layout. Instead of printing data organized by columns, LIST.LABEL and SORT.LABEL group data from each record in fixed-size blocks, or labels. SORT.LABEL creates labels sorted by record IDs.

Specify the output in LIST.LABEL and SORT.LABEL as you do in other queries. The output specification in LIST.LABEL or SORT.LABEL identifies which fields you want included in the block listing. For example, to do a mailing to all of your vendors, you would create mailing labels by entering the following:

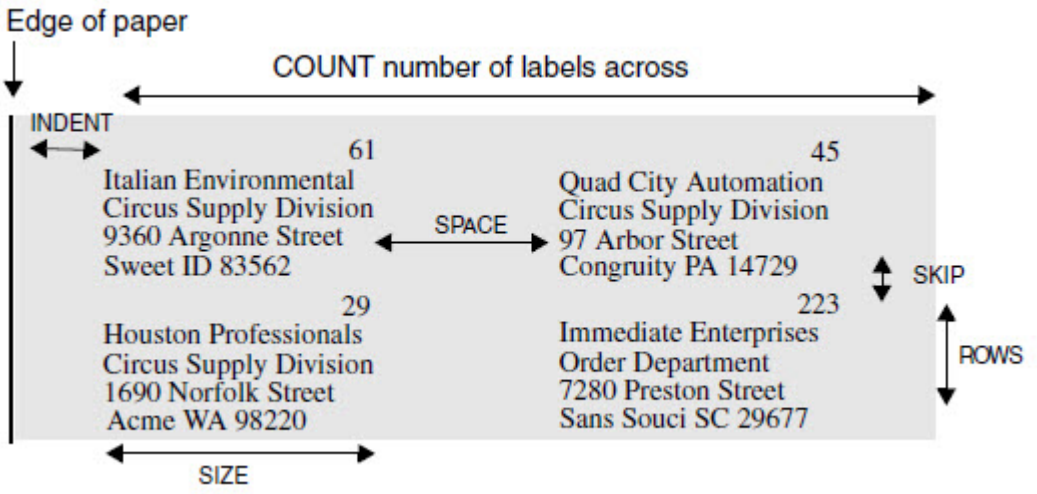
```
>LIST.LABEL VENDORS.F COMPANY ADR1 ADR2 ADR3 HDR.SUP
```

As in other queries, you can select records to be processed by using a list of record IDs, a select list, or a selection expression.

When you run LIST.LABEL or SORT.LABEL, you are prompted to enter the options that define the label layout:

```
>LIST.LABEL.F VENDORS COMPANY ADR1 ADR2 ADR3 HDR.SUP NOPAGE
COUNT, ROWS, SKIP, INDENT, SIZE, SPACE [ ,C ] ?
```

The following figure illustrates these options.



Note: When printing labels, include the NOPAGE report qualifier because normally you do not want the `Press any key to continue...` message to print at the bottom of each page.

The following list describes each option:

Option	Description
COUNT	The number of labels (records) across the page or screen.
ROWS	The number of lines (fields) listed per label.
SKIP	The number of lines to skip vertically between labels.
INDENT	The number of indented spaces from the left margin to the label.
SIZE	The maximum number of characters in any display field.
SPACE	The number of horizontal spaces between labels.
C	Do not print empty fields. (If you do not include C, empty fields are printed as a series of blanks.)

When you specify the label options, follow these guidelines to make sure that the options you specify are compatible with the physical layout of your output device:

- The SIZE option cannot exceed the page width (80 columns for most terminals, and 80 or 132 columns for printers).
- The total width specifications cannot exceed the width of the output device (generally the printer). Use the following formula to compute the total width:
- INDENT + COUNT(SIZE + SPACE)
- If you use NLS mode and want to print labels in languages with wide characters that print as two display positions on a screen, such as Korean, you must allow for this in your calculations. (Make similar calculations when you plan column headings, footings, and data that you do not want to wrap.) For more information about NLS, see the *UniVerse NLS Guide*.

Suppose that the mailing labels you want to produce by the previous query have the following specifications:

- Two labels across the page
- Five fields on each label (do not forget to count the record ID unless you suppress it)
- Four lines between labels
- Two spaces indented from the left margin

- Maximum of 25 characters in each field
- Three spaces between labels across the page
- Do not print empty fields

Given these specifications, you would enter the following values in response to the parameters prompt:

```
COUNT, ROWS, SKIP, INDENT, SIZE, SPACE [ ,C ] ? 2,5,4,2,25,3,C
```

RetrieVe then asks you to enter the row header for each row. Because they are not needed for this example, just press Enter after each prompt (or you could have specified HDR.SUP in the original query, which would eliminate this series of prompts).

The output resembles the following:

```
                    61                                  45
Italian Environmental               Quad City Automation
Circus Supply Division              Circus Supply Division
9360 Argonne Street                 97 Arbor Street
Sweet    ID 83562                   Congruity    PA 14729



                   29                        223
Houston Professionals                Immediate Enterprises
Circus Supply Division               Order Department
1690 Norfolk Street                  7280 Preston Street
Acme     WA 98220                    Sans Souci   SC 29677



                     207                                 191
United Advisers                      Convenient Promotions
Order Department                     3900 Brazos Street
1510 Rosewood Street                 Access   OH 43901
Access   OH 43901
```

Chapter 4: Creating and using select lists

This chapter describes creating and using select lists, and introduces you to the RetrieVe commands SELECT, SSELECT, and SEARCH (or ESEARCH).

Constructing queries, on page 24 and Customizing query output, on page 60 discussed how you can select records on the basis of record IDs, sampling, field values, and comparing one field to another. Another way to select records is by using select lists.

A select list contains the record IDs of records that meet specified criteria. Select lists can be used with RetrieVe commands, UniVerse BASIC programs, the UniVerse Editor, ReVise, and other UniVerse utilities.

Why use select lists?

Using a select list saves time because, once you have obtained the select list, you can operate on it instead of repeatedly searching the entire file. Because you can save a select list for future use, once you have created a select list and saved it, you will never have to create that select list again.

In programming terms, you can think of a select list as a string of pointers to records in a file. Suppose you are working with a sales file containing information organized by branch offices, and you are interested only in the branch offices in Australia. You can create a select list made up of record IDs of those records relating to branch offices in Australia. Then you can use one query to produce a report of the branch offices sorted by the highest volume sales and another query to produce another report listing the branch offices sorted by size. This procedure avoids reading the entire sales file for each query.

Creating select lists

Select lists are created using the RetrieVe commands SELECT, SSELECT, and SEARCH (or ESEARCH), as well as the non-RetrieVe commands NSELECT, QSELECT, and FORM.LIST. The SELECT, SSELECT, and SEARCH commands create a select list to hold the record IDs or field values of those records selected by the command. Use SSELECT if you want the select list in order by record ID. After the select list is created, it is available for use by a subsequent command.

You can have up to 11 active select lists (numbered 0 through 10) at one time. Unless you specify that a number be assigned to the select list you are creating, the number defaults to 0. If you assign a number, it can be a number from 1 through 10.

When a select list is available for use, it is called an active select list. Select list 0 is unique in that it is available (active) only for the next command, and you must use it then or the system deletes it. As you will see later, one option is to issue a SAVE.LIST command which saves the select list under a name so that you can recall it later. A select list numbered 1 through 10 is active until you issue a command that contains a FROM *n* clause (where *n* is the number of the select list) or you terminate the current session.

Returning to the Circus database, you could issue the following SELECT command to create a select list containing the record IDs of all vendors in the 208 area code:

```
>SELECT VENDORS.F WITH PHONE LIKE 208

... 8 record(s) selected to SELECT list #0.

>>
```

Because you have used SELECT, the output is in the form of a select list rather than a display or printed report. Notice that the prompt changes to >> to indicate that select list 0 is active and

available for use. For instance, you could use the newly created select list to print the names of the vendors in area code 208:

```
>>LIST VENDORS.F COMPANY

LIST VENDORS.F COMPANY 10:58:30AM 31 May 1995  PAGE    1
VENDORS.F.    COMPANY.................

61            Italian Environmental        208/748-6410
135           General Link                 208/730-7712
103           Grant Rentals                208/776-9628
131           Elite Salvage                208/850-1095
113           Commerce Exchange            208/730-4703
71            Reliable Wholesale           208/693-8859
201           Travelers House              208/423-2273
64            Paris House                  208/290-5374

8 records listed.
```

Creating numbered select lists

You can create up to 11 select lists, numbered 0 through 10. Select list 0 is the default, and is the one created when you do not indicate otherwise.

To create other numbered select lists, you must assign them a number from 1 through 10 using the TO clause, as follows:

```
>SELECT ACTS.F TO 4

6 record(s) selected to SELECT list #4.
```

Creating select lists of field values

The previous example used a select list built from the record IDs of selected records. You can also build a select list from some field other than the record ID and then use the values in that select list as search arguments against record IDs in another file.

Assume that you want a list of all the companies from which you have purchased equipment prior to 1990. You know that the EQUIPMENT.F file contains the vendor code for each piece of equipment owned by the circus, and the VENDORS.F file contains the company name of each vendor and has vendor code as its record ID. By creating a select list of the vendor codes in the EQUIPMENT.F file, you can use that select list as pointers to the matching records in the VENDORS.F file.

To save a field other than the record ID in a select list, use the SAVING keyword followed by the name of the field whose contents you want to place in the select list. To do this, first create a select list of the vendor codes from the records for equipment purchased before 1990. To indicate that it is VENDOR.CODE you want to store in the select list and not the record ID, you must include a SAVING clause:

```
>SELECT EQUIPMENT.F WITH PURCHASE.DATE < '01/01/90'
 SAVING VENDOR.CODE


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

To get a list of the company names and phone numbers from the VENDORS.F records that correspond to the select list values, all you would have to do is enter:

```
>>LIST VENDORS.F COMPANY PHONE ID.SUP

LIST VENDORS.F COMPANY PHONE ID.SUP 11:00:41AM 31 May 1995  PAGE    1
COMPANY.................    PHONE.......

Midtown Innovations          603/932-5898
Progressive Mart             603/433-5428
Convenient Promotions        513/221-4535
Action Operators             606/697-5680
United Advisers              513/643-1632
.
.
.
Miami Acceptances            715/645-4945
Reliable Wholesale           208/693-8859

15 records listed.
```

This command uses the active select list as a list of record IDs against the VENDORS.F file and retrieves and lists the names and phone numbers from those records. For a further discussion of using select lists as record IDs, refer to Using select lists as record IDs, on page 99.

Creating select lists with multivalued fields

To create a select list with multivalued fields, you must use an exploded sort with the WHEN clause just as you do when doing any other record selection on multivalued fields. Otherwise, you get values that do not satisfy the selection criteria.

To produce a select list with only those multivalues equal to or greater than 900, you need to use the BY.EXP keyword to explode the multivalues of ORDER.QTY along with the WHEN clause:

```
>SELECT INVENTORY.F BY.EXP ORDER.QTY WHEN
ORDER.QTY >= 900

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

Now if you use this select list against the VENDORS.F file, you will get a correct listing:

```
>>LIST VENDORS.F COMPANY BY COMPANY ID.SUP

LIST VENDORS.F COMPANY BY COMPANY ID.SUP 11:15:31AM 31 May 1995  PAGE    1

COMPANY.................

Acme Brothers
Advantage Selections
Amalgamated Academy
American International
Baltimore Energy
Beneficial Mart
Blue Selections
Boston Equipment
Chicago World
Columbus Interfaces
Columbus Interfaces
Detroit Stockists
Gray Merchandise
Green Producers
Philadelphia Stores
San Diego Promotions
San Francisco
Distributors
Toledo Energy
Twin Cities Resources

19 records listed.
```

Using SEARCH to select on character strings

SEARCH is another command that creates a select list, based on records that contain a specified string of characters anywhere in the record, rather than in a specific field.

An advantage of using SEARCH is that you do not have to know which field the string is in—the SEARCH command tests all the fields (except for the record ID field) for the string. However, a disadvantage is that with large records in large files, SEARCH can be very slow and is not the most efficient way to find what you are looking for. In many cases, SEARCH should be used only as a last resort. SEARCH, like most selection processes, is case-sensitive.

For example, to search the PERSONNEL.F file for the string San, regardless of in what fields it occurs, and create a select list of those records, enter:

```
>SEARCH
PERSONNEL.FSTRING: San
<Enter>STRING: <Enter>
>>

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

Then, if you are interested in looking at only the name and address fields of the records in which the string was found, you could follow with the command:

```
>>LIST PERSONNEL.F NAME ADR1 ADR2 ADR3

LIST PERSONNEL.F NAME ADR1 ADR2 ADR3 11:38:25AM 31 May 1995  PAGE    1

PERSONNEL.F. 152
NAME........ Hanson, Allen
ADR1........ 1260 San Jacinto Street
ADR2........ Beautiful  PA 15009
ADR3........

PERSONNEL.F. 184
NAME........ Hill, Sandra
ADR1........ 840 Locke Street
ADR2........ Carefree  AZ 85719
ADR3........

PERSONNEL.F. 58
NAME........ Sousa, Evelyn
ADR1........ 9610 Mcgowen Street
ADR2........ Sans Souci  SC 29677
ADR3........
.
.
.
132 records listed.
```

Saving a select list for future use

A third option for the command following a `SELECT` or `SSELECT` is to save the select list for future use. If you want to save the select list instead of immediately acting on it, use this syntax:

Syntax

`SAVE.LIST` *`listname`*

listname is the name by which the saved list can be recalled. The saved select lists are stored in the &SAVEDLISTS& file in your account.

Then, later, if you wanted to use this list in a RetrieVe command, all you would have to do is activate it with this syntax:

`GET.LIST` *`listname`*

Now you can use it, as if you had just created it.

For example, assume that you created the “vendors in area code 208” list and saved it:

```
>SELECT VENDORS.F WITH PHONE LIKE 208...

8 record(s) selected to SELECT list #0.
>>SAVE.LIST AREA.208

8 record(s) SAVEd to SELECT list "AREA.208".
```

Now when you want to work with vendors in area code 208, just activate this select list:

```
>GET.LIST AREA.208

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

Using select lists in commands

After you create select list 0, it is available for use by the next command, or you can save it for later use as shown in the preceding section. Select lists 1 through 10 are active until you use them in a subsequent command or until you end the current session. They also can be saved.

Many UniVerse commands can use or operate on a select list. Some of the more common ones include:

LIST	SORT	SREFORMAT
LIST.ITEM	SORT.ITEM	T.DUMP
LIST.LABEL	SORT.LABEL	
REFORMAT	SPOOL	

If you wanted to select vendors in the 208 area code and then sort those records in descending order by TERMS (so that the more lenient payment terms appear first), you could enter:

```
>SELECT VENDORS.F WITH PHONE LIKE 208...

8 record(s) selected to SELECT list #0.
>>LIST VENDORS.F TERMS PHONE BY.DSND TERMS

LIST VENDORS.F TERMS PHONE BY.DSND TERMS 11:55:16AM 31 May 1995  PAGE    1
VENDORS.F.    TERMS.....    PHONE.......

64            Net 40        208/290-5374
61            Net 30        208/748-6410
131           Net 30        208/850-1095
71            Net 25        208/693-8859
201           Net 25        208/423-2273
135           Net 25        208/730-7712
103           Net 25        208/776-9628
113           Net 20        208/730-4703

8 records listed.
```

Select lists can contain values other than record IDs or field values from a file. The next subsection shows how you create a select list of file names for use by another command.

Using select lists as record IDs

For the most part, the examples up to now have created a select list of record IDs and then used it to select records from the same file. As shown in an earlier example, where a select list of vendor codes was created from the INVENTORY.F file and then used to list company names from the VENDORS.F file, there are times when you will want to create a select list of field values that you can use as record IDs to select records from some other file. To do this, add the SAVING clause to the `SELECT` command to tell it to build the select list from the values of some named field rather than from the record ID values.

Take another example, where you create a select list of vendor codes from the EQUIPMENT.F file of vendors from whom you purchased equipment prior to 1990:

```
>SELECT EQUIPMENT.F WITH PURCHASE.DATE < '01/01/90'
 SAVING VENDOR.CODE
15 record(s) selected to SELECT list #0.
>>
```

At this point, select list 0 contains the vendors from which you purchased all pre-1990 equipment. Using that select list as a source of record IDs, you can now use that select list against the VENDORS.F file to retrieve the company names and phone numbers of those vendors as follows:

```
>>LIST VENDORS.F COMPANY PHONE BY COMPANY ID.SUP

LIST VENDORS.F COMPANY PHONE BY COMPANY ID.SUP 12:04:13PM 31 May 1995  PAGE
1
COMPANY.................    PHONE.......

Amalgamated Academy          218/728-5942
Budget Producers             607/587-8826
Century Group                517/829-6284
Custom Group                 601/888-3672
Dallas Equipment             804/636-8988
General Link                 208/730-7712
Greek Cousins                501/750-9837
Kwik Outlets                 814/647-2368
Kwik Outlets                 814/647-2368
London Treating              615/691-1943
Miami Acceptances            715/645-4945
Ohio Treating                615/757-2533
Progressive Mart             603/433-5428
Reliable Wholesale           208/693-8859
Yale Outlets                 719/357-1851

15 records listed.
```

Note that Kwik Outlets appears twice because two items were purchased from them prior to 1990.

Using select lists of file names

Some commands can use a select list as a source of file names instead of record IDs.

You could, for example, create a select list with all file names in the VOC file that start with the text UV and then use `ANALYZE.FILE` to analyze the contents of the files in the list. The following example shows both steps:

```
>SELECT VOC WITH NAME LIKE UV...

10 record(s) selected to SELECT list #0.
>>ANALYZE.FILE
File name ..................   UV_SCHEMA
Pathname ...................   /u1/uv/sql/catalog/UV_SCHEMA
File type ..................   DYNAMIC
Hashing Algorithm ..........   GENERAL
No. of groups (modulus) ....   1 current ( minimum 1 )
Large record size ..........   80 bytes
Group size .................   2048 bytes
Load factors ...............   80% (split), 50% (merge) and 8% (actual)
Total size .................   8192 bytes



File name ..................   UV_COLUMNS
Pathname ...................   /u1/uv/sql/catalog/UV_COLUMNS
File type ..................   DYNAMIC
Hashing Algorithm ..........   GENERAL
No. of groups (modulus) ....   512 current ( minimum 1 )
Large record size ..........   80 bytes
Group size .................   2048 bytes
Load factors ...............   80% (split), 50% (merge) and 69% (actual)
Total size .................   1619968 bytes
.
.
.
```

Using numbered select lists in RetrieVe commands

Use the FROM clause to specify the number of a select list. Assume, for example, that you sent the output of a `SELECT` command to select list 3:

```
>SELECT INVENTORY.F SAMPLED 5 TO 3

9 record(s) selected to SELECT list #3.>
```

To use that list in the subsequent command, enter:

```
>LIST INVENTORY.F DESCRIPTION FROM 3

LIST INVENTORY.F DESCRIPTION FROM 3 12:10:38PM 31 May 1995  PAGE    1
INVENTORY.F    DESCRIPTION.............

29             Paper Plates
4              Lemonade
10             Franks
22             Egg Rolls
31             Programs
40             Ticket Stock
42             Cheese Slices
23             Sausages
6              Ice Bags

9 records listed.
```

Once you create a numbered select list and then refer to it in some subsequent command, the select list is deleted. The only way to preserve a select list is to save it using the SAVE.LIST command (refer to Saving a select list for future use, on page 98).

Creating a sublist from a select list

Instead of LIST or SORT, the command following a SELECT or SSELECT command could be another SELECT or SSELECT. This allows you to perform a series of select operations, each time narrowing the selection process step by step.

For example, suppose you want to know how many vendors have a vendor code greater than 100, then how many of those have a phone number with an area code of 208, and finally list this latter subgroup in descending order by terms:

```
>SELECT VENDORS.F WITH VENDOR.CODE GT 100

132 record(s) selected to SELECT list #0.
>>SELECT VENDORS.F WITH PHONE LIKE 208...
5 record(s) selected to SELECT list #0.
>>LIST VENDORS.F TERMS PHONE BY.DSND TERMS

LIST VENDORS.F TERMS PHONE BY.DSND TERMS 12:13:41PM 31 May 1995  PAGE    1
VENDORS.F.    TERMS.....    PHONE.......
131           Net 30        208/850-1095
201           Net 25        208/423-2273
103           Net 25        208/776-9628
135           Net 25        208/730-7712
113           Net 20        208/730-4703

5 records listed.
```

First you created a select list of those vendors with a vendor code greater than 100 and then, from that list, you created a sublist of vendors in the 208 area code, and then you listed those vendors in descending order by TERMS. The beauty of this is that the second SELECT command operated on only those records contained in the first select list, not on the entire VENDORS.F file. VENDORS.F is a relatively small file, but you can appreciate the advantages of this when working with files containing hundreds of thousands of records. The reason for the order of SELECTs is that doing pattern matching (LIKE) is a slower process than doing a simple equality type selection on the same number of records.

Several commands create sublists besides SELECT and SSELECT. One is the NSELECT command, which creates a select sublist consisting only of those data elements from the active select list that are not in a specified file. As an example, assume that you have two INVENTORY files, INVENTORY.OLD and INVENTORY.NEW, and you want a list of item descriptions of those items in INVENTORY.NEW that are not in INVENTORY.OLD. To obtain this listing, you would enter the following:

```
>SELECT INVENTORY.NEW

53 record(s) selected to SELECT list #0.
>>NSELECT INVENTORY.OLD TO 1

9 record(s) selected to SELECT list #1.
>>LIST INVENTORY.NEW DESCRIPTION
```

Another approach to doing the same thing would be to use LIST.DIFF, which compares two saved lists and creates a third list of elements that were found in the first select list but not in the second.

Manipulating select lists

There are a number of LIST.*xxx* and other commands that manipulate select lists in various ways:

- LIST.DIFF compares two saved select lists and generates a third select list containing those elements in list 1 that are not in list 2.
- LIST.INTER compares two saved select lists and creates a third select list containing elements from the first select list that are also found in the second select list and are not redundant.
- LIST.UNION compares two saved select lists and creates a third list made up of elements from the first select list followed by elements in the second select list that do not appear in the first select list and are not redundant.
- MERGE.LIST generates a list containing any one of these formats (difference, intersection, or union).

There are a number of uses for these commands. For example, you might create two select lists and save them. One select list contains the record IDs of all inventory items with a QOH less than 100. The other select list represents those inventory items with a price greater than $75.

```
>SELECT INVENTORY.F WITH QOH LT 100

12 record(s) selected to SELECT list #0.
>>SAVE.LIST QOHL

12 record(s) SAVEd to SELECT list "QOHL".
>SELECT INVENTORY.F WITH PRICE GT 75

17 record(s) selected to SELECT list #0.
>>SAVE.LIST PRICEL

17 record(s) SAVEd to SELECT list "PRICEL".
```

Perhaps after using each of these saved lists separately in subsequent queries, you then decide you want to see a list of those items that appear in both lists. To do this, use the LIST.INTER command, specifying the first of the two lists to be compared. You are asked for the name of the second list, and then for the name of the new list that results from the comparison:

```
>LIST.INTER QOHL
WITH: PRICE
TO: MERGED

2 record(s) SAVEd to SELECT list "MERGED".
```

Now you can activate this new list, and use it:

```
>GET.LIST MERGED

2 record(s) selected to SELECT list #0.
>>LIST INVENTORY.F DESCRIPTION QOH PRICE BY DESCRIPTION ID.SUP

LIST INVENTORY.F DESCRIPTION QOH PRICE BY DESCRIPTION ID.SUP 10:32:42AM 31
May 1995 PAGE 1
DESCRIPTION..............  QOH..  PRICE.....

Bird Seed                        94   $101.03
Egg Rolls                        82   $159.85

2 records listed.
```

As another example, you could use the same two select lists to find out which inventory items have a QOH of less than 100 but do not have a price greater than $75. In this case, you use `LIST.DIFF`, which produces a select list of the elements in the first list (QOH > 100) that are not in the second list (PRICE > $75):

```
>LIST.DIFF QOHL
WITH: PRICE
TO: DIFFERL

10 record(s) SAVEd to SELECT list "DIFFERL".
>GET.LIST DIFFERL

10 record(s) selected to SELECT list #0.
>>LIST INVENTORY.F DESCRIPTION QOH PRICE BY DESCRIPTION ID.SUP

LIST INVENTORY.F DESCRIPTION QOH PRICE BY DESCRIPTION ID.SUP 09:14:06AM 31
May 1995  PAGE    1
DESCRIPTION.............    QOH..    PRICE.....

Balloons                     77        $64.40
Candy Selection              94        $71.28
Crabcakes                    87        $33.95
French Fries, Frozen         51        $45.78
Imported Ale                 83        $20.13
Ketchup                      71        $48.76
Onion Rings                  61        $25.96
Popcorn                      57        $50.54
Taffy                        71        $14.90
Ticket Stock                 96        $73.13

10 records listed.
```

`LIST.UNION` would generate a new select list that contains nonredundant elements from both lists, for example, inventory items that have a QOH of less than 100, or a price of more than $75, or both.

```
>LIST.UNION QOHL
WITH: PRICE
TO: UNIONL

27 record(s) SAVEd to SELECT list "UNIONL".
>GET.LIST UNIONL

27 record(s) selected to SELECT list #0.
>>LIST INVENTORY.F DESCRIPTION QOH PRICE BY DESCRIPTION ID.SUP

LIST INVENTORY.F DESCRIPTION QOH PRICE BY DESCRIPTION ID.SUP 10:34:49AM 31
May 1995  PAGE    1
DESCRIPTION.............    QOH..    PRICE.....

Balloons                      77       $64.40
Beer                         127      $116.92
Bird Seed                     94      $101.03
Candy Selection               94       $71.28
Cola                         185      $149.10
Cookies                      174      $143.55
Cotton Candy                 102       $75.83
Crabcakes                     87       $33.95
Domestic Cat Chow            100       $94.67
.
.
.
Ticket Stock                  96       $73.13

27 records listed.
```

Other things you can so with saved select lists

You can also edit, copy, and delete saved select lists. For more information about select lists, see the following commands in the *UniVerse User Reference*:

- CLEARSELECT
- COPY.LIST
- DELETE.LIST
- EDIT.LIST
- GET.LIST
- SAVE.LIST

Chapter 5: Redirecting output

All the queries presented thus far have directed their output to a terminal screen or printer. You can also direct query output to another file or to a tape. To do this, use the RetrieVe commands `REFORMAT`, `SREFORMAT`, and `T.DUMP`.

As one example of redirecting output, you might want to search the INVENTORY.F file for all records with a cost greater than $70 and write those records to a new file called INV.HICOST.

Redirecting output to a file

Query input can be directed to another file, with the new file containing one or more fields from the original file, possibly fields from other files (using the `TRANS` function), and virtual fields. Instead of just displaying these results, you can save them in a new file.

To write a query's output to a new file, use the RetrieVe `REFORMAT` and `SREFORMAT` commands. These commands are similar to the `LIST` and `SORT` commands and have an almost identical syntax, except that they direct their output to another file or to a tape instead of to a terminal screen or a printer. `REFORMAT` processes your data, as does the `LIST` command, before writing the data to another file or to tape. Any correlatives, conversions, and I-descriptors defined in the file dictionary are applied to the records before they are stored.

The new file can have the same record ID field as the original file, or it can be different.

Using the record IDs of the original file

Sometimes when you redirect a file, you want the new file to have the same record IDs as the original file. An INV.HICOST.F file, containing a copy of INVENTORY.F records for inventory items having a cost of over $70, is an example of this, and will be used in describing how to create and populate such a file.

Creating the new file

The first thing to do is to create the new file using the `CREATE.FILE` command. The command for creating the INV.HICOST.F file is as follows:

```
>CREATE.FILE
INV.HICOST.F 30Creating file "INV.HICOST.F" as Type 30.
Creating file "D_INV.HICOST.F" as Type 3, Modulo 1, Separation 2.
Added "@ID", the default record for RetrieVe, to "D_INV.HICOST.F".
```

Using REFORMAT to populate the new file

Next you issue a `REFORMAT` command to describe the selection criteria to be used in selecting the records and to name the fields you want included in the new file. Other than the record ID, if the only other fields of interest are DESCRIPTION, COST, and PRICE, then enter:

```
>REFORMAT INVENTORY.F @ID DESCRIPTION COST PRICE
    WITH COST > $70.00

File name =
```

Note that the first field you name (in this case, @ID) in the command becomes the record ID of the new file, and that the order in which you list the other fields determines the order of the fields in the new file. At the File name = prompt, enter the name of the new file:

```
File name = INV.HICOST.F
```

Now that the new file has been created and populated, you can list it. Use `LIST.ITEM` or `SORT.ITEM` instead of `LIST` or `SORT`, because the only entry in the file dictionary at this point is @ID and using `LIST` or `SORT` would display only that one field. For INV.HICOST.F, the command is:

```
>SORT.ITEM INV.HICOST.F

SORT.ITEM INV.HICOST.F 08:50:46AM 31 May 1995   PAGE  1

    1
001 Beer
002 $76.92
003 $116.92

    10
001 Franks
002 $99.92
003 $110.91

    12
001 Mustard
002 $91.52
003 $135.45

    14
001 Ice Cream, Various
002 $80.78
003 $99.36

.
.
.
```

Because the `CREATE.FILE` command creates a default file dictionary that has only one field definition (@ID), the field names, formatting, or conversion codes from the old file dictionary do not exist in this new file dictionary. Therefore, at some point, you would have to edit the file dictionary for INV.HICOST.F, adding names, formats, conversions, and so forth, for all the fields, as well as adding an @ phrase, I-descriptors, and other embellishments (refer to Querying a reformatted file, on page 110).

Using different record IDs in the new file

In other cases, the new file will have a different record ID field, and the fields may be in a different order than those of the original files. For example, you could take COMPANY, VENDOR.CODE, and TERMS from VENDORS.F, and write them to a new file called VEND.CO.F that has COMPANY instead of VENDOR.CODE as its record ID.

Creating the new file

As before, you must create the new file with the CREATE.FILE command. To create the VEND.CO.F file, enter:

```
>CREATE.FILE
VEND.CO.F 30Creating file "VEND.CO.F" as Type 30.
Creating file "D_VEND.CO.F.F" as Type 3, Modulo 1, Separation 2.
Added "@ID", the default record for RetrieVe, to "D_VEND.CO".
```

Using REFORMAT to populate the new file

Populating the new file is accomplished with the REFORMAT command, but this time a field other than @ID is specified as the record ID for the new file. For VEND.CO.F, this field is COMPANY and the command is:

```
>REFORMAT VENDORS.F COMPANY VENDOR.CODE TERMS

File name =
```

At the File name = prompt, enter the name of the new file, VEND.CO.F:

```
File name = VEND.CO.F
```

Now list the new file to make sure it contains what you expected. In the case of VEND.CO.F, record ID is company name, and vendor code and terms are its two data fields, as shown:

```
>SORT.ITEM VEND.CO.F

SORT.ITEM VEND.CO.F 08:56:14AM 31 May 1995     PAGE  1

    Accurate Surplus
001 3
002 Net 20

    Acme Brothers
001 1
002 Net 30

    Action Operators
001 2
002 Net 25

    Adam Supplies
001 216
002 Net 20

    Advantage Selections
001 5
002 Net 40

    Affordable Merchandise
001 4
002 Net 20
.
.
.
```

Reformatting from two or more file sources

Suppose that you want to create a file that contains data from more than one file source. For example, assume that you want a file containing the location codes, contact names, and contact phone numbers for only those sites where you have booked engagements. This involves two files, ENGAGEMENTS.F and LOCATIONS.F.

Creating the new file

Once again, you must issue a `CREATE.FILE` command to create a new file to hold the results. To create the PHONES.F file in this example, enter:

```
>CREATE.FILE PHONES.F 30
Creating file "PHONES.F" as Type 30.
Creating file "D_PHONES.F" as Type 3, Modulo 1, Separation 2.
Added "@ID", the default record for RetrieVe, to "D_PHONES.F".
```

Using REFORMAT to populate the new file

To populate the new file, enter a `REFORMAT` command. Remember that each field in the remote file from which you want to translate data requires a separate EVAL expression. The `REFORMAT` command to populate PHONES.F is:

```
>REFORMAT ENGAGEMENTS.F LOCATION.CODE BY LOCATION.CODE
    EVAL "TRANS(LOCATIONS.F, LOCATION.CODE, NAME,'X')"
    EVAL "TRANS(LOCATIONS.F, LOCATION.CODE,PHONE,'X')"
TRANS ( LOCATIONS.F , ( FIELD ( @ID , * , 1 ) ) , NAME , X )
TRANS ( LOCATIONS.F , ( FIELD ( @ID , * , 1 ) ) , PHONE , X )

 File name =
```

Here, the `TRANS` function takes the location codes from the ENGAGEMENTS.F file, finds the matching records in the LOCATIONS.F file, and for each record writes the LOCATION.CODE value from the ENGAGEMENTS.F file and the NAME and PHONE values from the LOCATIONS.F file into the new file. However, since LOCATION.CODE is the record ID of the new file and all its values must be unique, records with duplicate LOCATION.CODE values overwrite one another, so that the result is only one record per site.

At the following prompt, enter the name of the new file:

```
File name = PHONES.F
```

Now use `LIST.ITEM` or `SORT.ITEM` to list the new file. For PHONES.F, the command is:

```
>SORT.ITEM PHONES.F

SORT.ITEM PHONES.F 09:01:44AM 31 May 1995  PAGE    1

        CCLE001
001     Cleveland Properties, Inc.
002     216/965-8787

        CDET001
001     Detroit Properties, Inc.
002     313/774-4808

        CDFW001
001     Dallas Properties, Inc.
```

```
002 214/869-3105
...
```

Reformatting raw data

When you run REFORMAT or SREFORMAT, any conversions, I-descriptors, or correlatives defined in the file dictionary are applied before the file is reformatted. But if you want to redirect raw data without applying conversions, I-descriptors, and correlatives, you can do so by creating synonyms for the field definitions (omitting the conversions, I-descriptors, and correlatives) and then using those synonyms in the REFORMAT command line.

If you want to do this only for reformatting purposes and do not need to store a synonym in the file dictionary, you can define them only for the duration of the REFORMAT command. In such cases, you suppress unwanted conversions by using CONV " ". For example, to suppress conversions defined for COST in LIVESTOCK.F when reformatting the data, enter:

```
>REFORMAT LIVESTOCK.F …COST CONV ""…
```

Be careful here, because any field for which you do not provide a synonym or CONV " " uses the dictionary definition by default.

Querying a reformatted file

As mentioned earlier, the reformatted file initially has only one field defined for it in its file dictionary: the record ID (@ID). If you want to query the reformatted file using field names, as you are accustomed to doing, you must create entries in the file dictionary to define the new file's data structure. Otherwise, you must refer to the fields by their generic names, F1, F2, F3, and so forth (the record ID is still referred to as @ID). For example, to obtain a standard horizontal listing of the PHONES.F file, enter:

```
>SORT PHONES.F F1
F2
SORT PHONES.F F1 F2 09:08:19AM 31 May 1995  PAGE       1
 PHONES.F..   F1.............            F2...........

CCLE001         Cleveland           216/965-8787
                Properties,
                Inc.
CDET001         Detroit                 313/774-4808
                Properties,
                Inc.
CDFW001         Dallas                  214/869-3105
                Properties,
                Inc.
.
.
.
34 records listed.
```

The @ID entry is automatically listed, by default, and the output is sorted by @ID because of the SORT verb. The only other field in the file is the phone number field, and you refer to it as F1.

As another example, the INV.HICOST.F file created under Creating the new file, on page 106 contains four fields: the item code, description, cost, and price. Again, only the record ID (item code) is defined in the file dictionary, so you would have to refer to the other three fields as F1, F2, and F3 to list them.

To use meaningful field names, you have to recreate the applicable field definitions of the original file dictionary in the new file dictionary.

If the new file is simply a subset of records from the original file (as would be the case where INV.HICOST.F contained the entire record, not just the four fields), you could use the UniVerse COPY command to copy all the definitions from the INVENTORY.F dictionary to the INV.HICOST.F dictionary:

```
>COPY FROM DICT INVENTORY.F TO DICT INV.HICOST.F
```

On the other hand, if you specified that only certain fields are to be reformatted to the new file, or if you rearranged the fields, you could still copy the old dictionary to the new dictionary. However, you then need to edit the new dictionary manually to adjust field 2 of the data descriptor entries so they properly reflect the new locations or relative positions of the fields. Use ReVise or another appropriate editing tool such as the UniVerse Editor (or vi on UNIX systems) to do this.

Redirecting output to tape

There are two ways to redirect output to tape. If you are transferring all records in a file to tape, use the T.DUMP command. If you want to send only certain fields or rearrange the fields before sending the output to tape, use the REFORMAT or SREFORMAT command with an MTU option.

To use T.DUMP, do the following:

1. Attach the tape by using the ASSIGN command or the T.ATT command (a BASIC program which actually issues an ASSIGN command):

   ```
   >T.ATT MTU 001
   ```
2. Position the tape using T.FWD or some other tape positioning command.
3. Issue the T.DUMP command. For example, to copy records from ENGAGEMENTS.F for dates after 1995 to Tape 1 in Pick tape format, enter:

   ```
   >T.DUMP ENGAGEMENTS.F WITH DATE > '12/31/95' MTU 001 PICK.FORMAT
   ```
4. After the copying is completed, release the tape drive from your exclusive control by issuing an UNASSIGN command:

   ```
   >UNASSIGN MTU 001
   ```

   Alternately, to copy only the date, location, time, and advance fields for those same records from ENGAGEMENTS.F to tape, you would attach a tape and then use a REFORMAT (or SREFORMAT) command with an MTU keyword and enter TAPE as the filename:

   ```
   >REFORMAT ENGAGEMENTS.F WITH DATE > '12/31/95' DATE LOCATION
       TIME ADVANCE DATE LOCATION TIME ADVANCE MTU 001 PICK.FORMAT

   File Name = TAPE
   ```

   Once you have copied records to tape, you can just as easily load it back from tape to disk by using the T.LOAD command.

Loading T.DUMP files from tape to disk

The T.LOAD command copies files created by T.DUMP from tape to disk. To load the post-1995 engagement records created in the previous example back to disk, do the following:

1. Attach the tape unit on which the tape has been properly positioned:

   ```
   >T.ATT MTU 001
   ```
2. If necessary, create the file into which the tape's contents are to be loaded:

   ```
   >CREATE.FILE INV.HICOST1.F 30
   ```
3. Issue a T.LOAD command to load records from tape to the file:

```
>T.LOAD INV.HICOST1.F MTU 001 PICK.FORMAT
```

4. After the loading is completed, release the tape drive:

   ```
   >UNASSIGN MTU 001
   ```

   You can use a selection expression with `T.LOAD`, but you cannot specify sorting. If the file into which you are loading contains data, `T.LOAD` loads only those records that do not already exist. To cause `T.LOAD` to overwrite all records, use the keyword OVERWRITING.

Chapter 6: Creating an XML document with RetrieVe

This chapter discusses how to create an XML document with RetrieVe.

XML for UniVerse

The Extensible Markup Language (XML) is a markup language used to define, validate, and share document formats. It enables you to tailor document formats to specifications unique to your application by defining your own elements, tags, and attributes.

Note: XML describes how a document is structured, not how a document is displayed.

XML was developed by the World Wide Web Consortium (W3C), who describe XML as:

The Extensible Markup Language (XML) is the universal format for structured documents and data on the Web.

XML documents are text documents, intended to be processed by an application, such as a web browser.

An XML document consists of a set of tags that describe the structure of data. Unlike HTML, you can write your own tags. You can use XML to describe any type of data so that it is cross-platform and machine independent.

For detailed information about XML, see the W3C Website at http://www.w3.org/TR/REC-xml.

UniVerse enables you to receive and create XML documents, and process them through UniVerse BASIC, UniVerse SQL, or RetrieVe. In order to work with the XML documents in UniVerse, you will need to know some key terms:

- Document Type Definitions
- XML Schema
- Document Object Model
- Well-Formed and Valid Documents

Document type definitions

You must define the rules of the structure of your XML document. These rules may be part of the XML document, and are called the Document Type Definition, or DTD. The DTD provides a list of elements, tags, attributes, and entities contained in the document, and describes their relationship to each other.

A DTD can be external or internal.

- External DTD — An external DTD is a separate document from the XML document, residing outside of your XML document. External DTDs can be applied to many different XML documents. If you need to change the DTD, you can make the change once, and all referencing XML documents are updated automatically.
- Internal DTD — An internal DTD resides in the XML document as part of the header of the document, and applies only to that XML document.

You can combine external DTDs with internal DTDs in an XML document, and you can create DTDs in an XML document.

XML schema

The structure of the XML document can also be defined using XMLSchema, which is an XML-based alternative to the DTD. An XML Schema defines a class of XML documents, including the structure, content and meaning of the XML document. XML Schema is useful because it is written in XML and is extensible to future additions. You can create schema with XML, and you can use schema to validate XML. The XML Schema language can also be referred to as XML Schema Definition (XSD).

The Document Object Model (DOM)

The Document Object Model (DOM) is a platform- and language-independent interface that enables programs and scripts to dynamically access and update the content, structure, and style of documents. A DOM is a formal way to describe an XML document to another application or programming language. You can describe the XML document as a tree, with nodes representing elements, attributes, entities, an text.

Well-formed and valid XML documents

An XML document is either well-formed or valid:

- Well-formed XML documents must follow XML rules. All XML documents must be well-formed.
- Valid XML documents are both well-formed, and follow the rules of a specific DTD or schema. Not all XML documents must be valid.

For optimum exchange of data, you should try to ensure that your XML documents are valid.

The U2XMLOUT.map file is located in the $UVHOME directory. The U2XMLOUT.map file sets the flag for all UniVerse users.

Creating an XML document from RetrieVe

You can create an XML document from UniVerse files through RetrieVe. To create an XML document through RetrieVe, complete the following steps:

1. If you are the originator of the DTD or XML Schema, use RetrieVe to create the DTD or XMLSchema. If you are not the originator of the DTD or XML Schema, analyze the DTD or XML Schema associated with the application to which you are sending the XML file. Determine which of your dictionary attributes correspond to the DTD or XML Schema elements.

   You can also refer to Mapping to an external schema, on page 137.
2. Create an XML mapping file, if necessary. The mapping file will enable users to create many different forms of XML.
3. List the appropriate fields using the `LIST` command.

Create the &XML& file

UniVerse stores XML mapping files in the &XML& directory file. This directory is automatically created with new accounts. If you have an older account, create this file using the following command:

```
CREATE.FILE DIR &XML&
```

Mapping modes

UniVerse supports three modes for mapping data to XML files. These modes are:

- Attribute-centric
- Element-centric
- Mixed

Attribute-centric mode

In the attribute-centric mode, which is the default mode, each record displayed in the query statement becomes an XML element. The following rules apply to the record fields:

- Each singlevalued field becomes an attribute within the element.
- Each multivalued field becomes a sub-element of the record element. The name of the sub-element, if there is no association, is *fieldname_MV.*
- Within a sub-element, each multivalued field becomes an attribute of the sub-element.

This is the default mapping scheme. You can change the default by defining maps in the &XML& directory.

The following example shows data created in attribute mode:

```
>LIST STUDENT LNAME CGA TOXML SAMPLE 1<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
<STUDENT _ID = "987654321" LNAME = "Miller">
    <CGA-MV SEMESTER = "FA93" COURSE_NBR = "EG110" COURSE_NAME =
"Engineering Prin
ciples" COURSE_GRD = "C" COURSE_HOURS = "5" TEACHER = "Carnes"
COURSE_NBR = "MA2
20" COURSE_NAME = "Calculus- I" COURSE_HOURS = "5" TEACHER =
"Otis" COURSE_NBR =
 "PY100" COURSE_NAME = "Introduction to Psychology" COURSE_GRD =
"B" COURSE_HOUR
S = "3" TEACHER = "Masters"/>
    <CGA-MV SEMESTER = "SP94" COURSE_NBR = "EG140" COURSE_NAME =
"Fluid Mechanics"
 COURSE_GRD = "B" COURSE_HOURS = "3" TEACHER = "Aaron" COURSE_NBR
= "EG240" COUR
SE_NAME = "Circut Theory" COURSE_GRD = "B" COURSE_HOURS = "3"
TEACHER = "Carnes"
    COURSE_NBR = "MA221" COURSE_NAME = "Calculus - II" COURSE_HOURS =
"5" TEACHER =
    "Otis"/>
    </STUDENT>
    </ROOT>
    >
```

Element-centric mode

In the element-centric mode, as in the attribute-centric mode, each record becomes an XML element. The following rules apply:

- Each singlevalued field becomes a simple sub-element of the element, containing no nested sub-elements. The value of the field becomes the value of the sub-element.
- Each association whose multivalued fields are included in the query statement form a complex sub-element. In the sub-element, each multivalued field belonging to the association becomes a sub-element. There are two ways to display empty values in multivalued fields belonging to an association. For detailed information, see Displaying empty values in multivalued fields in an association, on page 117.
- By default, UniVerse converts text marks to an empty string.

Specify that you want to use element-centric mapping by using the ELEMENTS keyword in the RetrieVe statement. You can also define treated-as = “ELEMENT” in the U2XMLOUT.map file, so that all XML will be created in element mode.

The following example shows data created in element mode:

```
:LIST STUDENT LNAME CGA TOXML ELEMENTS SAMPLE 1
<?xml version="1.0"?>
<ROOT>
<STUDENT>
    <_ID>521814564</_ID>
    <LNAME>Smith</LNAME>
        <CGA-MV>
        <SEMESTER>FA93</SEMESTER>
        <CGA-MS>
            <COURSE_NBR>CS130</COURSE_NBR>
            <COURSE_NAME>Intro to Operating Systems</COURSE_NAME>
            <COURSE_GRD>A</COURSE_GRD>
            <COURSE_HOURS>5</COURSE_HOURS>
            <TEACHER>James</TEACHER>
        </CGA-MS>
        <CGA-MS>
            <COURSE_NBR>CS100</COURSE_NBR>
            <COURSE_NAME>Intro to Computer Science</COURSE_NAME>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_HOURS>3</COURSE_HOURS>
            <TEACHER>Gibson</TEACHER>
        </CGA-MS>
        <CGA-MS>
            <COURSE_NBR>PY10</COURSE_NBR>
            <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_HOURS>3</COURSE_HOURS>
            <TEACHER>Masters</TEACHER>
        </CGA-MS>
    </CGA-MV>
    <CGA-MV>
        <SEMESTER>SP94</SEMESTER>
        <CGA-MS>
            <COURSE_NBR>CS13</COURSE_NBR>
            <COURSE_NAME>Intro to Operating Systems</COURSE_NAME>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_HOURS>5</COURSE_HOURS>
            <TEACHER>Aaron</TEACHER>
        </CGA-MS>
        <CGA-MS>
            <COURSE_NBR>CS101</COURSE_NBR>
```

```
            <COURSE_NAME>Intro to Computer Science</COURSE_NAME>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_HOURS>4</COURSE_HOURS>
            <TEACHER>Gibson</TEACHER>
        </CGA-MS>
        <CGA-MS>
            <COURSE_NBR>PE220</COURSE_NBR>
            <COURSE_NAME>Racquetball</COURSE_NAME>
            <COURSE_GRD>A</COURSE_GRD>
            <COURSE_HOURS>3</COURSE_HOURS>
            <TEACHER>Fisher</TEACHER>
        </CGA-MS>
    </CGA-MV>
</STUDENT>
</ROOT>
```

Displaying empty values in multivalued fields in an association

UniVerse displays empty values in multivalued fields belonging to an association depending on the setting of the Matchelement field in the U2XMLOUT.map file.

If Matchelement is set to 1 (the default), matching values or subvalues belonging to the same association display as empty elements for matching pairs.

Consider the following example:

```
>LIST STUDENT LNAME FNAME COURSE_NBR COURSE_GRD COURSE_NAME
SEMESTER 10:50:27am

11 Sep 2007  PAGE    1

STUDENT.... 987654321
Last Name.. Miller
First Name. Susan
Crs #..... GD. Course Name.... Term
EG110      C   Engineering     FA93
               Principles
MA220          Calculus- I
PY100      B   Introduction to
               Psychology
EG140      B   Fluid Mechanics SP94
EG240      B   Circut Theory
MA221          Calculus - II

STUDENT.... 123456789
Last Name.. Martin
First Name. Sally
Crs #..... GD. Course Name.... Term
PY100          Introduction to SP94
               Psychology
PE100      C   Golf - I


2 records listed.
>
```

Notice that three of the GRADE fields are empty, while their associated values for COURSE # and COURSE NAME are not.

When Matchelement is set to 1, the missing values for COURSE _GRD, <COURSE_GRD/>, display as an empty value in the XML document, as shown in the following example:

```
>LIST STUDENT CGA TOXML XMLMAPPING student.map
<?xml version="1.0" encoding="UTF-8"?>
<main>
<STUDENT _ID = "987654321">
    <Term COURSE_HOURS = "5" COURSE_HOURS = "5" COURSE_HOURS = "3">
        <SEMESTER>FA93</SEMESTER>
        <Courses_Taken>
            <COURSE_NAME>Engineering Principles</COURSE_NAME>
            <COURSE_GRD>C</COURSE_GRD>
            <COURSE_NBR>EG110</COURSE_NBR>
            <TEACHER>Carnes</TEACHER>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_NAME>Calculus- I</COURSE_NAME>
            <COURSE_GRD/>
            <COURSE_NBR>MA220</COURSE_NBR>
            <TEACHER>Otis</TEACHER>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_NBR>PY100</COURSE_NBR>
            <TEACHER>Masters</TEACHER>
        </Courses_Taken>
    </Term>
    <Term COURSE_HOURS = "3" COURSE_HOURS = "3" COURSE_HOURS = "5">
        <SEMESTER>SP94</SEMESTER>
        <Courses_Taken>
            <COURSE_NAME>Fluid Mechanics</COURSE_NAME>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_NBR>EG140</COURSE_NBR>
            <TEACHER>Aaron</TEACHER>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_NAME>Circut Theory</COURSE_NAME>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_NBR>EG240</COURSE_NBR>
            <TEACHER>Carnes</TEACHER>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_NAME>Calculus - II</COURSE_NAME>
            <COURSE_GRD/>
            <COURSE_NBR>MA221</COURSE_NBR>
            <TEACHER>Otis</TEACHER>
        </Courses_Taken>
    </Term>
</STUDENT><STUDENT _ID = "123456789">
    <Term COURSE_HOURS = "3" COURSE_HOURS = "3">
        <SEMESTER>SP94</SEMESTER>
        <Courses_Taken>
            <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
            <COURSE_GRD/>
            <COURSE_NBR>PY100</COURSE_NBR>
            <TEACHER>Masters</TEACHER>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_NAME>Golf - I</COURSE_NAME>
            <COURSE_GRD>C</COURSE_GRD>
            <COURSE_NBR>PE100</COURSE_NBR>
```

```
            <TEACHER>Fisher</TEACHER>
        </Courses_Taken>
    </Term>
</STUDENT>
</main>
>
```

This is the default behavior.

When Matchelement is set to 0, the missing value for COURSE_GRD, <COURSE_GRD/>, is ignored in the XML document, as shown in the following example:

```
>LIST STUDENT CGA TOXML XMLMAPPING student.map
<?xml version="1.0" encoding="UTF-8"?>
<main>
<STUDENT _ID = "987654321">
    <Term COURSE_HOURS = "5" COURSE_HOURS = "5" COURSE_HOURS = "3">
        <SEMESTER>FA93</SEMESTER>
        <Courses_Taken>
            <COURSE_NAME>Engineering Principles
            </COURSE_NAME>
             <COURSE_GRD>C</COURSE_GRD>
            <COURSE_NBR>EG110</COURSE_NBR>
             <TEACHER>Carnes</TEACHER>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_NAME>Calculus- I</COURSE_NAME>
            <COURSE_NBR>MA220</COURSE_NBR>
            <TEACHER>Otis</TEACHER>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_NBR>PY100</COURSE_NBR>
            <TEACHER>Masters</TEACHER>
        </Courses_Taken>
    </Term>
    <Term COURSE_HOURS = "3" COURSE_HOURS = "3" COURSE_HOURS = "5">
        <SEMESTER>SP94</SEMESTER>
        <Courses_Taken>
            <COURSE_NAME>Fluid Mechanics</COURSE_NAME>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_NBR>EG140</COURSE_NBR>
            <TEACHER>Aaron</TEACHER>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_NAME>Circut Theory</COURSE_NAME>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_NBR>EG240</COURSE_NBR>
            <TEACHER>Carnes</TEACHER>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_NAME>Calculus - II</COURSE_NAME>
            <COURSE_GRD/>
            <COURSE_NBR>MA221</COURSE_NBR>
            <TEACHER>Otis</TEACHER>
        </Courses_Taken>
    </Term>
</STUDENT>
<STUDENT _ID = "123456789">
    <Term COURSE_HOURS = "3" COURSE_HOURS = "3">
        <SEMESTER>SP94</SEMESTER>
```

```
        <Courses_Taken>
            <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
            <COURSE_NBR>PY100</COURSE_NBR>
            <TEACHER>Masters</TEACHER>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_NAME>Golf - I</COURSE_NAME>
            <COURSE_GRD>C</COURSE_GRD>
            <COURSE_NBR>PE100</COURSE_NBR>
            <TEACHER>Fisher</TEACHER>
        </Courses_Taken>
    </Term>
</STUDENT>
</main>
>
```

Mixed mode

In the mixed-mode, you create your own map file, where you specify which fields are treated as attribute-centric and which fields are treated as element-centric.

Field-level mapping overrides the mode you specify in the RetrieVe statement.

The mapping file

You can create the U2XMLOUT.map file in $UVUVHOME to define commonly used global settings for creating XML documents. UniVerse reads and processes this mapping file each time UniVerse is started. For example, if you normally create element-centric output, and display empty elements for missing values or subvalues belonging to the same association, you can define these settings in the U2XMLOUT.map file, as shown in the following example:

```
<U2
    matchelement = "1"
    treated-as = "element"
/>
```

Defining these settings in the mapping file eliminates the need to specify them in each RetrieVe statement.

UniVerse processes XML options as follows:

1. Reads options defined in the U2XMLOUT.map file when UniVerse starts.
2. Reads any options defined in a mapping file. This mapping file resides in the &XML& directory in the current account, and is specified in the RetrieVe statement, as shown in the following example:

   ```
   LIST STUDENT SEMESTER TOXML XMLMAPPING mystudent.map
   ```
3. Processes any options you specify in the RetrieVe statement.

Options you specify in the RetrieVe statement override options defined in the mapping file. Options defined in the mapping file override options defined in the U2XMLOUT.map file.

A mapping file has the following format:

```
<?XML version="1.0"?>
<!--there can be multiple <U2xml:mapping> elements -->
    <U2xml:mapping file="file_name"
```

```
                    hidemv="0"
                    hidems="0"
                    hideroot="0"
                    collapsemv="0"
                    collapsems="0"
                    emptyattribute="0"
                    hastm="yes" | "1"
                    matchelement="0" |"1"
                    schematype="ref"
                    targetnamespace="targetURL"
                    xmlns:NAME="URL"
                    field="dictionary_display_name"
                    map-to="name_in_xml_douniversec"
                    type="MV" | "MS"
                    treated-as="attribute" | "element"
                    root="root_element_name"
                    record="record_element_name"
                    association-mv="mv_level_assoc_name"
                    association-ms="ms_level_assoc_name"
                    format (or Fmt)= "format -pattern" ..
                    conversion (or Conv)= "conversion code"
                    encode="encoding characters"
                    />
...
</U2xml-mapping>
```

The XML mapping file is, in itself, in XML format. There are three types of significant elements: the root element, the field element, and the association element.

- The root element – The root element describes the global options that control different output formats, such as the schema type, targetNamespace, hideroot, hidemv, and hidems. You can also use the root element to change the default root element name, or the record element name. You should have only one root element in the mapping file.
- The field element – UniVerse uses the field element to change the characteristics of a particular field's XML output attributes, such as the display name, the format, or the conversion.
- The association element – UniVerse uses the association element to change the display name of an association. By default, this name is the association phrase name, together with "-MV."

Distinguishing elements

You can distinguish the root element from the field and association elements because the root element does not define a field or association element.

Both the field element and the association element must have the file and field attribute to define the file name and the field name in the file that has been processed. Generally, the field name is a data-descriptor or I-descriptor defined in the dict file, making it a field element. If the field name is an association phrase, it is an association element.

The Mapping file example, on page 127 shows this in more detail.

Root element attributes

The default root element name in an XML document is ROOT. You can change the name of the root element, as shown in the following example:

```
root="root-element-name"
```

Record name attribute

The default record name is FILENAME_record. The record attribute in the root element changes the record name. The following example illustrates the record attribute:

```
record="record-element-name"
```

Hideroot attribute

The hideroot attribute allows you to specify whether to create the entire XML document or only a section of it, for example, using the SAMPLE keyword or other conditional clauses.

If hideroot is set to 1, UniVerse only creates the record portion of the XML document, it does not create a DTD or XMLSchema. The default value is 0.`Hideroot="1"/"0"`

Hidemv attribute

This attribute specifies whether to hide <MV> and </MV> tags for multivalued fields belonging to an association in the generated XML document and in the associated DTD or XML Schema. This parameter applies only if the XML document is created in element mode.

0 - Show MV tags for multivalued fields.

1 - HideMV tags for multivalued fields.

You can also use this option with XMLEXECUTE().

Note: If the document is created in attribute mode, it is not possible to eliminate the extra level of element tags.

Collapsemv attribute

This attribute specifies whether to collapse <MV> and </MV> tags, using only one set of these tags for multivalued fields belonging to an association in the generated XML document and in the associated DTD or XMLSchema. This parameter applies only if the XML document is created in element mode.

0 - Expand MV tags for multivalued fields.

1 - CollapseMV tags for multivalued fields.

Empty attribute

This attribute determines how to display the empty attributes for multivalued fields belonging to an association in the generated XML document and in the associated DTD or XML Schema. This option can be specified in the U2XMLOUT.map file, or in an individual mapping file.

0 - Hides the empty attributes in the multivalued fields.

1 - Shows the empty attributes in the multivalued fields.

Namespace attributes

UniVerse provides the following attributes for defining namespaces:

- xmlns:name-space-name=”URL”
- targetnamespace=”URL”

UniVerse displays the targetnamespace attribute in the XMLSchema as targetNamespace, and uses the URL you define in the XML document to define the schema location.

If you define the targetnamespace and other explicit namespace definitions, UniVerse checks if the explicitly defined namespace has the same URL as the targetnamespace. If it does, UniVerse uses the namespace name to qualify the schema element, and the XML document element name.

If there is no other namespace explicitly defined, UniVerse creates a defaultnamespace in the schema file as shown in the following example:

```
xmlns=”targetnamespace URL”
```

In this case, UniVerse does not qualify the schema element or the XML document element.

UniVerse uses the namespace attributes and xmlns:name-space-name together to define the namespace. All namespaces defined in the root element are for global element namespace qualifiers only.

Note: Namespace is used primarily for XMLSchema. If you do not specify XMLSchema in the command line, UniVerse will not use a global namespace to qualify any element in the document.

The following program illustrates the output of the TARGETNAMESPACE attribute:

```
>RUN BP XML4
XMLMAPPING=student.map ELEMENTS
TARGETNAMESPACE=www.rocketsoftware.com
Options XMLMAPPING=student.map ELEMENTS
TARGETNAMESPACE=www.rocketsoftware.com
XML output
<?xml version="1.0" encoding="UTF-8"?>
<main
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="www.rocketsoftware.com">
<STUDENT>
    <_ID>987654321</_ID>
    <LNAME>Miller</LNAME>
    <Term>
        <SEMESTER>FA93</SEMESTER>
        <Courses_Taken>
            <COURSE_GRD>C</COURSE_GRD>
            <COURSE_NAME>Engineering Principles</COURSE_NAME>
            <COURSE_NBR>EG110</COURSE_NBR>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_GRD/>
            <COURSE_NAME>Calculus- I</COURSE_NAME>
            <COURSE_NBR>MA220</COURSE_NBR>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
            <COURSE_NBR>PY100</COURSE_NBR>
        </Courses_Taken>
    </Term>
    <Term>
        <SEMESTER>SP94</SEMESTER>
        <Courses_Taken>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_NAME>Fluid Mechanics</COURSE_NAME>
```

```
            <COURSE_NBR>EG140</COURSE_NBR>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_GRD>B</COURSE_GRD>
            <COURSE_NAME>Circut Theory</COURSE_NAME>
            <COURSE_NBR>EG240</COURSE_NBR>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_GRD/>
            <COURSE_NAME>Calculus - II</COURSE_NAME>
            <COURSE_NBR>MA221</COURSE_NBR>
        </Courses_Taken>
    </Term>
    <FNAME>Susan</FNAME>
</STUDENT>
<STUDENT>
    <_ID>123456789</_ID>
    <LNAME>Martin</LNAME>
    <Term>
        <SEMESTER>SP94</SEMESTER>
        <Courses_Taken>
            <COURSE_GRD/>
            <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
            <COURSE_NBR>PY100</COURSE_NBR>
        </Courses_Taken>
        <Courses_Taken>
            <COURSE_GRD>C</COURSE_GRD>
            <COURSE_NAME>Golf - I</COURSE_NAME>
            <COURSE_NBR>PE100</COURSE_NBR>
        </Courses_Taken>
    </Term>
    <FNAME>Sally</FNAME>
</STUDENT>
</main>

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="www.rocketsoftware.com"
    xmlns:intf="www.rocketsoftware.com"
    xmlns="www.rocketsoftware.com"
    elementFormDefault="qualified">
    <xsd:annotation>
    <xsd:documentation xml:lang="en">
        account: C:\U2\uv\uvxml
        command: LIST STUDENT LNAME COURSE_NBR COURSE_GRD COURSE_NAME
SEMESTER FNAME
    TOXML XMLMAPPING student.map ELEMENTS WITHSCHEMA
        </xsd:documentation>
        </xsd:annotation>
        <xsd:element name="main">
            <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="intf:STUDENT" minOccurs="0"
maxOccurs="unbounded"/>
        </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
        <xsd:element name="STUDENT">
            <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="_ID" type="xsd:string"/>
                <xsd:element name="LNAME" type="xsd:string"/>
```

```
                <xsd:element ref="intf:Term" minOccurs="0"
maxOccurs="unbounded"/>
                <xsd:element name="FNAME" type="xsd:string"/>
            </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
            <xsd:element name="Term">
                <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="SEMESTER" type="xsd:string"/>
                <xsd:element ref="intf:Courses_Taken" minOccurs="0"
maxOccurs="unbound
ed"/>
            </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
            <xsd:element name="Courses_Taken">
                <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="COURSE_GRD" type="xsd:string"/>
                    <xsd:element name="COURSE_NAME" type="xsd:string"/>
                    <xsd:element name="COURSE_NBR" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
</xsd:element></xsd:schema>
```

Schema attribute

The default schema format is ref type schema. You can use the schema attribute to define a different schema format.

```
schema=”inline”|”ref”|”type”
```

Elementformdefault and Attributeformdefault attributes

UniVerse uses the elementformdefault and attributeformdefault attributes in the XML Schema. If you use them together with the namespace attribute in the root element, you can indicate all of the local elements and local attributes that need to be qualified with the namespace.

File attribute

UniVerse uses the file attribute to process both RetrieVe and UniVerse SQL commands. If you do not define the file attribute exactly as it is used on the command line, the field element will not be properly processed.

```
File=”filename”
```

Field attribute

The field attribute defines the field name. The field can be either a data-descriptor, an I-descriptor, or an ‘association phrase name’. For more information, see Association elements, on page 126.

```
Field=”field-name”
```

Note: The file and field attributes are used to identify the query file and field needed to change the default directions. Use these attributes in the same element of the XML mapping file to pinpoint the database file and field.

Map-to attribute

The map-to attribute allows you to define a new attribute tag or element tag name for the field. By default, UniVerse uses the dictionary display field name for the element or attribute name tag.

Type attribute

The type attribute defines how to treat the field in the XML document, either as a multivalued field or a multi-subvalued field.

```
type="MV"|"MS"
```

Treated-as attribute

The treated-as attribute determines if the field should be treated as an element or an attribute in the generated XML document.

Matchelement attribute

The matchelement attribute specifies whether to display empty elements for missing values or subvalues belonging to the same association, or to ignore the missing values.

Encode attribute

The encode attribute encodes unprintable characters, or characters that have special meanings in XML, such as { : }, with a macro.

```
encode="0x7B 0x7D"
```

Conv attribute

The conv attribute changes the conversion defined in the dictionary record to the conversion you define.

```
conv="new conv code" | conversion = "new conversion code"
```

Fmt attribute

The fmt attribute changes the format defined in the dictionary record to the format you define.

```
fmt="new format code" | format = "new format code"
```

Association elements

An association element contains the following four attributes:

- file = "file name"

- field = “association phrase name”
- association-mv = “new multivalue element tag”
- association-ms = “new multi-subvalue element tag”

Mapping file example

The following example illustrates a mapping file:

```
<!-- this is for STUDENT file -->
<U2
            root="main"
            collapsemv='1'
            collapsems='1'
            schema="ref"
            hidemv="1"
            hidems="1"
            hideroot="0"
            elementformdefault="qualified"
            attributeformdefault="qualified"
            treated-as="element"
/>
<U2 file="STUDENT"
                field = "CGA"
                association-mv="Term"
                association-ms="Courses_Taken"
/>
<U2 file="STUDENT"
                field = "COURSE_NBR"
                type="MS"
                treated-as="element"
/>
<U2 file="STUDENT"
                field = "SEMESTER"
                map-to="SEMESTER"
                type="MV"
                treated-as="element"
/>
<U2 file="STUDENT"
                field = "COURSE_GRD"
                map-to="COURSE_GRD"
                type="ms"
                treated-as="element"
/>
<U2 file="STUDENT"
                field = "COURSE_NAME"
                type="ms"
                treated-as="element"
/>
<U2 file="STUDENT"
                field = "TEACHER"
                type="ms"
                treated-as="element"
/>
```

Notice that the SEMESTER, COURSE_NBR, COURSE_GRD, and COURSE_NAME fields are to be treated as elements. When you create the XML document, these fields will produce element-centric XML data. Any other fields listed in the query statement will produce attribute-centric XML data, since attribute-centric is the default mode.

Additionally, COURSE_NBR, COURSE_GRD, and COURSE_NAME are defined as multi-subvalued fields. If they were not, UniVerse would create the XML data as if they were multivalued attributes.

Note: The global attributes listed above are not defined because they are set to “1”.

The next example illustrates an XMLSchema using the mapping file in the previous example.

Use the following command to create the `.xsd` schema:

LIST STUDENT LNAME SEMESTER COURSE_NBR TOXML XMLMAPPING student.map SCHEMAONLY

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
  <xsd:documentation xml:lang="en">
    account: C:\U2\uv\uvxml
    command: LIST STUDENT LNAME SEMESTER COURSE_NBR TOXML
XMLMAPPING SCHEMAONLY
  </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="main">
    <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="STUDENT" minOccurs="0"
maxOccurs="unbounded"
    </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
    <xsd:element name="STUDENT">
      <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Term" minOccurs="0"
maxOccurs="unbounded"/
      </xsd:sequence>
      <xsd:attribute name="_ID" type="xsd:string"/>
      <xsd:attribute name="LNAME" type="xsd:string"/>
      </xsd:complexType>
    </xsd:element>
      <xsd:element name="Term">
        <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="SEMESTER" minOccurs="0"
type="xsd:strin
          <xsd:element ref="Courses_Taken" minOccurs="0"
maxOccurs="

        </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
        <xsd:element name="Courses_Taken">
          <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="COURSE_NBR" minOccurs="0"
type="xsd:s
          </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
</xsd:schema>
>
```

The next example illustrates an XML document created using the mapping file in the previous example. Use the following command to display the XML to the screen:

```
<?xml version="1.0" encoding="UTF-8"?>
<main>
<STUDENT _ID = "987654321" LNAME = "Miller">
    <Term>
```

```
            <SEMESTER>FA93</SEMESTER>
            <Courses_Taken>
            <COURSE_NBR>EG110</COURSE_NBR>
            </Courses_Taken>
            <Courses_Taken>
                <COURSE_NBR>MA220</COURSE_NBR>
            </Courses_Taken>
            <Courses_Taken>
                <COURSE_NBR>PY100</COURSE_NBR>
            </Courses_Taken>
        </Term>
         <Term>
            <SEMESTER>SP94</SEMESTER>
            <Courses_Taken>
                <COURSE_NBR>EG140</COURSE_NBR>
            </Courses_Taken>
            <Courses_Taken>
                <COURSE_NBR>EG240</COURSE_NBR>
            </Courses_Taken>
            <Courses_Taken>
                <COURSE_NBR>MA221</COURSE_NBR>
            </Courses_Taken>
        </Term>
    </STUDENT>
    <STUDENT _ID = "123456789" LNAME = "Martin">
        <Term>
            <SEMESTER>SP94</SEMESTER>
            <Courses_Taken>
                <COURSE_NBR>PY100</COURSE_NBR>
            </Courses_Taken>
            <Courses_Taken>
                <COURSE_NBR>PE100</COURSE_NBR>
            </Courses_Taken>
        </Term>
    </STUDENT>
    </main>
    >
```

Conversion code considerations

UniVerse uses the following rules when extracting data from database files:

- If the dictionary record of a field you are extracting contains a conversion code, UniVerse uses that conversion code when extracting data from database files.
- If you specify a conversion code in the mapping file, the conversion code in the mapping file overrides the conversion code specified in the dictionary record.
- If you specify a conversion code using the CONV keyword during the execution of a RetrieVe statement, that conversion code overrides both the conversion code specified in the mapping file and the conversion code specified in the dictionary record.

Formatting considerations

UniVerse does not generally apply the dictionary format pattern to the extracted data. To specify a format, define it in the mapping file. If you specify a format using the FMT keyword in a RetrieVe statement, that format will override the format defined in the mapping file.

Mapping file encoding

For special characters encountered in data, UniVerse uses the default XML entities to encode the data. For example, '<' becomes <, '>' becomes >, '&' becomes &, and ' " ' becomes ". However, UniVerse does not convert ' to ', unless you specify it in attribute encode. (<, >, &, ', and " are all built-in entities for the XML parser).

Use the encode field in the mapping file to add flexibility to the output. You can define special characters to encode in hexadecimal form. UniVerse encodes these special characters to &#x##;. For example, if you want the character '{' to be encoded for field FIELD1, specify the following encode value in the mapping file for FIELD1:

```
encode="0x7B"
```

In this case, UniVerse will convert '{' found in the data of FIELD1 to &#x7B.

You can also use this type of encoding for any nonprintable character. If you need to define more than one character for a field, add a space between the hexadecimal definitions. For example, if you want to encode both '{' and '}', the encode value in the mapping file should look like the following example:

```
encode="0x7B 0x7D"
```

How data is mapped

Regardless of the mapping mode you choose, the outer-most element in the XML document is created as <ROOT>, by default. The name of each record element defaults to <*file_name*>.

You can change these mapping defaults in the mapping file, as shown in the following example:

```
<U2xml:mapping root="root_name"
                          record="record_name"/>
```

Mapping example

The following example illustrates the creation of XML documents. These examples use the STUDENT file, which contains the following fields:

```
>LIST DICT STUDENT

DICT STUDENT    11:39:32am  11 Sep 2007  Page    1
                Type &
Field......... Field. Field........ Conversion.. Column......... Output
Depth &
Name.......... Number Definition... Code........ Heading........ Format
Assoc..

@ID             D    0                          STUDENT          10L     S
ID              D    0                          STUDENT          12R###  S
                                                                 -##-##
                                                                 ##
LNAME           D    1                          Last Name        40T     S
FNAME           D    2                          First Name       10L     S
MAJOR           D    3                          Major            20L     S
MINOR           D    4                          Minor            4L      S
ADVISOR         D    5                          Advisor          8L      S
SEMESTER        D    6                          Term             4L      S
CGA
TESTSEME        D    6                          Term             4L      S
COURSE_NBR      D    7                          Crs #            10L     S
CGA
TESTCOURSE      D    7                          Crs #            5L      S
COURSE_GRD      D    8                          GD               3L      S
CGA
?¼
     ? ` ?
GPA1            I      SUBR('GPA1',C MD3        GPA              5R      S

                Type &
Field......... Field. Field........ Conversion.. Column......... Output
Depth &
Name.......... Number Definition... Code........ Heading........ Format
Assoc..

                       OURSE_HOURS,C
                       OURSE_GRD)
TEACHER         I      TRANS('COURSE            Teacher          10L     M
CGA
                       S',COURSE_NBR
                       ,'TEACHER','X
                       ')
COURSE_NAME     I      TRANS('COURSE            Course Name      15T     S
CGA
                       S',COURSE_NBR
                       ,'NAME','X')
COURSE_HOURS    I      TRANS('COURSE            Hours            5R      M
CGA
                       S',COURSE_NBR
                       ,CREDITS,'X')
@               PH      LNAME FNAME
                       MAJOR MINOR
                       ADVISOR

                       SEMESTER
                       COURSE_NBR
                Type &
Field......... Field. Field........ Conversion.. Column......... Output
Depth &
Name.......... Number Definition... Code........ Heading........ Format
Assoc..

                       COURSE_GRD
CGA             PH     SEMESTER
                       COURSE_NBR
                       COURSE_NAME
                       COURSE_GRD
                       COURSE_HOURS
                       TEACHER
@ORIGINAL       S  @ID                                                  M
@SYNONYM        S   ID                                                  M

22 records listed.
```

Creating an XML document

To create an XML document using RetrieVe, use the `LIST` command.

```
LIST [DICT | USING [DICT] dictname] filename ... [TOXML [ELEMENTS]
[WITHDTD] [WITHSCHEMA | SCHEMAONLY] [XMLMAPPING mapping_file] [TO
xmlfile]]
```

The following table describes each parameter of the syntax.

Parameter	Description
DICT	Lists records in the file dictionary of *filename*. If you do not specify DICT, records in the data file are listed.
USING [DICT] *dictname*	If DICT is not specified, uses the data portion of *dictname* as the dictionary of *filename*. If DICT is specified, the dictionary of *dictname* is used as the dictionary of *filename*.
filename	The file whose records you want to list. You can specify *filename* anywhere in the sentence. LIST uses the first word in the sentence that has a file descriptor in the VOC file as the file name.
TOXML	Outputs `LIST` results in XML format.
ELEMENTS	Outputs results in element-centric format.
WITHDTD	Output produces a DTD corresponding to the query.
WITHSCHEMA	The output produces an XML schema corresponding to the XML output.
SCHEMAONLY	The output produces a schema for the corresponding query.
XMLMAPPING *mapping_file*	Specifies a mapping file containing transformation rules for display. This file must exist in the &XML& file.
TO *xmlfile*	This option redirects the query XML output from the screen to the &XML& file. This file has a `.xml` suffix. If you specify WITHSCHEMA in the query, UniVerse creates an `xmlfile.xsd` file in the &XML& directory. If you specify WITHDTD, UniVerse creates an `xmlfile.dtd` file as well.

Examples

This example creates a view that includes all the columns from the INVENTORY table. The column names in INV_VIEW are the same as the names of the columns in the INVENTORY table.

```
>CREATE VIEW INV_VIEW AS SELECT * FROM INVENTORY;
Creating View "INV_VIEW"
Adding Column "PROD.NO"
Adding Column "DESC"
Adding Column "QOH"
Adding Column "COST"
Adding Column "SELL"
Adding Column "MARKUP"
```

If you subsequently add columns to the underlying table using ALTER TABLE, this view is unaffected by the additional columns. That is, the view in the preceding example still contains six columns.

The next example creates a view PRODQTY_VIEW that includes only two columns, PRODNO and QOH, from the INVENTORY table:

```
>CREATE VIEW PRODQTY_VIEW AS SELECT PRODNO, QOH FROM INVENTORY;
```

The next example creates a view DEPTINFO based on the DEPTS table in the OTHERSCHEMA schema:

```
>CREATE VIEW DEPTINFO (DEPTNAME, DEPTHEAD)
SQL+AS SELECT DEPTNAME, MANAGER FROM OTHERSCHEMA.DEPTS;
```

The query specification in the next example extracts the columns ORDERS.ORDERNO and ITEMS.ORDERNO from the ORDERS and ITEMS tables. You must specify unique, corresponding column names (CUSTNO and OCUSTNO) in the CREATE VIEW statement. Since this query specification specifies a virtual column resulting from a calculation (ITEMS.TOTPRICE * 1.25), you must also name the corresponding column in the view (NEWPRICE).

```
>CREATE VIEW SOLD_VIEW (CUSTNO, OCUSTNO, NEWPRICE)
SQL+AS SELECT ORDERS.ORDERNO, ITEMS.ORDERNO,
SQL+ITEMS.TOTPRICE * 1.25
SQL+FROM ORDERS, ITEMS
SQL+WHERE ORDERS.ORDERNO = ITEMS.ORDERNO
SQL+AND ITEMS.TOTPRICE > 150;
```

In the next example the CREATE VIEW statement uses SLIST 0 to create the ORD_VIEW view. When you use a SELECT statement to access the view, SLIST 0 must be active.

```
>CREATE VIEW ORD_VIEW AS SELECT * FROM ORDERS SLIST 0;
     Creating View "ORD_VIEW"
     Adding Column "ORDER.NO"
     Adding Column "CUST.NO"
     Adding Column "PROD.NO"
     Adding Column "QTY"
     Adding Column "ORDER.TOTAL"
     Adding association "BOUGHT"
     >SELECT TO SLIST 0 FROM ORDERS;
     7 record(s) selected to SELECT list #0.
     >>SELECT * FROM ORD_VIEW;
     Order No Customer No Product No Qty. Total.......
     10002        6518        605    1     $55.00
                              501    1
                              502    1
                              504    1
     10006        6518        112    3     $18.00
     10004        4450        704    1     $205.00
                              301    9
     10005        9874        502    9     $45.00
     10003        9825        202    10    $100.00
                              204    10
     10001        3456        112     7     $265.00
                              418    4
                              704    1
     10007        9874        301    3     $30.00
     7 records listed.
```

The next example uses the INQUIRING keyword to create the ORD_VIEW view. When you use a SELECT statement to access the view, you are prompted to enter primary keys.

```
>CREATE VIEW ORD_VIEW AS SELECT * FROM ORDERS INQUIRING;
Creating View "ORD_VIEW"
Adding Column "ORDER.NO"
Adding Column "CUST.NO"
Adding Column "PROD.NO"
Adding Column "QTY"
Adding Column "ORDER.TOTAL"
```

```
Adding association "BOUGHT"
>SELECT * FROM ORD_VIEW;
Primary Key for table ORD_VIEW = 10002
Order No Customer No Product No Qty. Total.......
10002            6518         605 1 $55.00
                              501 1
                              502 1
                              504 1
Primary Key for table ORD_VIEW =
```

Creating an attribute-centric XML document

Using the mapping file described in Mapping file example, on page 127, the following example creates an attribute-centric XML document. To use a mapping file, specify the XMLMAPPING keyword in the RetrieVe statement.

```
>LIST STUDENT LNAME FNAME SEMESTER COURSE_NBR COURSE_GRD
COURSE_NAME TOXML XMLMA
PPING student.map

<?xml version="1.0" encoding="UTF-8"?>
<main>
<STUDENT _ID = "987654321" LNAME = "Miller" FNAME = "Susan">
    <Term SEMESTER = "FA93">
        <Courses_Taken COURSE_NBR = "EG110" COURSE_GRD = "C"
COURSE_NAME = "Engineer
ing Principles"/>
        <Courses_Taken COURSE_NBR = "MA220" COURSE_NAME = "Calculus-
I"/>
        <Courses_Taken COURSE_NBR = "PY100" COURSE_GRD = "B"
COURSE_NAME = "Introduc
tion to Psychology"/>
    </Term>
     <Term SEMESTER = "SP94">
        <Courses_Taken COURSE_NBR = "EG140" COURSE_GRD = "B" COURSE_NAME = "Fluid Me
chanics"/>
        <Courses_Taken COURSE_NBR = "EG240" COURSE_GRD = "B" COURSE_NAME = "Circut T
heory"/>
        <Courses_Taken COURSE_NBR = "MA221" COURSE_NAME = "Calculus -
II"/>
    </Term>
</STUDENT>
<STUDENT _ID = "123456789" LNAME = "Martin" FNAME = "Sally">
    <Term SEMESTER = "SP94">
        <Courses_Taken COURSE_NBR = "PY100" COURSE_NAME =
"Introduction to Psycholog
y"/>
        <Courses_Taken COURSE_NBR = "PE100" COURSE_GRD = "C"
COURSE_NAME = "Golf - I
"/>
    </Term>
</STUDENT>
</main>
>
```

Creating an XML document with a DTD or XML schema

If you only include the TOXML keyword in the RetrieVe statement, the resulting XML document does not include a DTD or XML Schema. To create an XML document that includes a DTD, use the WITHDTD keyword. To create an XML document that includes an XML Schema, use the WITHSCHEMA keyword.

The following example illustrates an XML document that includes a DTD:

```
>LIST STUDENT SEMESTER COURSE_NBR COURSE_GRD COURSE_NAME TOXML
WITHDTD

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ROOT[
<!ELEMENT ROOT (STUDENT*)>
<!ELEMENT STUDENT ( CGA-MV* )>
<!ATTLIST STUDENT
        _ID CDATA #REQUIRED
>
<!ELEMENT CGA-MV  EMPTY>
<!ATTLIST CGA-MV
        SEMESTER CDATA #IMPLIED
        COURSE_NBR CDATA #IMPLIED
        COURSE_GRD CDATA #IMPLIED
        COURSE_NAME CDATA #IMPLIED
>
]>

<ROOT>
<STUDENT _ID = "987654321">
  <CGA-MV SEMESTER = "FA93" COURSE_NBR = "EG110" COURSE_GRD = "C"
COURSE_NAME =
"Engineering Principles" COURSE_NBR = "MA220" COURSE_NAME =
"Calculus- I" COURSE
_NBR = "PY100" COURSE_GRD = "B" COURSE_NAME = "Introduction to
Psychology"/>
  <CGA-MV SEMESTER = "SP94" COURSE_NBR = "EG140" COURSE_GRD = "B"
COURSE_NAME =
"Fluid Mechanics" COURSE_NBR = "EG240" COURSE_GRD = "B"
COURSE_NAME = "Circut Th
eory" COURSE_NBR = "MA221" COURSE_NAME = "Calculus - II"/>
</STUDENT>
<STUDENT _ID = "123456789">
  <CGA-MV SEMESTER = "SP94" COURSE_NBR = "PY100" COURSE_NAME =
"Introduction to
Psychology" COURSE_NBR = "PE100" COURSE_GRD = "C" COURSE_NAME =
"Golf - I"/>
</STUDENT>
</ROOT>
>
```

Using WITHSCHEMA

Use the WITHSCHEMA keyword with the RetrieVe `LIST` command to create an XML schema.

Syntax

The syntax for the `LIST` command is:

```
LIST [DICT | USING [DICT] dictname] filename ... [TOXML [ELEMENTS]
[WITHSCHEMA][WITHDTD] [SCHEMAONLY] TO filename [XMLMAPPING
mapping_file] [TO xmlfile]]...
```

The syntax for the UniVerse SQL `SELECT` command is:

```
SELECT command.
SELECT clause FROM clause
[WHERE clause]
[WHEN clause [WHEN clause]...]
[GROUP BY clause]
[HAVING clause]
[ORDER BY clause]
[report_qualifiers]
[processing_qualifiers]
[TOXML [ELEMENTS] [WITHDTD][WITHSCHEMA][SCHEMAONLY]
[XMLMAPPING mapping_file]]
[XMLDATA extraction_mapping_file]
[TO xmlfile];
```

Note: If you specify both WITHDTD and WITHSCHEMA in the same RetrieVe statement, UniVerse does not produce an XML schema.

WITHSCHEMA creates an XML schema *filename*.xsd. By default, UniVerse writes this file to the &XML& directory. If you do not specify a targetNamespace in the mapping file, the filename.xml's root element contains the following:

`noNamespaceSchemaLocation=`*`filename`*`.xsd`

to define the schema location. If you specify the targetNamespace in the mapping file, UniVerse generates the following:

`schemaLocation="namespaceURL filename.xsd"`

In both of these cases, you can validate the files using the XML schema validator, or the UniVerse BASIC API `XDOMValidate`() function.

Mapping to an external schema

A mapping file enables users to define how the dictionary attributes correspond to the DTD or XML Schema elements. This allows you to create many different forms of XML. Defining settings in the mapping file eliminates the need to specify them in each RetrieVe statement. The following example illustrates how to map to an external schema.

Assume you are trying to map to the following schema:

```
:<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:rocketsoftware="http://www.rocketsoftware.com"
  elementFormDefault="qualified">
  <xsd:annotation>
  <xsd:documentation xml:lang="en">
    This is a sample schema
  </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="transcript">
    <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="student" type="studentType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
    <xsd:complexType name="studentType">
      <xsd:sequence>
        <xsd:element name="semesterReport"
type="semesterReportType" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="ref" type="xsd:string"/>
      <xsd:attribute name="firstName" type="xsd:string"/>
      <xsd:attribute name="lastName" type="xsd:string"/>
    </xsd:complexType>
      <xsd:complexType name="semesterReportType">
        <xsd:sequence>
          <xsd:element name="results" type="resultsType"
minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="term" type="xsd:string"/>
      </xsd:complexType>
        <xsd:complexType name="resultsType">
          <xsd:sequence>
            <xsd:element name="courseGrade" type="xsd:string"/>
            <xsd:element name="courseHours" type="xsd:string"/>
          </xsd:sequence>
          <xsd:attribute name="courseNumber" type="xsd:string"/>
          <xsd:attribute name="courseName" type="xsd:string"/>
          <xsd:attribute name="courseInstructor"
type="xsd:string"/>
        </xsd:complexType>
</xsd:schema>
```

The following map illustrates how to map your student file to this schema. Use the steps shown below to create the map:

1. Set the default settings for the map.
2. Rename singlevalued fields to match the schema names.
3. Rename the element tags used for the association.
4. Rename the multivalued fields.
5. Rename the multi-subvalued fields.

```
 :
<u2
<!-- First set the default settings for the map -->
        root="transcript"
        record="student"
```

```
        targetnamespace1="http://www.rocketsoftware.com"
        schema="type"
        xmlns:rocketsoftware="http://www.rocketsoftware.com"
        treated-as="element"
        collapsemv="1"
/>

 <!-- Rename singlevalued fields to match the schema names -->

<u2
file="STUDENT"
field="@ID"
map-to="ref"
type="S"
treated-as="attribute"
/>

<u2
file="STUDENT"
field="FNAME"
map-to="firstName"
type="S"
treated-as="attribute"
/>

<u2
file="STUDENT"
field="LNAME"
map-to="lastName"
type="S"
treated-as="attribute"
/>

 <!-- Rename the element tags used for the association -->
<u2
file="STUDENT"
field="CGA"
association-mv="semesterReport"
association-ms="results"
/>

 <!-- Rename the multivalued fields -->
<u2
file="STUDENT"
field="SEMESTER"
map-to="term"
type="MV"
treated-as="attribute"
/>

<!-- Rename the multi-subvalued fields -->
<u2
file="STUDENT"
field="COURSE_NBR"
map-to="courseNumber"
type="MSV"treated-as="attribute"
/>

<u2
file="STUDENT"
field="COURSE_NAME"
map-to="courseName"
```

```
treated-as="attribute"
type="MSV"
/>

<u2
file="STUDENT"
field="COURSE_GRD"
map-to="courseGrade"
type="MSV"
/>

<u2
file="STUDENT"
field="COURSE_HOURS"
map-to="courseHours"
type="MSV"
/>

 <u2
file="STUDENT"
field="TEACHER"
map-to="courseInstructor"
type="MSV"
treated-as="attribute"
/>
```

You can now view the output from the schema using the following command:

```
:LIST STUDENT FNAME LNAME CGA SAMPLE 1 TOXML XMLMAPPING
transcript.map
<?xml version="1.0"?>
<transcript
    xmlns:rocketsoftware="http://www.rocketsoftware.com"
>
<student ref = "123456789" firstname = "Sally" lastname =
"Martin">
    <semesterReport term = "SP94">
        <results courseNumber = "PY100" courseInstructor = "Masters">
            <courseName>Introduction to Psychology</courseName>
            <courseGrade></courseGrade>
            <courseHours>3</courseHours>
        </results>
        <results courseNumber = "PE100" courseInstructor = "Fisher">
            <courseName>Golf - I</courseName>
            <courseGrade>C</courseGrade>
            <courseHours>3</courseHours>
        </results>
    </semesterReport>
</student>

 </transcript>
:
```

Creating an XML document with UniVerse SQL

In addition to RetrieVe, you can also create XML documents using UniVerse SQL. To create an XML document through UniVerse SQL, complete the following steps:

1. Analyze the DTD or XML schema associated with the application to which you are sending the XML file. Determine which of your dictionary attributes correspond to the DTD or XML schema elements.
2. Create an XML mapping file, if necessary.
3. List the appropriate fields using the UniVerse SQL `SELECT` command.

To create an XML document from UniVerse SQL, use the UniVerse SQL `SELECT` command.

Syntax

```
SELECT clause FROM clause
[WHERE clause] [WHEN clause [WHEN clause]...]
[GROUP BY clause]
[HAVING clause]
[ORDER BY clause]
[report_qualifiers]
[processing_qualifiers]
[TOXML [ELEMENTS] [WITHDTD] [WITHSCHEMA]
[SCHEMAONLY]
[XMLMAPPING 'mapping_file']]
[XMLDATA 'extraction_mapping_file']
TO 'xmlfile'];
```

Parameter

The following table describes each parameter of the syntax.

Parameter	Description
SELECT clause	Specifies the columns to select from the database.
FROM clause	Specifies the tables containing the selected columns.
WHERE clause	Specifies the criteria that rows must meet to be selected.
WHEN clause	Specifies the criteria that values in a multivalued column must meet for an association row to be output.
GROUP BY clause	Groups rows to summarize results.
HAVING clause	Specifies the criteria that grouped rows must meet to be selected.
ORDER BY clause	Sorts selected rows.
report_qualifiers	Formats a report generated by the SELECT statement.
processing_qualifiers	Modifies or reports on the processing of the SELECT statement.
TOXML	Outputs SELECT results in XML format.
ELEMENTS	Outputs results in element-centric format.
WITHDTD	Output produces a DTD corresponding to the query.
WITHSCHEMA	Output produces an XML schema corresponding to the query.
SCHEMAONLY	The output will produce a schema for the corresponding query.
XMLMAPPING '*mapping_file*'	Specifies a mapping file containing transformation rules for display. This file must exist in the &XML& file.

Parameter	Description
XMLDATA *'extraction_mapping_file'*	Specifies the file containing the extraction rules for the XML document. This file is used for receiving an XML file.
TO *'xmlfile'*	This option redirects the query XML output from the screen to the &XML& file. This file has a `.xml` suffix. If you specify WITHSCHEMA in the query, UniVerse creates an `xmlfile.xsd` file in the &XML& directory. If you specify WITHDTD, UniVerse creates an `xmlfile.dtd` file as well.

You must specify clauses in the SELECT statement in the order shown in the syntax.

For a full discussion of the UniVerse SQL SELECT statement clauses, see *Using UniVerse SQL*.

Processing rules for UniVerse SQL SELECT statements

UniVerse processes SELECT statements much the same as it processes LIST statements, with a few exceptions.

The processing rules for a UniVerse SQL SELECT statement against a single table are the same as the RetrieVe LIST rules. For a discussion of how UniVerse SQL processes these statements, see Creating an XML document from RetrieVe, on page 114.

Processing multiple tables

When processing a UniVerse SQL SELECT statement involving multiple files, UniVerse attempts to keep the nesting inherited in the query in the resulting XML document. Because of this, the order in which you specify the fields in the UniVerse SQL SELECT statement is important for determining how the elements are nested.

Processing in attribute-centric mode

As with RetrieVe, the attribute-centric mode is the default mapping mode.

For more information about the attribute-centric mode, see Attribute-centric mode, on page 115.

- In this mode, UniVerse uses the name of the file containing the first field you specify in the SELECT statement as the outer-most element in the XML output. Any singlevalued fields you specify in the SELECT statement that belong to this file become attributes of this element.
- UniVerse processes the SELECT statement in the order you specify. If it finds a field that belongs to another file, UniVerse creates a sub-element. The name of this sub-element is the new file name. All singlevalued fields found in the SELECT statement that belong to this file are created as attributes for the sub-element.
- If UniVerse finds a multivalued field in the SELECT statement, it creates a sub-element. The name of this element is the name of the association of which this field is a member.
- When you execute UNNEST against an SQL table, it flattens the multivalues into single values.

UniVerse processes the ELEMENTS, WITHDTD, WITHSCHEMA, SCHEMAONLY and XMLMAPPING keywords in the same manner as it processes them for the RetrieVe `LIST` command.

Processing in element-centric mode

When using the element-centric mode, UniVerse automatically prefixes each file name to the association name. For example, the CGA association in the STUDENT file is named STUDENT_CGA in the resulting XML file.

XML limitations in UniVerse SQL

The TOXML keyword is not allowed in the following cases:

- In a sub-query
- In a SELECT statement that is part of an INSERT statement.
- In a SELECT statement that is part of a UNION definition.
- In a SELECT statement that is part of a VIEW definition.

Examples

This section illustrates XML output from the UniVerse SQL SELECT statement. The examples use sample CUSTOMER, TAPES, and STUDENT files.

The following example lists the dictionary records from the CUSTOMER file that are used in the examples:

```
>LIST DICT CUSTOMER

DICT CUSTOMER    02:11:01pm  11 Sep 2007  Page    1

                 Type &
Field..... Field. Field........ Conversion.. Column..... Output
Depth &
Name...... Number Definition... Code........ Heading..... Format
Assoc..

@ID              D    0                        CUSTOMER        10L    S
NAME             D    1                        Customer Name   15T    S
TAPES_RENTED     D    7                        Tapes           10L    M

TAPE_INFO        PH     TAPES_RENTED
                 Type &
                        DATE_OUT
                        DATE_DUE
                        DAYS_BETWEEN
                        TAPE_COST
                        TAPE_NAME
                        UP_NAMES
 >LIST DICT TAPES

DICT TAPES    02:15:08pm  11 Sep 2007  Page    1

                 Type &
Field..... Field. Field........ Conversion.. Column..... Output
Depth &
Name...... Number Definition... Code........ Heading..... Format
Assoc..

@ID              D    0                        TAPES           10L    S
ID               D    0                        TAPES           10L    S
NAME             D    1                        Tape Name       20T    S
CATS
                        RIES',CATEGOR
                        IES,'NAME','X
                        ')
>
```

Using WITHSCHEMA

The syntax for the UniVerse SQL `SELECT` command is:

```
:SELECT command.
SELECT clause FROM clause
[WHERE clause]
[WHEN clause [WHEN clause]...]
[GROUP BY clause]
[HAVING clause]
[ORDER BY clause]
[report_qualifiers]
[processing_qualifiers]
[TOXML [ELEMENTS] [WITHDTD][WITHSCHEMA][SCHEMAONLY]
[XMLMAPPING mapping_file]]
[XMLDATA extraction_mapping_file]
[TO xmlfile];
```

When the `TOXML` command is used in SQL, both the mapping file and the TO xml file need to be quoted.

Creating an XML document from multiple files with a multivalued field

The next example illustrates creating an XML document from multiple files with a multivalued field. In the example, TAPES_RENTED is multivalued and belongs to the TAPE_INFO association in the CUSTOMER file. In the XML document, TAPES_RENTED appears in the CUSTOMER_TAPE_INFO_MV element.

```
>SELECT CUSTOMER.NAME, TAPES.NAME, CAT_NAME, TAPES_RENTED FROM
CUSTOMER, TAPES WHERE TAPES_RENTED = TAPES.@ID ORDER BY
CUSTOMER.NAME  TOXML;


<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
<CUSTOMER NAME = "Fischer, Carrie">
  <TAPES NAME = "Girl Friday">
    <TAPES_CATS-MV CAT_NAME = "Comedy"/>
    <TAPES_CATS-MV CAT_NAME = "Old Classic"/>
  </TAPES>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V110"/>
</CUSTOMER>
<CUSTOMER NAME = "Smith, Harry">
  <TAPES NAME = "Blue Velvet">
    <TAPES_CATS-MV CAT_NAME = "Horror"/>
    <TAPES_CATS-MV CAT_NAME = "Drama"/>
    <TAPES_CATS-MV CAT_NAME = "Avant Garde"/>
  </TAPES>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V2001"/>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V5004"/>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V8181"/>
</CUSTOMER>
<CUSTOMER NAME = "Smith, Harry">
  <TAPES NAME = "Journey Abroad">
    <TAPES_CATS-MV CAT_NAME = "B - Movie"/>
  </TAPES>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V2001"/>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V5004"/>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V8181"/>
</CUSTOMER>
<CUSTOMER NAME = "Smith, Harry">
  <TAPES NAME = "Catch 22">
    <TAPES_CATS-MV CAT_NAME = "Comedy"/>
    <TAPES_CATS-MV CAT_NAME = "Avant Garde"/>
  </TAPES>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V2001"/>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V5004"/>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V8181"/>
</CUSTOMER>
</ROOT>
>
```

Creating an XML document from multiple files with a DTD

The following example illustrates creating an XML document from multiple files with a DTD. To include the DTD, use the WITHDTD keyword.

```
>SELECT CUSTOMER.NAME, TAPES.NAME, CAT_NAME, TAPES_RENTED FROM
CUSTOMER, TAPES WHERE TAPES_RENTED = TAPES.@ID ORDER BY
CUSTOMER.NAME TOXML  WITHDTD;


<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ROOT[
<!ELEMENT ROOT (CUSTOMER*)>
<!ELEMENT CUSTOMER ( TAPES* , CUSTOMER_TAPE_INFO-MV* )>
<!ATTLIST CUSTOMER
        NAME CDATA #REQUIRED
>
<!ELEMENT TAPES ( TAPES_CATS-MV* )>
<!ATTLIST TAPES
        NAME CDATA #IMPLIED
>
<!ELEMENT TAPES_CATS-MV  EMPTY>
<!ATTLIST TAPES_CATS-MV
        CAT_NAME CDATA #IMPLIED
>
<!ELEMENT CUSTOMER_TAPE_INFO-MV  EMPTY>
<!ATTLIST CUSTOMER_TAPE_INFO-MV
        TAPES_RENTED CDATA #IMPLIED
>
]>

<ROOT>
<CUSTOMER NAME = "Fischer, Carrie">
  <TAPES NAME = "Girl Friday">
    <TAPES_CATS-MV CAT_NAME = "Comedy"/>
    <TAPES_CATS-MV CAT_NAME = "Old Classic"/>
  </TAPES>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V110"/>
</CUSTOMER>
<CUSTOMER NAME = "Smith, Harry">
  <TAPES NAME = "Blue Velvet">
    <TAPES_CATS-MV CAT_NAME = "Horror"/>
    <TAPES_CATS-MV CAT_NAME = "Drama"/>
    <TAPES_CATS-MV CAT_NAME = "Avant Garde"/>
  </TAPES>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V2001"/>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V5004"/>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V8181"/>
</CUSTOMER>
<CUSTOMER NAME = "Smith, Harry">
  <TAPES NAME = "Journey Abroad">
    <TAPES_CATS-MV CAT_NAME = "B - Movie"/>
  </TAPES>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V2001"/>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V5004"/>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V8181"/>
</CUSTOMER>
<CUSTOMER NAME = "Smith, Harry">
  <TAPES NAME = "Catch 22">
    <TAPES_CATS-MV CAT_NAME = "Comedy"/>
    <TAPES_CATS-MV CAT_NAME = "Avant Garde"/>
  </TAPES>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V2001"/>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V5004"/>
  <CUSTOMER_TAPE_INFO-MV TAPES_RENTED = "V8181"/>
</CUSTOMER>
</ROOT>
>
```

Creating an XML document From multiple files using a mapping file

As with RetrieVe, you can create a mapping file to define transformation rules differing from the defaults.

For information about creating the mapping file, see Mapping file example, on page 127.

The following mapping file defines rules for the CUSTOMER and TAPES file.

```
>ED &XML& CUST_TAPES.map
Top of "CUST_TAPES.map" in "&XML&", 22 lines, 259 characters.
*--: p
001: <U2xml
002:    file="TAPES"
003:    field = "CAT_NAME"
004:    map-to="Cat_name"
005:    type="mv"
006: />
007: <u2
008:    file="CUSTOMER"
009:    field="TAPES_RENTED"
010:    map-to="Tapes_rented"
011:    TYPE="mv"
012: />
013: <u2
014:    file="CUSTOMER"
015:    field="DATE_OUT"
016:    TYPE="mv"
017: />
018: <u2
019:    file="CUSTOMER"
020:    field="DATE_DUE"
021:    TYPE="mv"
022: />
```

To use this mapping file in the SELECT statement, specify the XMLMAPPING keyword, as shown in the following example:

Note: You must surround the name of the mapping file in single quotation marks.

```
>SELECT CUSTOMER.NAME, TAPES.NAME, CAT_NAME, DATE_OUT, DATE_DUE
FROM CUSTOMER, TAPES WHERE TAPES_RENTED = TAPES.@ID ORDER BY
CUSTOMER.NAME TOXML XMLMAPPING 'CUST_TAPES.MAP';

<?xml version="1.0" encoding="UTF-8"?>
<ROOT>
<CUSTOMER NAME = "Fischer, Carrie">
  <TAPES NAME = "Girl Friday">
    <TAPES_CATS-MV Cat_name = "Comedy"/>
    <TAPES_CATS-MV Cat_name = "Old Classic"/>
  </TAPES>
  <CUSTOMER_TAPE_INFO-MV DATE_OUT = "04/23/94" DATE_DUE =
"04/25/94"/>
</CUSTOMER>
<CUSTOMER NAME = "Smith, Harry">
  <TAPES NAME = "Blue Velvet">
    <TAPES_CATS-MV Cat_name = "Horror"/>
    <TAPES_CATS-MV Cat_name = "Drama"/>
    <TAPES_CATS-MV Cat_name = "Avant Garde"/>
  </TAPES>
  <CUSTOMER_TAPE_INFO-MV DATE_OUT = "04/24/94" DATE_DUE =
"04/26/94"/>
  <CUSTOMER_TAPE_INFO-MV DATE_OUT = "04/23/94" DATE_DUE =
"04/25/94"/>
  <CUSTOMER_TAPE_INFO-MV DATE_OUT = "04/24/94" DATE_DUE =
"04/26/94"/>
</CUSTOMER>
<CUSTOMER NAME = "Smith, Harry">
  <TAPES NAME = "Journey Abroad">
    <TAPES_CATS-MV Cat_name = "B - Movie"/>
  </TAPES>
  <CUSTOMER_TAPE_INFO-MV DATE_OUT = "04/24/94" DATE_DUE =
"04/26/94"/>
  <CUSTOMER_TAPE_INFO-MV DATE_OUT = "04/23/94" DATE_DUE =
"04/25/94"/>
  <CUSTOMER_TAPE_INFO-MV DATE_OUT = "04/24/94" DATE_DUE =
"04/26/94"/>
</CUSTOMER>
<CUSTOMER NAME = "Smith, Harry">
  <TAPES NAME = "Catch 22">
    <TAPES_CATS-MV Cat_name = "Comedy"/>
    <TAPES_CATS-MV Cat_name = "Avant Garde"/>
  </TAPES>
  <CUSTOMER_TAPE_INFO-MV DATE_OUT = "04/24/94" DATE_DUE =
"04/26/94"/>
  <CUSTOMER_TAPE_INFO-MV DATE_OUT = "04/23/94" DATE_DUE =
"04/25/94"/>
  <CUSTOMER_TAPE_INFO-MV DATE_OUT = "04/24/94" DATE_DUE =
"04/26/94"/>
</CUSTOMER>
</ROOT>
>
```

UniVerse BASIC example

The following example illustrates a UniVerse BASIC program that generates an XML document:

```
CMD = "LIST STUDENT LNAME COURSE_NBR COURSE_GRD COURSE_NAME
SEMESTER FNAME"
OPTIONS ="XMLMAPPING=student.map"
STATUS = XMLExecute(CMD,OPTIONS,XMLVAR,XSDVAR)
IF  STATUS = 0 THEN
        STATUS =
XDOMValidate(XMLVAR,XML.FROM.STRING,XSDVAR,XML.FROM.STRING)
        IF STATUS <> XML.SUCCESS THEN
                STATUS = XMLGetError(code,msg)
                PRINT code,msg
      PRINT "Validate FAILED."
      PRINT XSDVAR
      PRINT XMLVAR
        END
   ELSE
      PRINT "Options ":OPTIONS
      PRINT 'XML output:'
      PRINT XMLVAR
      PRINT
   END
END
ELSE
   STATUS = XMLGetError(code,msg)
   PRINT code,msg
   PRINT "XMLExecute failed."
END
```

The next example illustrates the output from the program described in the previous example:

```
<?xml version="1.0"?>
<MAIN>
<STUDENT _ID = "123456789" LNAME = "Martin">
    <SEMESTER>SP94</SEMESTER>
        <COURSE_NBR>PY100</COURSE_NBR>
        <COURSE_NBR>PE100</COURSE_NBR>
</STUDENT>

</MAIN>
:
```

Using the XMLExecute() function

The `XMLExecute` function enables you to create an XML document using the RetrieVe from UniVerse BASIC and returns the `.xml` and `.xsd` in UniVerse BASIC variables. By default, the `XMLExecute` command generates an XML Schema.

Syntax

XMLExecute(*cmd, options, xmlvar, xsdvar*)

Note: This function is case-sensitive.

Parameters

The following table describes each parameter of the syntax.

Parameter	Description	
cmd	Holds the text string of the RetrieVe LIST statement or the UniVerse SQL SELECT statement. [IN]	
options	Each XML-related option is separated by a field mark (@FM). If the option requires a value, the values are contained in the same field, separated by value marks (@VM).	
	WITHDTD	Creates a DTD and binds it with the XML document. By default, UniVerse creates an XML schema. However, if you include WITHDTD in your RetrieVe or UniVerse SQL statement, UniVerse does not create an XML schema, but only produces the DTD.
	ELEMENTS	The XML output is in element-centric format.
	'XMLMAPPING': @VM:'*mapping_file_ name*'	Specifies the mapping file containing transformation rules for display. This file must exist in the &XML& directory.
	'SCHEMA':@VM: '*type*'	The default schema format is ref type schema. You can use the SCHEMA attribute to define a different schema format.
	HIDEMV, HIDEMS	Normally, when UniVerse processes multivalued or multi-subvalued fields, UniVerse adds another level of elements to produce multiple levels of nesting. You have the option of disabling this additional level by adding the HIDEMV and HIDEMS attributes. When these options are on, the generated XML document and the associated DTD or XML schema have fewer levels of nesting.
	HIDEROOT	Allows you to specify to only create a segment of an XML document, for example, using the SAMPLE keyword and other conditional clauses. If you specify HIDEROOT, UniVerse only creates the record portion of the XML document, it does not create a DTD or XML schema.
	'RECORD':@VM: '*newrecords*'	The default record name is FILENAME_*record*. The record attribute in the ROOT element changes the record name.
	'ROOT':@VM: '*newroot*'	The default root element name in an XML document is ROOT. You can change the name of the root element as shown in the following example: root="*root_element_name*"

Parameter	Description	
options (continued)	TARGETNAME-SPACE:@FM:'*namespaceURL*'	UniVerse displays the targetnamespace attribute in the XMLSchema as *targetNamespace*, and uses the URL you specify to define *schemaLocation*. If you define the targetnamespace and other explicit namespace definitions, UniVerse checks if the explicitly defined namespace has the same URL and the targetnamespace. If it does, UniVerse uses the namespace name to qualify the schema element, and the XML document element name.
	COLLAPSEMV, COLLAPSEMS	Normally, when UniVerse processes multivalued or multi-subvalued fields, UniVerse adds another level of elements to produce multiple levels of nesting. You have the option of disabling this additional level by adding the COLLAPSEMV and COLLAPSEMS attributes. When these options are on, the generated XML document and the associated DTD or XML Schema have fewer levels of nesting.
	EMPTYATTRIBUTE	This attribute determines how to display the empty attributes for multivalued fields belonging to an association in the generated XML document and in the associated DTD or XML Schema. When this option is on, the generated XML document and the associated DTD or XML Schema have fewer levels of nesting.
XmlVar	The name of the variable to which to write the generated XML document [OUT]	
XsdVar	The name of the variable in which to store the XML Schema if one is generated along with the XML document. [OUT]	

Hidemv option

This option specifies whether to hide <MV> and </MV> tags for multivalued fields belonging to an association in the generated XML document and in the associated DTD or XML Schema. This parameter applies only if the XML document is created in element mode.

- 0 - Show MV tags for multivalued fields.
- 1 - Hide MV tags for multivalued fields.

You can also use this attribute with `XMLEXECUTE`().

Note: If the document is created in attribute mode, it is not possible to eliminate the extra level of element tags.

Collapsemv option

This option specifies whether to collapse <MV> and </MV> tags, using only one set of these tags for multivalued fields belonging to an association in the generated XML document and in the associated DTD or XMLSchema. This parameter applies only if the XML document is created in element mode.

- 0 - Expand MV tags for multivalued fields.
- 1 - CollapseMV tags for multivalued fields.

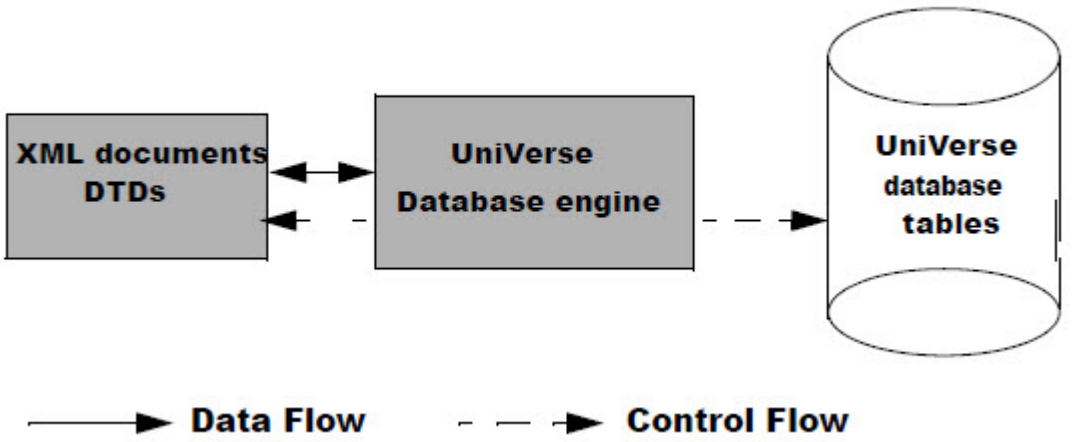
Chapter 7: Receiving an XML document with RetrieVe

Receiving an XML document through UniVerse BASIC

XML documents are text documents, intended to be processed by an application, such as a web browser. UniVerse enables you to receive and create XML documents, and process them through UniVerse BASIC, UniVerse SQL, or RetrieVe.

You can receive an XML document, then read the document through UniVerse BASIC, and execute UniVerse BASIC commands against the XML data.

The following example illustrates the UniVerse implementation of receiving XML documents:



Defining extraction rules

You must define the extraction rules for each XML document you receive. This extraction file defines where to start extracting data from the XML document, how to construct UniVerse data file fields from the data, the name of the data file dictionary to use, and how to treat a missing value.

Note: The extraction file can reside anywhere. We recommend that it reside in the &XML& file, and have a file extension of `.ext`.

Extraction file syntax

An extraction file has the following format:

```
<?XML version = "1.0"?>
<U2xml-extraction xmlns:U2xml="http://www.ibm.com/U2-xml">
```

```
<!-- there must be one and only one <U2xml:extraction>
element with mode/start/dictionary -->
<U2xml:extraction
start=”xpath_expression”
dictionary=”dict1 filename ...”
null=”NULL” | “EMPTY”
/>
<! -- there can be zero or multiple
<U2xml:extraction> elements with
field/path/format -->
<U2xml:field_extraction
field=”field name”
path=”xpath_expression”
/>
...
</U2xml_extraction>
```

The following tables describes the elements of the extraction file.

Element	Description
XML version	The XML version number.
Namespace	The name of the namespace. A namespace is a unique identifier that links an XML markup element to a specific DTD. They indicate to the processing application, for example, a browser, which DTD you are using.
start	Defines the starting node in the XML file. This specifies where UniVerse should begin extracting data from the XML file.
dictionary	Specifies the UniVerse dictionary of the file name to use when viewing the XML data.
null	Determines how to treat a missing node. If null is set to “NULL,” a missing node will be result in the null value in the resulting output. If null is set to EMPTY, a missing node will be replaced with an empty string.
field	The field name.
path	The XPath definition for the field you are extracting.

Defining the XPath

Note: The examples in this section use the STUDENT.F and COURSES files. To create these files, execute the **MAKE.DEMO.FILES** from the TCL prompt.

Note: For the full XPath specification, see http:/www.w3.org/TR/xpath.

At this release, UniVerse supports the following XPath syntax:

Parameter	Description
/	Node path divider.
.	Current node.
..	Parent node.
@	Attributes

Parameter	Description
text()	The contents of the element.
xmldata()	The remaining, unparsed, portion of the selected node.
,	Node path divider, and also specifies multivalued field.

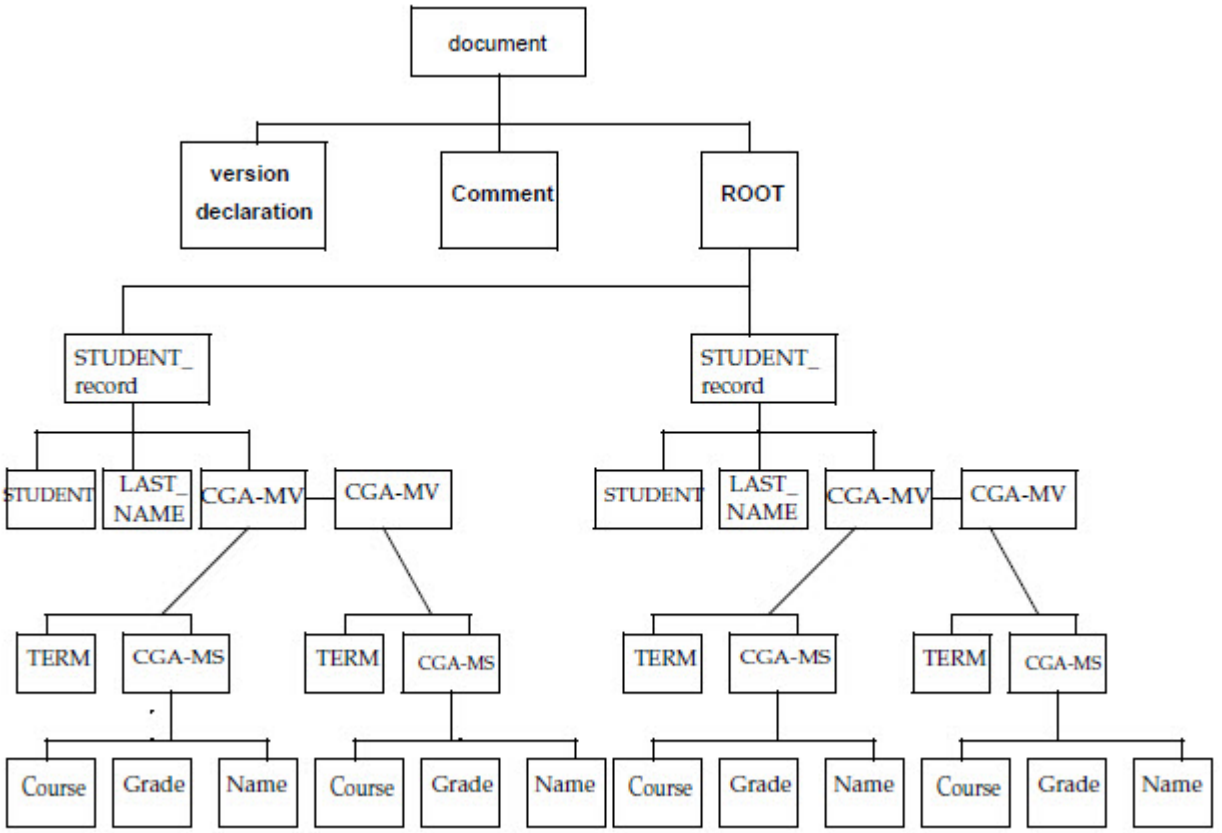
Consider the following DTD and XML document:

```
<?xml version="1.0"?>
<!DOCTYPE ROOT[
<!ELEMENT ROOT (STUDENT_record*)>
<!ELEMENT STUDENT_record ( STUDENT , Last_Name , CGA-MV* )>
<!ELEMENT STUDENT (#PCDATA) >
<!ELEMENT Last_Name (#PCDATA) >
<!ELEMENT CGA-MV ( Term* , CGA-MS* )
<!ELEMENT Term (#PCDATA) >
<!ELEMENT CGA-MS ( Crs__* , GD* , Course_Name* )>
<!ELEMENT Crs__ (#PCDATA) >
<!ELEMENT GD (#PCDATA) >
<!ELEMENT Course_Name (#PCDATA) >
]>

 <ROOT>
<STUDENT_record>
    <STUDENT>424-32-5656</STUDENT>
    <Last_Name>Martin</Last_Name>
    <CGA-MV>
        <Term>SP94</Term>
        <CGA-MS>
            <Crs__>PY100</Crs__>
            <GD>C</GD>
            <Course_Name>Introduction to Psychology</Course_Name>
        </CGA-MS>
        <CGA-MS>
            <Crs__>PE100</Crs__>
            <GD>C</GD>
            <Course_Name>Golf - I </Course_Name>
        </CGA-MS>
    </CGA-MV>
</STUDENT_record>
<STUDENT_record>
    <STUDENT>414-44-6545</STUDENT>
    <Last_Name>Offenbach</Last_Name>
    <CGA-MV>
        <Term>FA93</Term>
        <CGA-MS>
            <Crs__>CS104</Crs__>
            <GD>D</GD>
            <Course_Name>Database Design</Course_Name>
        </CGA-MS>
        <CGA-MS>
            <Crs__>MA101</Crs__>
            <GD>C</GD>
            <Course_Name>Math Principles </Course_Name>
        </CGA-MS>
        <CGA-MS>
            <Crs__>FA100</Crs__>
            <GD>C</GD>
            <Course_Name>Visual Thinking </Course_Name>
        </CGA-MS>
        </CGA-MV>
```

```
    <CGA-MV>
        <Term>SP94</Term>
    <CGA-MS>
            <Crs__>CS105</Crs__>
            <GD>B</GD>
            <Course_Name>Database Design</Course_Name>
        <CGA-MS>
            <Crs__>MA102</Crs__>
            <GD>C</GD>
            <Course_Name>Introduction of Psychology</Course_Name>
        </CGA-MS>
    </CGA-MV>
<STUDENT_record>
    <STUDENT>221-34-5665</STUDENT>
    <Last_Name>Miller</Last_Name>
    <CGA-MV>
        <Term>FA93</Term>
        <CGA-MS>
            <Crs__>EG110</Crs__>
            <GD>C</GD>
            <Course_Name>Engineering Principles</Course_Name>
        </CGA-MS>
        <CGA-MS>
            <Crs__>PY100</Crs__>
            <GD>B</GD>
            <Course_Name>Introduction to Psychology</Course_Name>
        </CGA-MS>
    </CGA-MV>
        <Term>SP94</Term>
        <CGA-MS>
            <Crs__>EG140</Crs__>
            <GD>B</GD>
            <Course_Name>Fluid Mechanics</Course_Name>
        </CGA-MS>
        <CGA-MS>
            <Crs__>MA221</Crs__>
            <GD>B</GD>
            <Course_Name>Calculus -- II</Course_Name>
        </CGA-MS>
    </CGA-MV>
 </STUDENT_record>
</ROOT>
```

This document could be displayed as a tree, as shown in the following example:



In the previous example, each element in the XML document appears in a box. These boxes are called nodes when using XPath terminology. As shown in the example, nodes are related to each other. The relationships in this example are:

- The document node contains the entire XML document.
- The document node contains three children: the version declaration, the comment node, and the ROOT node. These three children are siblings.
- The ROOT node contains two STUDENT nodes, which are children of ROOT, and are siblings of each other.
- The STUDENT node contains three nodes: the ID, NAME, and CGA-MV. These nodes are children of the STUDENT node, and are siblings of each other.
- The CGA-MV node contains TERM nodes and CGA-MS nodes. These nodes are children of the CGA-MV node, and are siblings of each other.
- Finally, the CGA-MS node contains three nodes: the Course, Grade, and Name nodes. These three nodes are children of the CGA-MS node, and are siblings of each other.

When you define the XPath in the extraction file, you must indicate how to treat these different nodes.

Defining the starting location

The first thing to define in the extraction file is the starting node in the XML document from which you want to begin extracting data. In our example, we want to start at the STUDENT_record node. You can also define the dictionary file to use when executing RetrieVe LIST statements or UniVerse SQL SELECT statements against the data.

The following example illustrates how to specify the STUDENT_record node as the starting node, and use the STUDENT dictionary file:

```
<file_extraction start = "ROOT/STUDENT_record " dictionary = "STUDENT"/
>
```

If you want to start the extraction at the CGA-MV node, specify the file extraction node as follows:

```
<file_extraction start = "ROOT/STUDENT_record/CGA-MV" dictionary =
"STUDENT"/>
```

Specifying field equivalents

Next, you specify the rules for extracting fields from the XML document. In this example, there are six fields to extract (@ID, NAME, TERM, COURSE, GRADE and NAME).

Extracting singlevalued fields

The following example illustrates how to define the extraction rule for two singlevalued fields:

```
<field_extraction field = "@ID" path = "STUDENT/text()",/>
```

```
<field_extraction field = "LNAME" path = "Last_Name/text()",/>
```

In the first field extraction, the @ID value in the UniVerse record will be extracted from the STUDENT node. The text in the STUDENT node will be the value of @ID.

In the next field extraction rule, the LNAME field will be extracted from the text found in the Last_Name node in the XML document.

Extracting multivalued fields

To access multivalued data in the XML document, you must specify the location path relative to the start node (full location path).

UniVerse uses the "/" character to specify levels of the XML document. The "/" tells the xmlparser to go to the next level when searching for data.

Use a comma (",") to tell the xmlparser where to place marks in the data.

The following example illustrates how to define the path for a multivalued field (SEMESTER) in the XML document:

```
<field_extraction field "SEMESTER" path = "CGA-MV,Term/text()" />
```

In this example, the value of the SEMESTER field in the UniVerse data file will be the text in the Term node. The "/" in the path value specifies multiple levels in the XML document, as follows:

1. Start at the CGA-MV node in the XML document.
2. From the CGA-MV node, go to the next level, the Term node.
3. Return the text from the Term node as the first value of the SEMESTER field in the UniVerse data file.
4. Search for the next CGA-MV node under the same STUDENT, and extract the text from the Term node belonging to that CGA-MV node, and make it the next multivalue. The comma tells the xmlparser to get the node preceding the command for the next sibling.
5. Continue processing all the CGA-MV nodes belonging to the same parent.

   The SEMESTER field will appear in the following manner:

   ```
   Term<Value mark>Term<Value Mark>...
   ```

Extracting multi-subvalued fields

As with multivalued fields, UniVerse uses the “/” character to specify levels of the XML document. The “/” tells the xmlparser to go to the next level when searching for data.

Use the comma (“,”) to define where to place marks in the data. You can specify 2 levels of marks, value marks and subvalue marks.

Consider the following example of a field extraction XPath definition:

```
<field_extraction field = “COURSE_NBR” path = “CGA-MV, CGA-MS,
Course_Name/ text()” />
```

In this case, the resulting data will appear as follows:

<Value Mark>Course_Name <subvalue mark>Course_Name<subvalue mark>Course_Name...<Value Mark>...

Suppose the XPath definition contains another level of data, as shown in the next example:

```
<field_extraction field = “COURSE_NBR” path = “CGA-MV/CGA-MS/
Course_Name/Comment/text()”/>
```

You must determine where you want the marks to appear in the resulting data. If you want Comment to represent the multi-subvalue, begin inserting commas after CGA-MS, since the Comment is three levels below CGA-MS.

```
<field_extraction field = “COURSE_NBR” path = “CGA-MV/CGA-MS,
Course_Name,Comment/text()” />
```

Suppose we add yet another level of data to XPath definition:

```
<field_extraction field = “COURSE_NBR” path = “CGA-MV/CGA-MS,
Course_Name,Comment,activities/text()” />
```

This is not a valid XPath, since there are more than three levels of XML data. If you want your data to have subvalue marks between Comment and activities, change the XPath definition as follows:

```
<field_extraction field = “COURSE_NBR” path = “CGA-MV/CGA-MS/
Course_Name,Comment,activities/text()” />
```

The “/” and the “,” characters are synonymous when defining the navigation path, UniVerse still uses the “/” AND the “,” to define the navigation path of the data, but only the “,” to determine the location of the marks in the resulting data.

Like multivalued fields, you must start at the XPath with the parent node of the multivalue.

The next example illustrates how to extract data for a multi-subvalued field:

```
<field_extraction field = “COURSE_NBR” path = “CGA-MV, CGA-MS, Crs__/
text()” />
```

The COURSE_NBR field in the UniVerse data file will be extracted as follows:

1. Start at the CGA-MV node in the XML document, under the start node (ROOT/STUDENT_record).
2. From the first CGA-MV node, go to the next level, the CGA-MS node.
3. From the first CGA-MS node, go to the Crs__ node. Return the text from the Crs__node, and make that text the first multi-subvalue of COURSE_NBR.
4. Go back to the CGA-MS node, and search the siblings of the CGA-MS nodes to see if there are any more CGA-MS nodes of the same name. If any are found, return the Crs__/text() under these nodes, and make them the next multi-subvalues of COURSE_NBR.
5. Go back to the CGA-MV node and search for siblings of the CGA-MS node that have the same CGA-MV node name. If any are found, repeat steps 3 and 4 to get the values for these CGA-MV nodes, and make them multivalues.

The COURSE_NBR field will look like this:

```
<Field Mark>Crs__text() value under 1st CGA-MS node of 1st CGA-MV
node<multi-subvalue mark>Crs__text() under 2nd CGA-MS node of 1st
CGA-MV node<multi-subvalue mark>...<multivalue mark>Crs__text()
under 1st CGA-MS node of the 2nd CGA-MV node<multi-subvalue
mark>Crs__text() under 2nd CGA-MS node of the 2nd CGA-MV
node<multi-subvalue mark>Crs__text() value under the 3rd CGS-MS node
of the 2nd CGA-MV node>...<Field Mark>
```

The following example illustrates the complete extraction file for the above examples

```
    <U2XML_extraction>
        <file_extraction start = "/ROOT/STUDENT_record" dictionary =
"D_MYSTUDENT"
        <!--field extraction rule in element mode-->
            <field_extraction field = "@ID" path = "STUDENT/text()"/>
        <field_extraction field = "LNAME" path = "Last_Name/text()"/>
        <field_extraction field = "SEMESTER" path = "CGA-MV/Term/text()"/>
            <field_extraction field = "COURSE_NBR" path = "CGA-MV, CGA-MS,
Crs__/text"/>
            <field_extraction field = "COURSE_GRD" path = "CGA-MV, CGA-MS,
GD/text()"/>
            <field_extraction field = "COURSE_NAME" path = "CGA-MV, CGA-MS,
Course_Name/text()"/>
    </U2XML_extraction>
:
```

Extracting XML data through UniVerse BASIC

Complete the following steps to access the XML data through UniVerse BASIC:

Context for the current task

1. Familiarize yourself with the elements of the DTD associated with the XML data you are receiving.
2. Create the extraction file for the XML data.
3. Prepare the XML document using the UniVerse BASIC `PrepareXML` function.
4. Open the XML document using the UniVerse BASIC `OpenXMLData` function.
5. Read the XML data using the UniVerse BASIC `ReadXMLData` function.
6. Close the XML document using the UniVerse BASIC `CloseXMLData` function.
7. Release the XML document using the UniVerse BASIC `ReleaseXML` function.

Preparing the XML document

You must first prepare the XML document in the UniVerse BASIC program. This step allocates memory for the XML document, opens the document, determines the file structure of the document, and returns the file structure.

Syntax

`Status=`**`PrepareXML`**`(`*`xml_file`*`,`*`xml_handle`*`)`

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
xml_file	The path to the file where the XML document resides.
xml_handle	The return value. The return value is the UniVerse BASIC variable for *xml_handle*. Status is one of the following return values: XML.SUCCESS Success XML.ERROR Error

Example

The following example illustrates use of the `PrepareXML` function:

```
STATUS = PrepareXML("&XML&/MYSTUDENT.XML",STUDENT_XML)
IF STATUS=XML.ERROR THEN
     STATUS = XMLError(errmsg)
     PRINT "error message ":errmsg
     STOP "Error when preparing XML document "
END
```

Opening the XML document

After you prepare the XML document, open it using the `OpenXMLData` function.

Syntax

`Status=`**`OpenXMLData`**`(`*`xml_handle,xml_data_extraction_rule, xml_data_handle`*`)`

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
xml_handle	The XML handle generated by the `PrepareXML`() function.
*xml_data_extraction_* *rule*	The path to the XML extraction rule file.
xml_data_handle	The XML data file handle. The following are the possible return values: XML.SUCCESS Success. XML.ERROR Failed XML.INVALID.HANDLE Invalid XML handle

Example

The following example illustrates use of the `OpenXMLData` function:

```
status = OpenXMLData("STUDENT_XML", "&XML&/MYSTUDENT.ext",STUDENT_XML_DATA)
If status = XML.ERROR THEN
    STOP "Error when opening the XML document. "
END
IF status = XML.INVALID.HANDLE THEN
```

```
    STOP "Error: Invalid parameter passed."
END
```

Reading the XML Document

After opening the XML document, read the document using the `ReadXMLData` function. UniVerse BASIC returns the XML data as a dynamic array.

Syntax

`Status=`**`ReadXMLData`**`(`*`xml_data_handle, rec`*`)`

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
xml_data_handle	A variable that holds the XML data handle created by the `OpenXMLData` function.
rec	A mark-delimited dynamic array containing the extracted data. Status if one of the following: XML.SUCCESS Success XML.ERROR Failure XML.INVALID.HANDLE 2 Invalid *xml_data_handle* XML.EOF End of data

After you read the XML document, you can execute any UniVerse BASIC statement or function against the data.

Example

The following example illustrates use of the `ReadXMLData` function:

```
MOREDATA=1
LOOP WHILE (MOREDATA=1)
     status = ReadXMLData(STUDENT_XML,rec)
     IF status = XML.ERROR THEN
         STOP "Error when preparing the XML document. "
END ELSE IF status = XML.EOF THEN
        PRINT "No more data"
        MOREDATA = 0
END ELSE
        PRINT "rec = ":rec
    END
REPEAT
```

Closing the XML document

After you finish using the XML data, use `CloseXMLData` to close the dynamic array variable.

Syntax

`Status=`**`CloseXMLData`**`(`*`xml_data_handle`*`)`

where *xml_data_handle* is the name of the XML data file handle created by the `OpenXMLData()` function.

The return value is one of the following:

XML.SUCCESS	Success
XML.ERROR	Failure
XML.INVALID.HANDLE2	Invalid *xml_data_handle*

Example

The following example illustrates use of the `CloseXMLData` function:

```
status = CloseXMLData(STUDENT_XML)
```

Releasing the XML document

Finally, release the dynamic array variable using the `ReleaseXML` function.

Syntax

`ReleaseXML``(`*`XMLhandle`*`)`

where *XMLhandle* is the XML handle created by the `PrepareXML()` function.

`ReleaseXML` destroys the internal DOM tree and releases the associated memory.

Getting error messages

Use the `XMLError` function to get the last error message.,

Syntax

`XMLError``(errmsg)`

Where errmsg is the error message string, or one of the following return values:

XML.SUCCESS	Success
XML.ERROR	Failure

Example

The following example illustrates a UniVerse BASIC program that prepares, opens, reads, closes, and releases an XML document:

```
# INCLUDE UNIVERSE.INCLUDE XML.H
STATUS=PrepareXML("&XML&/MYSTUDENT.XML",STUDENT_XML)
IF STATUS=XML.ERROR THEN
   STATUS = XMLError(errmsg)
   PRINT "error message ":errmsg
   STOP "Error when preparing XML document "
   END

STATUS =
OpenXMLData("STUDENT_XML","&XML&/MYSTUDENT.ext",STUDENT_XML_DATA)

IF STATUS = XML.ERROR THEN
   STOP "Error when opening the XML document. "
END

IF STATUS = XML.INVALID.HANDLE THEN
   STOP "Error: Invalid parameter passed." END

MOREDATA=1
  LOOP WHILE (MOREDATA=1)
   STATUS=ReadXMLData(STUDENT_XML_DATA,rec)
   IF STATUS = XML.ERROR THEN
        STOP "Error when preparing the XML document. "
   END ELSE IF STATUS = XML.EOF THEN
       PRINT "No more data"
       MOREDATA = 0
   END ELSE
       PRINT "rec = ":rec
       PRINT "rec = ":rec
   END
REPEAT
STATUS = CloseXMLData(STUDENT_XML_DATA)
STATUS = ReleaseXML(STUDENT_XML)
```

Displaying an XML document through RetrieVe

You can display the contents of an XML file through RetrieVe by defining an extraction file, preparing the XML document, then using `LIST` to display the contents.

Preparing the XML document

Before you execute the LIST statement against the XML data, you must first prepare the XML file using the `PREPARE XML` command.

`PREPARE.XML` *`xml_file xml_data`*

xml_file is the path to the location of the XML document.

xml_data is the name of the working file you assign to the XML data.

The following example illustrates preparing the MYSTUDENT.XML document:

```
PREPARE.XML "&XML&/MYSTUDENT.XML" STUDENT_XML

PREPARE.XML successful.
```

Listing the XML data

Use the RetrieVe `LIST` command with the XMLDATA option to list the XML data.

Syntax

```
LIST XMLDATA xml_data "extraction_file" [fields]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
XMLDATA *xml_data*	Specifies to list the records from the *xml_data* you prepared.
extraction_file	The full path to the location of the extraction file. You must surround the path in quotation marks.
fields	The fields from the dictionary you specified in the extraction file that you want to display.

When you list an XML document, RetrieVe uses the dictionary you specify in the extraction file. The following example lists the dictionary records for the MYSTUDENT dictionary:

```
>LIST DICT MYSTUDENT

DICT MYSTUDENT    10:25:32am  19 Oct 2001  Page    1

                Type &
Field..... Field. Field........ Conversion.. Column..... Output
Depth &
Name...... Number Definition... Code........ Heading..... Format
Assoc..

@ID             D    0                      MYSTUDENT      10L    S
LNAME           D    1                      Last Name      15T    S
SEMESTER        D    2                      Term           4L     M
CGA
COURSE_NBR      D    3                      Crs #          5L     M
CGA
COURSE_GRD      D    4                      GD             3L     M
CGA

5 records listed.
```

The fields in the dictionary record must correspond to the position of the fields in the XML extraction file. In the following extraction file, @ID is position 0, LNAME is position 1, SEMESTER is position 2, COURSE_NBR is position 3, COURSE_GRD is position 4, and COURSE_NAME is position 5. The dictionary of the MYSTUDENT file matches these positions.

The following example illustrates listing the fields from the MYSTUDENT XML document, using the MYSTUDENT.EXT extraction file:

```
LIST XMLDATA STUDENT_XML "&XML&/MYSTUDENT.EXT" LNAME SEMESTER COURSE_NBR
COURSE
_GRD COURSE_NAME 11:58:01am  19 Oct 2001  PAGE    1
MYSTUDENT.   Last Name......   Term   Crs #   GD.   Course
Name..............

424-32-565   Martin            SP94   PY100   C     Introduction to
Psycholog
6                                                   y
                                      PE100   C     Golf - I
414-44-654   Offenbach         FA93   CS104   D     Database Design
5
                                      MA101   C     Math Principals
                                      FA100   C     Visual Thinking
                               SP94   CS105   B     Database Design
                                      MA102   C     Algebra
                                      PY100   C     Introduction to
Psycholog
                                                    y
221-34-566   Miller            FA93   EG110   C     Engineering Principles
5
                                      MA220   B     Calculus- I
                                      PY100   B     Introduction to
Psycholog
                                                    y
                               SP94   EG140   B     Fluid Mechanics
                                      EG240   B     Circut Theory
                                      MA221   B     Calculus - II

978-76-667   Muller            FA93   FA120   A     Finger Painting
6
                                      FA230   C     Photography Principals
                                      HY101   C     Western Civilization
                               SP94   FA121   A     Watercorlors
                                      FA231   B     Photography Practicum
                                      HY102   I     Western Civilization -
15
                                                    00 to 1945
521-81-456   Smith             FA93   CS130   A     Intro to Operating
System
4                                                   s
                                      CS100   B     Intro to Computer
Science
                                      PY100   B     Introduction to
Psycholog
                                                    y
                               SP94   CS131   B     Intro to Operating
System
                                                    s
                                      CS101   B     Intro to Computer
Science
                                      PE220   A     Racquetball
291-22-202   Smith             SP94   FA100   B     Visual Thinking
1
6 records listed.
>
```

Release the XML document

When you finish with the XML document, release it using the RELEASE.XML.

Syntax

RELEASE.XML *xml_data*

Displaying an XML document through UniVerse SQL

You can display an XML document through UniVerse SQL using the SELECT statement.

Preparing the XML document

Before you execute the SELECT statement against the XML data, you must first prepare the XML file using the `PREPARE XML` command.

Syntax

`PREPARE.XML` *`xml_file xml_data`*

xml_file is the path to the location of the XML document.

xml_data is the name of the working file you assign to the XML data.

The following example illustrates preparing the MYSTUDENT.XML document:

```
PREPARE.XML "&XML&/MYSTUDENT.XML"
STUDENT_XMLPREPARE.XML successful.
```

Listing the XML data

Use the UniVerse SQL `SELECT` command with the XMLDATA option to list the XML data.

Syntax

```
SELECT clause FROM XMLDATA xml_data extraction_file
[WHERE clause]
[WHEN clause [WHEN clause]...]
[GROUP BY clause]
[HAVING clause]
[ORDER BY clause]
[report_qualifiers]
[processing_qualifiers]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
SELECT clause	Specifies the columns to select from the database.
FROM XMLDATA *xml_data*	Specifies the XML document you prepared from which you want o list data.
extraction_file	Specifies the file containing the extraction rules for the XML document.
WHERE clause	Specifies the criteria that rows must meet to be selected.
WHEN clause	Specifies the criteria that values in a multivalued column must meet for an association row to be output.
GROUP BY clause	Groups rows to summarize results.
HAVING clause	Specifies the criteria that grouped rows must meet to be selected.

Parameter	Description
ORDER BY clause	Sorts selected rows.
report_qualifiers	Formats a report generated by the SELECT statement.
processing_qualifiers	Modifies or reports on the processing of the SELECT statement.

You must specify clauses in the SELECT statement in the order shown in the syntax. You can use the SELECT statement with type 1, type 19, and type 25 files only if the current isolation level is 0 or 1.

For a full discussion of the UniVerse SQL SELECT statement clauses, see the *UniVerse SQL Reference*.

The following example illustrates displaying the XML document using the UniVerse SQL SELECT statement:

```
>SELECT * FROM XMLDATA STUDENT_XML "&XML&/MYSTUDENT.EXT";
MYSTUDENT.   Last Name.....   Term   Crs #   Course Name.............
GD.

424-32-565   Martin           SP94   PY100   Introduction to Psycholog   C
6                                             y
                                     PE100   Golf - I                    C
414-44-654   Offenbach        FA93   CS104   Database Design             D
5
                                     MA101   Math Principals             C
                                     FA100   Visual Thinking             C
                              SP94   CS105   Database Design             B
                                     MA102   Algebra                     C
                                     PY100   Introduction to Psycholog   C
                                             y
221-34-566   Miller           FA93   EG110   Engineering Principles      C
5
                                     MA220   Calculus- I                 B
                                     PY100   Introduction to Psycholog   B
                                             y
                              SP94   EG140   Fluid Mechanics             B
                                     EG240   Circut Theory               B
                                     MA221   Calculus - II               B
978-76-667   Muller           FA93   FA120   Finger Painting             A
6
Press any key to continue...

MYSTUDENT.   Last Name.....   Term   Crs #   Course Name.............
GD.

                                     FA230   Photography Principals      C
                                     HY101   Western Civilization        C
                              SP94   FA121   Watercorlors                A
                                     FA231   Photography Practicum       B
                                     HY102   Western Civilization - 15   I
                                             00 to 1945
521-81-456   Smith            FA93   CS130   Intro to Operating System   A
4                                            s
                                     CS100   Intro to Computer Science   B
                                     PY100   Introduction to Psycholog   B
                                             y
                              SP94   CS131   Intro to Operating System   B
                                             s
                                     CS101   Intro to Computer Science   B
                                     PE220   Racquetball                 A
291-22-202   Smith            SP94   FA100   Visual Thinking             B
1

6 records listed.
>
```

Release the XML document

When you finish with the XML document, release it using the RELEASE.XML.

Syntax

RELEASE.XML *xml_data*

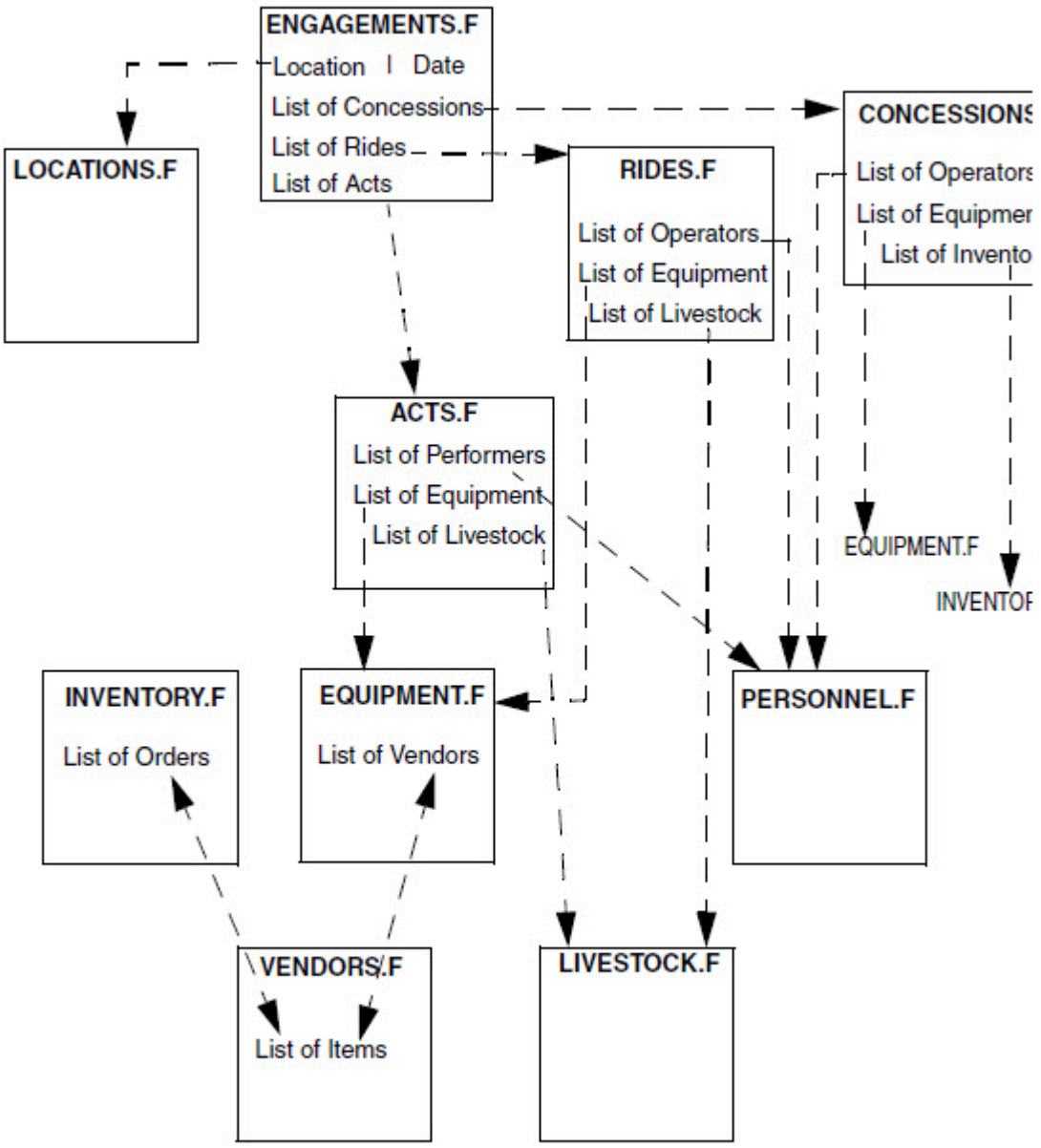
Appendix A: The sample database

The Circus database, used for the examples in this manual, is a database that might typically be used to conduct the day-to-day business of “The Sample Database.” a traveling circus.

The database consists of the following 10 UniVerse files, listed in alphabetical order:

File	Description
ACTS.F	Contains a record for each act, including its description and duration, and the personnel, animals, and equipment needed.
CONCESSIONS.F	Contains a record for each concession, including a description and the personnel, equipment, and stock required.
ENGAGEMENTS.F	Contains a record for each scheduled booking, including its location, date, and time, any advance paid, the acts scheduled, and gate, ride, and concession receipts.
EQUIPMENT.F	Contains a record of each piece of equipment owned, and its description, depreciation, cost, use life, tax life, voltage, and acquisition date, and the vendor from whom it was purchased.
INVENTORY.F	Contains a record for each inventory item stocked, including its description, type, quantity on hand, cost, selling price, and the vendor source and quantity of each purchase made.
LIVESTOCK.F	Contains a record for each animal, including its name, description, use, date of birth, country of origin, estimated life span, and vaccination history.
LOCATIONS.F	Contains a record for each location at which the circus appears, including its name, address, phone numbers, acreage, seating capacity, media contacts in the area, and government agency contacts and the fees they charge.
PERSONNEL.F	Contains a record for each employee, including badge number, name, address, phone, date of birth, dependents, benefits, and the jobs (equipment, rides, concessions) for which they’re qualified and the hourly rate for each.
RIDES.F	Contains a record for each ride owned, including a description, qualified operators, and equipment and animals needed.
VENDORS.F	Contains a record for each vendor that supplies equipment or stock to the circus, including company name, address, phone numbers, terms, and contact names.

The relationships among the files is illustrated in the following example, and the detailed structure of each file, in the form of its file dictionary, is shown on the pages following the figure.


ENGAGEMENTS.F
Location | Date
List of Concessions
List of Rides
List of Acts
LOCATIONS.F
RIDES.F
List of Operators
List of Equipment
List of Livestock
CONCESSIONS
List of Operators
List of Equipmer
List of Invento
ACTS.F
List of Performers
List of Equipment
List of Livestock
EQUIPMENT.F
INVENTOF
INVENTORY.F
List of Orders
EQUIPMENT.F
List of Vendors
PERSONNEL.F
VENDORS.F
List of Items
LIVESTOCK.F
-------- Broken lines indicate logical joins between files.

DICT ACTS.F file

```
                Type &
Field.......... Field. Field........ Conversion.. Column..... Output Depth &
Name........... Number Definition... Code........ Heading..... Format Assoc..

@ID             D    0                           ACTS.F       10L    S
ACT.NO          D    0                                        5R     S
DESCRIPTION     D    1                                        24T    S
DURATION        D    2                                        5R     S
OPERATOR        D    3                                        5R     M
ANIMAL.ID       D    4                                        5R     M
EQUIP.CODE      D    5                                        5R     M
OP.NAME         I      TRANS(PERSONN                Operator     25T    M
                       EL.F,OPERATOR
                       ,NAME,"X")
ANIMALS         I      TRANS(LIVESTO                Animals      10T    M
                       CK.F,ANIMAL.I
                       D,NAME,"X")
EQUIPMENT       I      TRANS(EQUIPME                Equipment    25T    M
                       NT.F,EQUIP.CO
                       DE,DESCRIPTIO
                       N,"X")
@REVISE         PH     DESCRIPTION
                       DURATION
                       OPERATOR
                       ANIMAL.ID
                       EQUIP.CODE
@               PH     ID.SUP ACT.NO
                       DESCRIPTION
                       DURATION

12 records listed.
```

DICT CONCESSIONS.F file

```
                Type &
Field.......... Field. Field........ Conversion.. Column..... Output Depth &
Name........... Number Definition... Code........ Heading..... Format Assoc..

@ID             D    0                           CONCESSIONS.F 10L    S
CONC.NO         D    0                                         5R     S
DESCRIPTION     D    1                                         25T    S
OPERATOR        D    2                                         5R     M
EQUIP.CODE      D    3                                         5R     M
ITEM.CODE       D    4                                         5R     M STOCK
QTY             D    5                                         5R     M STOCK
@               PH     ID.SUP
                       CONC.NO
                       DESCRIPTION
@REVISE         PH     DESCRIPTION
                       OPERATOR
                       EQUIP.CODE
                       ITEM.CODE QTY
STOCK           PH     ITEM.CODE QTY


10 records listed.
```

DICT ENGAGEMENTS.F file

```
                Type &
Field.......... Field. Field........ Conversion.. Column......... Output Depth &
Name........... Number Definition... Code........ Heading........ Format Assoc..

@ID             D    0                           ENGAGEMENTS.F    10L    S
TIME            D    1               MTH                          10R    S
ADVANCE         D    2               MD2$,                        12R    S
GATE.NUMBER     D    3                                            5R     M
GATE.REVENUE    D    4               MD2$,                        12R    M GATES
                                                                          .ASSOC
GATE.TICKETS    D    5                                            5R     M GATES
                                                                          .ASSOC
ACT.NO          D    6                                            5R     M
RIDE.ID         D    7                                            5R     M RIDES
                                                                          .ASSOC
RIDE.REVENUE    D    8               MD2$,                        12R    M RIDES
                                                                          .ASSOC
RIDE.TICKETS    D    9                                            5R     M RIDES
                                                                          .ASSOC
CONC.ID         D   10                                            5R     M CONCS
                                                                          .ASSOC
CONC.REVENUE    D   11               MD2$,                        12R    M CONCS
                                                                          .ASSOC
CONC.TICKETS    D   12                                            5R     M CONCS
                                                                          .ASSOC
LABOR           D   13                                            5R     S
PAY             D   14               MD2$,                        10R    S
LOCATION.CODE   I       FIELD(@ID,"*"                             7L     S
                        ,1)
DATE            I       FIELD(@ID,"*" D2/                         10R    S
                        ,2)
RIDES.ASSOC     PH      RIDE.ID
                        RIDE.REVENUE
                        RIDE.TICKETS
CONCS.ASSOC     PH      CONC.ID
                        CONC.REVENUE
                        CONC.TICKETS
@REVISE         PH      TIME ADVANCE
                        GATE.NUMBER
                        GATE.REVENUE
                        GATE.TICKETS
                        ACT.NO
                        RIDE.ID
                        RIDE.REVENUE
                        RIDE.TICKETS
                        CONC.ID
                        CONC.REVENUE
                        CONC.TICKETS
                        LABOR PAY
GATES.ASSOC     PH      GATE.NUMBER
                        GATE.REVENUE
                        GATE.TICKETS
@               PH      ID.SUP
                        LOCATION.CODE
                        DATE TIME
                        ADVANCE

22 records listed.
```

DICT EQUIPMENT.F file

```
                Type &
Field.......... Field. Field........ Conversion.. Column..... Output Depth &
Name........... Number Definition... Code........ Heading..... Format Assoc..

@ID             D    0                           EQUIPMENT.F    10L    S
EQUIP.CODE      D    0                                          5R     S
VENDOR.CODE     D    1                                          5R     S
VENDOR.REF      D    2                                          10L    S
DEPRECIATION    D    3                                          1L     S
DESCRIPTION     D    4                                          25T    S
COST            D    5              MD2$,                       12R    S
USE.LIFE        D    6                                          5R     S
TAX.LIFE        D    7                                          5R     S
VOLTS           D    8                                          5R     S
PURCHASE.DATE   D    9              D2/                         10R    S
@REVISE         PH     VENDOR.CODE
                       VENDOR.REF
                       DEPRECIATION
                       DESCRIPTION
                       COST USE.LIFE
                       TAX.LIFE
                       VOLTS
                       PURCHASE.DATE
@               PH     ID.SUP
                       EQUIP.CODE
                       DESCRIPTION
                       PURCHASE.DATE

13 records listed.
```

DICT INVENTORY.F file

```
                Type &
Field.......... Field. Field........ Conversion.. Column......... Output Depth &
Name........... Number Definition... Code........ Heading........ Format Assoc..

@ID             D    0                           INVENTORY.F      10L    S
ITEM.CODE       D    0                                            5R     S
TYPE            D    1                                            1L     S
DESCRIPTION     D    2                                            25T    S
QOH             D    3                                            5R     S
COST            D    4              MD2$,                         10R    S
PRICE           D    5              MD2$,                         10R    S
VENDOR.CODE     D    6                                            5R     M ORDER
                                                                         S.ASSOC
ORDER.QTY       D    7                                            5R     M ORDER
                                                                         S.ASSOC
@REVISE         PH     TYPE
                       DESCRIPTION
                       QOH COST
                       PRICE
                       VENDOR.CODE
                       ORDER.QTY
ORDERS.ASSOC    PH     VENDOR.CODE
                       ORDER.QTY

@               PH     ID.SUP
                       ITEM.CODE
                       TYPE
                       DESCRIPTION
                       COST PRICE

12 records listed.
```

DICT LIVESTOCK.F file

```
                Type &
Field.......... Field. Field........ Conversion.. Column..... Output Depth &
Name........... Number Definition... Code........ Heading..... Format Assoc..

@ID             D    0                           LIVESTOCK.F   10L    S
ANIMAL.ID       D    0                                         5R     S
NAME            D    1                                         10T    S
DESCRIPTION     D    2                                         10T    S
USE             D    3                                         1L     S
DOB             D    4              D2/                        10L    S
ORIGIN          D    5                                         12T    S
COST            D    6              MD2$,                      12R    S
EST.LIFE        D    7                                         3R     S
VAC.TYPE        D    8                                         1L     M VAC.A
                                                                      SSOC
VAC.DATE        D    9              D2/                        10R    M VAC.A
                                                                      SSOC
VAC.NEXT        D   10              D2/                        10L    M VAC.A
                                                                      SSOC
VAC.CERT        D   11                                         6L     M VAC.A
                                                                      SSOC
VAC.ASSOC       PH      VAC.TYPE
                        VAC.DATE
                        VAC.NEXT
                        VAC.CERT
@               PH      ID.SUP
                        ANIMAL.ID
                        NAME
                        DESCRIPTION
@REVISE         PH      NAME
                        DESCRIPTION
                        USE DOB
                        ORIGIN COST
                        EST.LIFE
                        VAC.TYPE
                        VAC.DATE
                        VAC.NEXT
                        VAC.CERT

16 records listed.
```

DICT LOCATIONS.F file

Field..... Name.....	Type & Field. Number	Field..... Definition...	Conversion.. Code.....	Column..... Heading.....	Output Format	Depth & Assoc..
@ID	D 0			LOCATIONS.F	10L	S
LOCATION.CODE	D 0				7L	S
DESCRIPTION	D 1				25T	S
NAME	D 2				25T	S
ADR1	D 3				25T	S
ADR2	D 4				25T	S
ADR3	D 5				25T	S
PHONE	D 6				12L	S
FAX	D 7				8L	S
ACRES	D 8				5R	S
SEATS	D 9				5R	S
PARKS	D 10				5R	S
MEDIA.NAME	D 11				15T	M MEDIA .ASSOC
MEDIA.CONTACT	D 12				25T	M MEDIA .ASSOC
MEDIA.PHONE	D 13				12L	M MEDIA .ASSOC
MEDIA.FAX	D 14				8L	M MEDIA .ASSOC
GOV.AGENCY	D 15				25T	M GOV.A SSOC
GOV.CONTACT	D 16				25T	M GOV.A SSOC
GOV.PHONE	D 17				12L	M GOV.A SSOC
GOV.FAX	D 18				8L	M GOV.A SSOC
GOV.FEE	D 19		MD2$,		12R	M GOV.A SSOC
GOV.CHECK	D 20				5L	M GOV.A SSOC
GOV.RATE	D 21		MD3		7R	M GOV.A SSOC
@REVISE	PH	DESCRIPTION NAME ADR1 ADR2 ADR3 PHONE FAX ACRES SEATS PARKS MEDIA.NAME MEDIA.CONTACT MEDIA.PHONE MEDIA.FAX GOV.AGENCY GOV.CONTACT GOV.PHONE GOV.FAX GOV.FEE GOV.CHECK GOV.RATE				
GOV.ASSOC	PH	GOV.AGENCY GOV.CONTACT GOV.PHONE GOV.FAX GOV.FEE GOV.CHECK GOV.RATE				

```
MEDIA.ASSOC     PH      MEDIA.NAME
                        MEDIA.CONTACT
                        MEDIA.PHONE
                        MEDIA.FAX
@               PH      ID.SUP
                        LOCATION.CODE
                        DESCRIPTION
                        PHONE ACRES
                        SEATS

27 records listed.
```

DICT PERSONNEL.F file

```
                 Type &
Field..... Field. Field..... Conversion.. Column..... Output Depth &
Name...... Number Definition... Code..... Heading..... Format Assoc..

@ID            D    0                                PERSONNEL.F    10L    S
BADGE.NO       D    0                                               5R     S
DOB            D    1                D2/                            10R    S
BENEFITS       D    2                                               10T    S
NAME           D    3                                               25T    S
ADR1           D    4                                               25T    S
ADR2           D    5                                               25T    S
ADR3           D    6                                               25T    S
PHONE          D    7                                               12L    S
DEP.NAME       D    8                                               10T    M DEP.A
                                                                            SSOC
DEP.DOB        D    9                D2/                            10R    M DEP.A
                                                                            SSOC
DEP.RELATION   D   10                                               5T     M DEP.A
                                                                            SSOC
EQUIP.CODE     D   11                                               5R     M EQUIP
                                                                            .ASSOC
EQUIP.PAY      D   12                MD2$                           10R    M EQUIP
                                                                            .ASSOC
ACT.NO         D   13                                               5R     M ACTS.
                                                                            ASSOC
ACT.PAY        D   14                MD2$                           10R    M ACTS.
                                                                            ASSOC
RIDE.ID        D   15                                               5R     M RIDES
                                                                            .ASSOC
RIDE.PAY       D   16                MD2$                           10R    M RIDES
                                                                            .ASSOC
EQUIP.ASSOC    PH     EQUIP.CODE
                      EQUIP.PAY
ACTS.ASSOC     PH     ACT.NO
                      ACT.PAY
@              PH     ID.SUP
                      BADGE.NO DOB
                      NAME ADR1
                      ADR2 ADR3
                      PHONE
RIDES.ASSOC    PH     RIDE.ID
                      RIDE.PAY
DEP.ASSOC      PH     DEP.NAME
                      DEP.DOB
                      DEP.RELATION
@REVISE        PH     DOB BENEFITS
                      NAME ADR1
                      ADR2 ADR3
                      PHONE
                      DEP.NAME
                      DEP.DOB
                      DEP.RELATION
                      EQUIP.CODE
                      EQUIP.PAY
                      ACT.NO
                      ACT.PAY
                      RIDE.ID
                      RIDE.PAY

24 records listed.
```

DICT RIDES.F file

Field..... Name.....	Type & Field. Number	Field..... Definition...	Conversion.. Code.....	Column..... Heading.....	Output Format	Depth & Assoc..
@ID	D	0		RIDES.F	10L	S
RIDE.ID	D	0			5R	S
DESCRIPTION	D	1			20T	S
OPERATOR	D	2			5R	M
EQUIP.CODE	D	3			5R	M
ANIMAL.ID	D	4			5R	M
ANIMALS	I	TRANS(LIVESTOCK.F,ANIMAL.ID,NAME,"X")		Animals	10T	M
OP.NAME	I	TRANS(PERSONNEL.F,OPERATOR,NAME,"X")		Operator	25T	M
EQUIPMENT	I	TRANS(EQUIPMENT.F,EQUIP.CODE,DESCRIPTION,"X")		Equipment	25T	M
@	PH	ID.SUP RIDE.ID DESCRIPTION OPERATOR EQUIP.CODE ANIMAL.ID				
@REVISE	PH	DESCRIPTION OPERATOR EQUIP.CODE ANIMAL.ID				

11 records listed.

DICT VENDORS.F file

```
                    Type &
Field.......... Field. Field........ Conversion.. Column......... Output Depth &
Name........... Number Definition... Code........ Heading........ Format Assoc..

@ID               D     0                          VENDORS.F       10L    S
VENDOR.CODE       D     0                                          5R     S
COMPANY           D     1                                          25T    S
ADR1              D     2                                          25T    S
ADR2              D     3                                          25T    S
ADR3              D     4                                          25T    S
TERMS             D     5                                          10T    S
CONTACT           D     6                                          25T    S
PHONE             D     7                                          12L    S
FAX               D     8                                          8L     S
EQUIP.CODE        D     9                                          5R     M
ITEM.CODE         D    10                                          5R     M PROD.
                                                                           ASSOC
LEAD.TIME         D    11                                          5R     M PROD.
                                                                           ASSOC
@REVISE           PH       COMPANY ADR1
                           ADR2 ADR3
                           TERMS CONTACT
                           PHONE FAX
                           EQUIP.CODE
                           ITEM.CODE
                           LEAD.TIME
PROD.ASSOC        PH       ITEM.CODE
                           LEAD.TIME
@                 PH       ID.SUP
                           VENDOR.CODE
                           COMPANY ADR1
                           ADR2 ADR3
                           CONTACT PHONE

16 records listed.
```