Rocket UniData

Using UniQuery

Version 8.2.1

July 2017
UDT-821-UNQU-1

Notices

Edition

Publication date: July 2017
Book number: UDT-821-UNQU-1
Product version: Version 8.2.1

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 to UniQuery

UniQuery is a language that enables you to perform queries against Rocket UniData files and records. Once you have entered data, you can use UniQuery to produce reports using the dictionary records you have defined for a file. UniQuery statements range from very simple to very complex. You can define selection criteria, sorting criteria, formatting options, and specify certain record IDs to display in a UniQuery statement. UniQuery also provides the `MODIFY` command, which enables you to enter or modify data in a UniData file.

UniQuery commands are similar to UniData ECL (Environment Control Language) commands. Like UniData ECL commands, you enter UniQuery commands from the ECL command line. You can also execute UniQuery commands from a UniBasic program, a paragraph, or a Proc. UniData ECL commands are used for file management functions, while UniQuery commands are used for database queries and tasks.

This manual assumes that you have an understanding of the UniData Relational Database Management System, the UniData file structure, file dictionaries, and virtual attributes. For information about these topics, see *Using UniData*.

The UniData demo database

UniData provides a demo database that you can use to test UniQuery features. This database uses a retail store as the foundation for the account. The files relating to the retail store are:

- CLIENTS — Contains customer information for clients of the store.
- INVENTORY — Consists of product records for the store.
- ORDERS — Contains records of customer orders.

Copying demo database files

The UniData demo account normally resides in *udthome*`/demo` on UniData for UNIX or *udthome*`\demo` on UniData for Windows Platforms. The examples in this manual use the demo database files. You may want to consider copying the demo database files to your own account so you can follow the examples without changing the original demo database. You can copy the files from the demo database to your account at the operating system level or from the ECL command line.

Copying files on UniData for UNIX

The following section describes how to copy demo database files to different accounts on UniData for UNIX.

Copying files at the UNIX level

If you are using the Recoverable File System, do not copy the files from the demo database to your account using a UNIX `copy` command while UniData is running. Doing so may cause unpredictable results.

Complete the following steps to copy the demo database file to another account from UNIX:

1. Change directories to the account where you want to copy the demo database files:
   `cd /accountname`

2. Copy the INVENTORY file and the dictionary for the INVENTORY file from the demo database using the UNIX `cp` command:

   ```
   cp -r /demo_directory/INVENTORY /account_name/INVENTORY
   cp /demo_directory/D_INVENTORY /account_name/D_INVENTORY
   ```

   Because the INVENTORY file and the ORDERS file are dynamic files, you must use the -r option with the cp command to copy the entire dynamic file directory. You do not need the -r option for the CLIENTS file or the dictionary files because they are static files.

   Repeat the preceding commands for the CLIENTS file and its dictionary, then the ORDERS file and its dictionary.

3. Enter UniData from your account. You must create a VOC record for each one of the demo database files you have copied in order to access the files in UniData. To create the VOC record, use any valid text editor. In the following example, the VOC records are created using the Alternate Editor (AE). See *Using UniData* for more information about the VOC file.

   ```
   :AE VOC INVENTORY
   Top of New "INVENTORY" in "VOC".
   *--: I
   001= F
   002= INVENTORY
   003= D_INVENTORY
   *--: FI
   Filed "INVENTORY" in file "VOC".
   :
   :AE VOC CLIENTS
   Top of New "CLIENTS" in "VOC".
   *--: I
   001= F
   002= CLIENTS
   003= D_CLIENTS
   *--: FI
   Filed "CLIENTS" in file "VOC".
   :AE VOC ORDERS
   Top of New "ORDERS" in "VOC".
   *--: I
   001= F
   002= ORDERS
   003= D_ORDERS
   *--: FI
   Filed "ORDERS" in file "VOC".
   ```

   The INVENTORY, CLIENTS, and ORDERS files should now be accessible from your account.

Copying files from the ECL command line

Complete the following steps to copy files from the demo database to your account from the ECL command line:

1. Create the INVENTORY file using the `CREATE.FILE` command. Because the INVENTORY file is a recoverable dynamic file, use the DYNAMIC keyword and the RECOVERABLE keyword with the `CREATE.FILE` command, as shown in the following example:

   ```
   :CREATE.FILE INVENTORY DYNAMIC RECOVERABLE
   modulos for file INVENTORY=101
   Create file D_INVENTORY, modulo/1,blocksize/1024
   Hash type = 0
   Create dynamic file INVENTORY, modulo/101,blocksize/1024
   Hash type = 0
   Split/Merge type = KEYDATA
   Added "@ID", the default record for UniData to DICT INVENTORY.
   ```

The ORDERS file is also a recoverable dynamic file. Use the CREATE.FILE command with the DYNAMIC and RECOVERABLE keywords to create the ORDERS file, as shown in the preceding example.

The CLIENTS file is a nonrecoverable static file. Use the CREATE.FILE command with no options to create the CLIENTS file, as shown in the following example:

```
:CREATE.FILE CLIENTS 101
Create file D_CLIENTS, modulo/1,blocksize/1024
Hash type = 0
Create file CLIENTS, modulo/101,blocksize/1024
Hash type = 0
Added "@ID", the default record for UniData to DICT CLIENTS.
:
```

For more information about the CREATE.FILE command, see the *UniData Commands Reference*.

2. Create a pointer to the INVENTORY file in the demo database from the ECL command line using the SETFILE command, as shown in the following example:

```
:SETFILE /usr/ud73/demo/INVENTORY PRACTICE1
Establish the file pointer
Tree name /usr/ud73/demo/INVENTORY
Voc name PRACTICE1
Dictionary name /usr/ud73/demo/D_INVENTORY
Ok to establish pointer(Y/N) = Y
SETFILE completed.
```

For more information about the SETFILE command, see the *UniData Commands Reference*.

3. Copy the records from the INVENTORY file in the demo database to the INVENTORY file in your account using the ECL COPY command with the ALL keyword. After you copy the records from the INVENTORY file, copy the records from the INVENTORY dictionary file in the demo database to the dictionary of the INVENTORY file in your account, as shown in the following example:

```
:COPY FROM PRACTICE1 TO INVENTORY ALL
175 records copied
:
:COPY FROM DICT PRACTICE1 TO DICT INVENTORY ALL
@ID exists in INVENTORY, cannot overwrite
15 records copied
:
```

PRACTICE1 is the name of the VOC record pointer established in step 2. Because you have completed copying the records from the demo database INVENTORY file to your INVENTORY file, you may now delete the VOC record for PRACTICE1. The following example uses the UniData DELETE command to delete the VOC record for PRACTICE1:

```
:DELETE VOC PRACTICE1
'PRACTICE1' deleted.
:
```

For more information about the DELETE command, see the *UniData Commands Reference*.

4. Repeat steps 2 and 3 for the ORDERS file, and the CLIENTS file. The INVENTORY, CLIENTS, and ORDERS file should now be accessible from your account.

Copying files on UniData for Windows platforms

The following section describes how to copy demo database files to different accounts on UniData for Windows platforms.

Copying files at the MS-DOS level

Complete the following steps to copy the demo database file to another account from MS-DOS:

1. Change directories to the account where you want to copy the demo database files:

```
cd \accoutname
```

2. Copy the INVENTORY file and the dictionary for the INVENTORY file from the demo database using the MS-DOS `COPY` command:

```
D:\U2\ud73\demo>COPY \UniData\Demo\INVENTORY
\UniData\Claireg\INVENTORY
D:\ud73\ud73\demo>COPY\UniData\Demo\D_INVENTORY
\UniData\Claireg\D_INVENTORY
```

   Repeat the preceding commands for the CLIENTS file and its dictionary, then the ORDERS file and its dictionary.

3. Enter UniData from your account. You must create a VOC record for each one of the demo database files you have copied in order to access the files in UniData. To create the VOC record, use any valid text editor. In the following example, the VOC records are created using the Alternate Editor (AE). See *Using UniData* for more information about the VOC file.

```
:AE VOC INVENTORY
Top of New "INVENTORY" in "VOC".
*--: I
001= F
002= INVENTORY
003= D_INVENTORY
*--: FI
Filed "INVENTORY" in file "VOC".
:
:AE VOC CLIENTS
Top of New "CLIENTS" in "VOC".
*--: I
001= F
002= CLIENTS
003= D_CLIENTS
*--: FI
Filed "CLIENTS" in file "VOC".
:AE VOC ORDERS
Top of New "ORDERS" in "VOC".
*--: I
001= F
002= ORDERS
003= D_ORDERS
*--: FI
Filed "ORDERS" in file "VOC".
```

   The INVENTORY, CLIENTS, and ORDERS files should now be accessible from your account.

Copying files from the ECL command line

Complete the following steps to copy files from the demo database to your account from the ECL command line:

1. Create the INVENTORY file using the `CREATE.FILE` command. Because the INVENTORY file is a recoverable dynamic file, use the DYNAMIC keyword and the RECOVERABLE keyword with the `CREATE.FILE` command, as shown in the following example:

```
:CREATE.FILE INVENTORY DYNAMIC RECOVERABLE
modulos for file INVENTORY=101
Create file D_INVENTORY, modulo/1,blocksize/1024
Hash type = 0
Create dynamic file INVENTORY, modulo/101,blocksize/1024
Hash type = 0
Split/Merge type = KEYDATA
Added "@ID", the default record for UniData to DICT INVENTORY.
```

   The ORDERS file is also a dynamic file. Use the `CREATE.FILE` command with the DYNAMIC and RECOVERABLE keywords to create the ORDERS file, as shown in the preceding example.

The CLIENTS file is a static file. Use the CREATE.FILE command with no options to create the CLIENTS file, as shown in the following example:

```
:CREATE.FILE CLIENTS 101
Create file D_CLIENTS, modulo/1,blocksize/1024
Hash type = 0
Create file CLIENTS, modulo/101,blocksize/1024
Hash type = 0
Added "@ID", the default record for UniData to DICT CLIENTS.
:
```

For more information about the CREATE.FILE command, see the *UniData Commands Reference*.

2. Create a pointer to the INVENTORY file in the demo database from the ECL command line using the SETFILE command, as shown in the following example:

```
:SETFILE \u2\ud72\Demo\INVENTORY PRACTICE1
Establish the file pointer
Tree name \u2\ud73\Demo\INVENTORY
Voc name PRACTICE1
Dictionary name \u2\ud73\Demo\D_INVENTORY
Ok to establish pointer(Y/N) = Y
SETFILE completed.
:
```

For more information about the SETFILE command, see the *UniData Commands Reference*.

3. Copy the records from the INVENTORY file in the demo database to the INVENTORY file in your account using the ECL COPY command with the ALL keyword. After you copy the records from the INVENTORY file, copy the records from the INVENTORY dictionary file in the demo database to the dictionary of the INVENTORY file in your account, as shown in the following example:

```
:COPY FROM PRACTICE1 TO INVENTORY ALL
175 records copied
:
:COPY FROM DICT PRACTICE1 TO DICT INVENTORY ALL
@ID exists in INVENTORY, cannot overwrite
15 records copied
:
```

PRACTICE1 is the name of the VOC record pointer established in step 2. Because you have completed copying the records from the demo database INVENTORY file to your INVENTORY file, you may now delete the VOC record for PRACTICE1. The following example uses the UniData DELETE command to delete the VOC record for PRACTICE1:

```
:DELETE VOC PRACTICE1
'PRACTICE1' deleted.
:
```

For more information about the DELETE command, see the *UniData Commands Reference*.

4. Repeat steps 2 and 3 for the ORDERS file, and the CLIENTS file. The INVENTORY, CLIENTS, and ORDERS file should now be accessible from your account.

Overview of dictionary files

Every UniData data file has a corresponding dictionary file. A dictionary contains a set of records that define the structure of the records in the data file, called D-type records. A dictionary may also contain phrases, called PH-types, and items which calculate or manipulate data, called virtual fields, or V-types. A user may also define a dictionary item to store user-defined data, called X-types.

Dictionary file

A dictionary file, just like a data file, is a collection of records containing attributes. DICT.DICT is the master dictionary for a dictionary file. It describes each attribute that makes up a dictionary record, and is located in *udthome*/sys on UniData for UNIX or *udthome*\sys on UniData for Windows platforms.

D-type dictionary records

The purpose of a D-type dictionary record is to define the location of an attribute in the data file. Other information, including the conversion code, column display heading, display format, and the value code specifier is also included in the dictionary record.

V-type dictionary records

The result of a virtual attribute is information that does not literally exist in the data portion of a file. Information is calculated or otherwise derived from other attributes, other files, or other information in the database.

In a UniData virtual attribute, you can use functions to manipulate data; you may refer to data in the same file or other related files. Virtual attributes also allow arithmetic, relational, and Boolean operations in combination with conditional expressions, such as IF/THEN/ELSE.

PH-type dictionary records

A PH-type dictionary record describes a phrase. A phrase is a part of a UniQuery statement that does not contain a verb. The most common use of a phrase is to define associations. Phrases are also used for the following:

- @UQ — displays a default set of dictionary records when you issue a LIST or SORT command.
- @LPTR — displays a default set of dictionary records when you issue the LPTR command with no other dictionary items specified.
- Alias — stores a fragment of a UniQuery statement that you frequently use.

X-type dictionary records

An X-type dictionary record stores user-defined information, and is ignored by UniQuery. It can contain any type of information desired. X-type attributes are commonly used to store data that you do not want stored in the data portion of the file, such as the next available sequential number for an @ID in a file.

SQ-type dictionary records

An SQ-type dictionary record is automatically created by UniData when converting dictionary records for SQL/ODBC compliance. You should not access these dictionary records.

Because UniQuery uses attributes defined in dictionaries to create reports, you should have a clear understanding of dictionary concepts. For detailed information about dictionaries, see *Using UniData*.

Chapter 2: The basics of UniQuery

This chapter introduces the syntax for UniQuery statements and discusses the UniQuery commands (verbs) and keywords available for retrieving data from the database and creating UniQuery reports.

Elements of a UniQuery statement

A UniQuery statement in the simplest form consists of a UniQuery command (verb) and an existing file name. Most UniQuery statements also contain keywords, which provide selection criteria, sorting criteria, and formatting options. To make UniQuery statements more readable, UniQuery also provides throwaway keywords. You can use these keywords anywhere in a UniQuery statement.

Syntax

```
command [DICT] filename [display_attributes | ALL] [record_IDs]
[selection_criteria] [sorting_criteria] [format_options]
[control_options]
```

With the exception of *command*, DICT, and *filename*, all other elements of a UniQuery statement can appear in any order.

The following table describes the parameters of a UniQuery statement.

Function	Description
command	Specifies a UniQuery command. You can only use one command in a UniQuery statement.
DICT	Specifies the dictionary portion of *filename*. DICT must immediately follow the command.
filename	Specifies the UniData *filename* on which to perform the operation. Only one *filename* can be specified in a UniQuery statement. *filename* must immediately follow the command, unless you are specifying DICT.
display_attributes | ALL	Specifies the attributes in the dictionary of *filename* to include in the output. The ALL option displays all D-type dictionary attributes.
record_IDs	Specifies the record IDs in *filename* to access.
selection_criteria	Specifies conditions that records must meet before records can be included in the UniQuery statement.
sorting_criteria	Specifies the order to display records.
format_options	Specifies how to format the report, including page breaks, headers and footers.
control_options	Keywords you can specify that control report output.

How UniQuery statements are evaluated

UniQuery always evaluates the first word in a UniQuery statement as a command, and searches the VOC file in the current account to make sure the command exists. If the specified command is not found in the VOC file, UniQuery returns an error message.

If the DICT keyword is not specified in a UniQuery statement, UniQuery evaluates the next word as a file name. Again, UniQuery searches the VOC file for the existence of the file name, and if it is not found

returns a message. If the DICT keyword is specified in the UniQuery statement, UniQuery searches the VOC file for the existence of the record for the specified file name and uses the dictionary file defined in the VOC record for the specified file name.

After verifying that the UniQuery statement contains a valid command and file name, UniQuery processes the remainder of the elements in the UniQuery statement by verifying that display attributes appear in the dictionary of the file, the DICT.DICT file, or the VOC file. All other elements of a UniQuery statement must appear in the VOC file. For information about dictionary files, DICT.DICT and the VOC file, see *Using UniData*.

Limitations of a UniQuery statement

A UniQuery statement has the following limitations:

- Cannot exceed 9247 characters.
- Cannot contain more than 150 attributes.
- Cannot contain more than 20 sort fields (BY...).
- Cannot contain more than 120 `WITH` statements.
- Cannot contain more than 60 `WHEN` statements.
- A `WHEN` statement cannot process more than 10,240 values in one attribute.
- Cannot contain more than 54 arithmetic operators (SUM, AVG, PCT, CALC).
- A `BY.EXP` statement cannot explode more than 10,240 values in one attribute.
- Cannot contain more than 15 `BREAK.ON`/`BREAK.SUP` clauses.
- Cannot contain more than 999 display attributes.
- A header or footer cannot exceed 2120 characters.
- Cannot contain more than 256 virtual attributes.
- Page width cannot exceed 272 characters.

UniQuery commands

A UniQuery command, or verb, is the first word in a UniQuery statement, defining what action to take on the file you specify in the UniQuery statement.

The following table lists the verbs available in UniQuery.

Verb	Description
`BSELECT`	Retrieves specified attribute(s) into a select list.
`COUNT`	Counts the number of records in a file meeting the selection criteria.
`ESEARCH`	Searches a file for a specified string and creates a select list of record IDs that contain the string.
`LIST`	Lists records in a file meeting the selection criteria.
`LIST.ITEM`	Lists the full data record of each record meeting the selection criteria.
`LIST.LABEL`	Displays records meeting the selection criteria in a user-defined label format.
`REFORMAT`	Directs output of records meeting the selection criteria to another UniData file.
`SELECT`	Creates a list of record IDs meeting the selection criteria.

Verb	Description
SORT	Lists records in a file meeting the selection criteria. SORT is the same as list, except SORT lists records in @ID order if no other sorting criteria is specified.
SORT.ITEM	Lists the full data record of each record meeting the selection criteria. SORT.ITEM is the same as LIST.ITEM, except SORT.ITEM lists records in @ID order if no other sorting criteria is specified.
SORT.LABEL	Displays records meeting the selection criteria in a user-defined label format. SORT.LABEL is the same as LIST.LABEL, except SORT.LABEL lists records in @ID order if no other sorting criteria is specified.
SREFORMAT	Directs output of records meeting the selection criteria to another UniData file. SREFORMAT is the same as REFORMAT, except that SREFORMAT copies records in @ID order if no other sorting criteria is specified.
SSELECT	Creates a list of record IDs meeting the selection criteria. SSELECT is the same as SELECT, except SSELECT sorts the records by @ID if no other sorting criteria is specified.
SUM	Sums one or more numeric fields meeting the selection criteria.
USHOW	Creates an interactive list of records meeting the selection criteria and allows the user to scroll forward, scroll backward, and select records to produce a select list.

In the following example, UniQuery displays the NAME attribute in the CLIENTS file using the LIST verb:

```
:LIST CLIENTS NAME
LIST CLIENTS NAME 14:56:10 Apr 04 2011 1
CLIENTS... Name.........................
         9999         Paul Castiglione
         10034        Fredrick Anderson
         9980          Beverly Ostrovich
         10015     Cal di Grigorio
...
```

UniQuery keywords

A UniQuery keyword is a specially designed word that qualifies UniQuery commands. UniQuery provides the following keywords.

Keyword	Description
ALL	Displays every D-type attribute for each record in a file.
ASSOCIATED	Used in conjunction with the WHEN keyword. Returns only those values in associated attributes that together satisfy the selection criteria. Operates on multivalued and multi-subvalued attributes.
AVERAGE	Calculates and prints the average for values of numeric attributes.
BREAK.ON	Groups data and creates a break in a report whenever the value of the break attribute changes.
BREAK.SUP	Groups data and creates a break in a report whenever the value of the break attribute changes, but suppresses the display of the breakpoint value.

Keyword	Description
BY	Sorts specified attribute values in ascending order.
BY.DSND	Sorts specified attribute values in descending order.
BY.EXP	Sorts values of a specified multivalued attribute in ascending order.
BY.EXP.DSND	Sorts values of a specified multivalued attribute in descending order.
CALCULATE	Performs a total calculation based on a formula defined in a virtual attribute.
CNV	Applies a conversion to an attribute during the execution of the UniQuery statement. Overrides the conversion code defined in attribute 3 of the dictionary record for the attribute.
COL.HDG	Displays a specified column header for an attribute during the execution of the UniQuery statement. Overrides the column header defined in attribute four of the dictionary record for the attribute.
COL.HDR.SUPP	Suppresses the display of column headers for each attribute in a UniQuery statement.
COL.SPACES	Specifies the number of spaces separating columns in a UniQuery report.
COUNT.SUP	Suppresses the record count message at the end of the report.
DBL.SPC	Double-spaces records in a UniQuery report.
DET.SUP	Suppresses detail lines and only prints breakpoint lines and totals in a UniQuery report.
DICT	Specifies the dictionary portion of a file rather than the data portion.
EACH	Retrieves only those records where each value in a multivalued or multi-subvalued attribute exactly meets the selection criteria.
EVAL	Enables you to define a virtual attribute expression in a UniQuery statement.
EVERY	Retrieves only those records where each value in a multivalued or multi-subvalued attribute exactly meets the selection criteria.
FIRST	Returns the first *n* records from a file in a UniQuery statement.
FMT	Formats an attribute in the format you designate during the execution of the UniQuery statement. Overrides the format defined in attribute 5 of the dictionary record for the attribute.
FOOTING	Enables you to specify a footer for each page of a UniQuery report.
FROM	Points to a numbered select list with the `SAVE.LIST` command. Use with the ECL `COPY` command to specify the file from which you are copying records.
GRAND.TOTAL	Inserts a text string on the total line of a UniQuery report.
HDR.SUP	Suppresses the default header in a UniQuery report.
HEADING	Allows you to specify a header for each page of a UniQuery report.
ID.SUP	Suppresses display of the record ID in a UniQuery report.
INTERSECT	Retrieves records that have identical values in two different attributes.
IS NULL.VAL	Checks for the presence of the null value.
LPTR	Directs UniQuery output to a printer.
MARGIN	Specifies the width of the left margin in a UniQuery report.
NO	Retrieves records that do not meet the select criteria.
NO.INDEX	Specifies not to use an alternate key index when processing the UniQuery statement.

Keyword	Description
NO.NULLS	Used with the SAVING or AVERAGE keyword. With SAVING, does not save empty strings to a select list. With AVERAGE, does not include empty strings in the calculation.
NOPAGE	Suppresses pauses between pages in a report sent to the terminal.
NO.SPLIT	Prevents records being split across page boundaries.
ONLY	Suppresses the use of the @UQ and @LPTR phrases.
PERCENT	Calculates percentages of numeric attributes in a report.
REQUIRE.INDEX	Forces use of an alternate key index when processing a UniQuery statement.
REQUIRE.SELECT	Verifies that a select list is active before processing a UniQuery statement.
SAMPLE	Returns sample *n* records from a file in a UniQuery statement.
SAMPLED	Returns every *n*th record from a file.
SAVING	Creates a select list of a specified attribute value rather than record IDs.
SELECT.ONLY	Ensures that a select list is active before processing a UniQuery statement.
TO	Directs output to a numbered select list or a text file.
TOTAL	Accumulates and displays totals for numeric attributes in a UniQuery report.
UNIQUE	Used with the SAVING keyword to create a list of unique attribute values rather than record IDs. If a value appears multiple times, it is only saved once.
USING	Uses a dictionary from another file during the execution of a UniQuery statement.
VAL.OF	Returns all of the values of a multi-subvalued attribute when at least one of subvalues meets the selection criteria.
VERTICAL	Displays a UniQuery report in vertical format.
WHEN	Returns only those values of multivalued attributes that satisfy the selection criteria in a UniQuery statement.
WITH	Used to specify selection criteria. Valid with singlevalued, multivalued, and multi-subvalued attributes.
WITHIN	Displays hierarchical relationships between records.

Throwaway keywords

Throwaway keywords are words that are used to make a sentence more readable, and are ignored by UniQuery. You can use throwaway keywords anywhere after the command in a UniQuery statement.

The following throwaway keywords are available:

- A
- ANY
- ARE
- FILE
- FOR

- IN
- OF
- PRINT
- THAN
- THE

Displaying attributes

UniQuery displays attributes in the order they appear in the UniQuery statement. Each attribute must exist in the dictionary of the specified file, in the VOC file, or in the DICT.DICT file. When searching for the attribute, UniQuery first searches the dictionary of the specified file, then the VOC file, then the DICT.DICT file.

The following example lists PROD_NAME and COLOR in the INVENTORY file:

```
:LIST INVENTORY PROD_NAME COLOR
LIST INVENTORY PROD_NAME COLOR 14:04:48 Apr 17 2011 1
Product
INVENTORY. Name..... Color.....
           53050 Photocopie Beige
                         r
           56060 Trackball Gray
           57030 Scanner Gray
           31000 CD System Black
```

If you include the ALL keyword in a UniQuery statement, UniQuery display all D-type dictionary attributes, as shown in the following example:

```
:LIST CLIENTS ALL
LIST CLIENTS ALL 16:07:22 Apr 17 2011 1
CLIENTS                      9999
First Name          Paul
Last Name           Castiglione
Company Name    Chez Paul
Address                 45, reu de Rivoli
City                        Paris
State/Territory
Postal Code             75008
Country                 France
Phone Number Phone Category
        3342425544 Work
        3342664857 Fax
.
.
.
```

Displaying phrases

You can specify the name of an existing phrase in a UniQuery statement to display multiple attributes without having to specify each attribute name in the UniQuery statement. A phrase is a part of a UniQuery statement that does not contain a command.

UniData allows you to define two special PH-type dictionary records, @UQ and @LPTR. These two phrases list attributes that you define when you do not specify any other attributes in a UniQuery statement. For information on creating these types of phrases, see *Using UniData*.

The following example shows the @UQ phrase in the dictionary of the INVENTORY file in the demo database:

```
:AE DICT INVENTORY @UQ
Top of "@UQ" in "DICT INVENTORY", 7 lines, 55 characters.
*--: P
001: PH
002: INV_DATE INV_TIME PROD_NAME FEATURES LINE_ITEMS
```

When you do not specify any attributes in a UniQuery statement, UniData automatically displays the attributes defined in the @UQ phrase, as shown in the following example:

```
:LIST INVENTORY
LIST INVENTORY INV_DATE INV_TIME PROD_NAME FEATURES COLOR PRICE
QTY REORDER DIFF 15:40:17 Aug 17 2011 1
INVENTORY                          53050
Inventory Date                  01/09/1996
Inventory Time                  08:00AM
Product Name                    Photocopier
Features                            Personal    Photocopier
Color       Price       Quantity        Reorder         Difference
Beige           $369.95             785                 50                  735
INVENTORY                       56060
Inventory Date              01/11/1996
Inventory Time              12:00PM
Product Name                Trackball
Features                        Super Deluxe Model
Color       Price       Quantity        Reorder         Difference
Gray            $98.99          494                 70                  424
.
.
.
```

You can create your own PH-type dictionary item to display the attributes of your choice. The following example shows a phrase created in the dictionary of the CLIENTS file called PHONE_ITEMS. This phrase lists the PHONE_NUM and PHONE_TYPE attributes:

```
:AE DICT CLIENTS PHONE_ITEMS
Top of "PHONE_ITEMS" in "DICT CLIENTS", 2 lines, 23 characters.
001: PH
002: PHONE_NUM PHONE_TYPE
Bottom.
*--:
```

To use the preceding phrase in a UniQuery statement, include the name of the phrase in the statement, as shown in the following example:

```
:LIST CLIENTS PHONE_ITEMS
LIST CLIENTS PHONE_NUM PHONE_TYPE 15:47:51 Apr 17 2011 1
CLIENTS... Phone Number.. Phone Category

     9999     3342425544 Work
               3342664857 Fax
    10034     3342482815 Work
               3342481924 Fax
     9980     6124168875 Work
```

```
                6124168879 Fax
```

For more information about defining PH-type dictionary records, see *Using UniData*.

Specifying record IDs

You can include record IDs in a UniQuery statement to produce a report for only those record IDs you specify. You can specify more than one record ID in a UniQuery statement by separating them with spaces. In the following example, record IDs 15001 and 35000 in the INVENTORY file are specified:

```
:LIST INVENTORY 15001 35000 PROD_NAME COLOR
LIST INVENTORY 15001 35000 PROD_NAME COLOR 16:19:06 Apr 17 2011 1
           Product
INVENTORY. Name..... Color.....

    15001 Modem      N/A
    35000 Speaker    Black
                      Charcoal
2 records listed
```

Record IDs are often enclosed in single or double quotation marks for readability, and to avoid conflict with other UniQuery keywords, attribute names, and phrases. If a record ID contains special characters, such as an asterisk, it must be enclosed in quotation marks.

Chapter 3: Selecting and sorting data

This chapter describes how to select data from the database using relational and Boolean operators. This chapter also describes how to sort selected data in a UniQuery report.

Selecting data

When you issue a UniQuery statement without defining specific record IDs or selection criteria, UniQuery executes the statement against the entire data file. Normally, you probably do not want to view all the records in a file, but rather a subset of data. For instance, you might want to create a report for clients that live in certain states, for inventory items of a certain color, for orders that were placed on a certain date, and so forth. You can create reports on subsets of data by using selection criteria.

UniQuery allows you to create subsets of data in the following ways:

- Selecting specific record IDs
- Selecting data which matches specific selection criteria
- Creating and using select lists
- Sampling data

Using WITH and WHEN

UniQuery provides two keywords, `WITH` and `WHEN`, which you can use to specify selection criteria. `WITH` can be used with singlevalued, multivalued, and multi-subvalued attributes, limiting the records that a UniQuery statement selects. `WHEN` is used only with multivalued or multi-subvalued attributes, and limits which values a UniQuery statement displays.

The WITH keyword

The `WITH` keyword returns all attributes that meet the defined selection criteria.

Syntax

...WITH [EVERY] *condition* [AND | OR] [EVERY] *condition* ...

A UniQuery statement can contain multiple `WITH` statements, but no more than 120 `WITH` statements can appear in one UniQuery statement. If two `WITH` statements are joined with the `AND` operator, both conditions must be true. If the statements are joined with the `OR` operator, either condition must be true.

You can specify more than one value in a conditional statement. In the following example, the `WITH` keyword is used to select records from the ORDERS file with the ORD_DATE equal to 01/14/96, 01/15/96, or 01/21/96:

```
:LIST ORDERS WITH ORD_DATE = "01/14/96" OR WITH
ORD_DATE = "01/15/96" OR WITH ORD_DATE = "01/21/96"
ORD_DATE

LIST ORDERS WITH ORD_DATE = "01/14/96" OR WITH
ORD_DATE = "01/15/96" OR WITH ORD_DATE = "01/21/96"
ORD_DATE 15:14:26 Aug 18 1998 1
ORDERS.... Order Date
```

```
965          01/15/96
934          01/14/96
873          01/21/96
966          01/15/96
935          01/14/96
.
.
.
```

If more than one value is specified using `WITH` statements, you can separate the values with spaces if OR is the operator, as shown in the following example:

```
:LIST ORDERS WITH ORD_DATE = "01/14/96" "01/15/96" "01/21/96" ORD_DATE
LIST ORDERS WITH ORD_DATE = "01/14/96" "01/15/96" "01/21/96" ORD_DATE 15:46:18
Aug 18 2011 1
ORDERS.... Order Date

965          01/15/96
934          01/14/96
873          01/21/96
966          01/15/96
935          01/14/96
.
.
.
```

Notice that both previous examples return the same results. This is because `OR` is implied when you enter multiple values; you do not have to reenter `OR WITH`.

If a UniQuery statement contains multiple `WITH` statements not joined with `AND` or `OR`, UniQuery assumes the `AND` operator. In the following example, UniQuery returns all records from the ORDERS file for client number 9968, with an order date prior to 02/15/96:

```
:LIST ORDERS WITH CLIENT_NO = '9968' WITH ORD_DATE < '01/15/96' ORD_DATE
LIST ORDERS WITH CLIENT_NO = '9968' WITH ORD_DATE < '01/15/96' ORD_DATE 16:03:19
 Aug 18 2011 1
ORDERS.... Order Date

901          01/13/96
840          06/20/95
2 records listed
```

The previous UniQuery statement returns the records where both conditions are true. If you use the `OR` operator, UniQuery returns all records where either condition is true.

WITH and multiple values

When UniQuery locates a record that contains at least one value that meets the selection criteria of the `WITH` statement, UniQuery lists all the values for the attribute. In the following example, UniQuery displays records from the ORDERS file that contain the color green:

```
:LIST ORDERS WITH COLOR = "Green" PRODUCT_NO COLOR
LIST ORDERS WITH COLOR = "Green" PRODUCT_NO COLOR 13:22:40 Jul 07 2011 1
ORDERS.... Product Number Color.....

848                57070 N/A
                   39400 Gray
                   11010 Gray
                   40010 Almond
```

```
                    51070 Green
                    35000 Black
950                 51060 Red
                          Green
                    51060 Red
828                 10110 N/A
                    11090 Silver
                    10110 N/A
                    10030 Green
802                 10020 Black
                    40009 Green
                    40012 Black
                    40012 White
                    40002 Green
.
.
 .
```

If you want to limit the output to just those values that contain “Green,” use the WHEN keyword, described in the following section.

The WHEN keyword

The WHEN keyword used in selection criteria compares each value in a multivalued or multi-subvalued attribute against the selection criteria. UniQuery returns only those values that meet the selection criteria.

Syntax

...**WHEN** [ASSOCIATED] [EVERY] *condition* [AND | OR] [EVERY] *condition* ...

A UniQuery statement can contain multiple WHEN statements, but no more than 15 WHEN statements can appear in one UniQuery statement. If two WHEN statements are joined with the AND operator, both conditions must be true. If the statements are joined with the OR operator, either condition must be true.

The WHEN keyword is used with the LIST and SORT verbs. If you use the WHEN keyword with the SELECT verb, you must include the BY.EXP keyword. WHEN limits the *values* within an attribute UniQuery displays, while the WITH keyword limits the *records* selected.

Note: The WHEN keyword is only valid in ECLTYPE U.

Unlike the previous example, UniQuery returns only those values that contain the word “Green” when you use the WHEN keyword:

```
:LIST ORDERS WHEN COLOR = "Green" PRODUCT_NO COLOR
LIST ORDERS WHEN COLOR = "Green" PRODUCT_NO COLOR 14:01:10 Jul 07 2011 1
ORDERS.... Product Number Color.....

848                 51070 Green
950                 51060 Green
828                 10030 Green
802                 40009 Green
                    40002 Green
806                 10030 Green
5 records listed
:
```

Using WHEN ASSOCIATED

The UniQuery ASSOCIATED keyword is used in conjunction with the WHEN keyword, and operates on multivalued and multi-subvalued attributes where an association is defined in the dictionary record of the attribute. The ASSOCIATED keyword returns only those records, and values within those records, that have values in the same position in the multivalued or multi-subvalued attribute matching both specified criteria.

Attributes listed in an ASSOCIATED phrase must have an association defined in attribute seven of the dictionary record. The association definitions must be the same for the associated attributes, and the association phrase record must be defined in the dictionary.

Note: UDT.OPTIONS 94 affects UniQuery statements that use a WHEN clause with two or more associated multivalued or multi-subvalued attributes. In this kind of statement, UDT.OPTIONS 94 ON makes a WHEN clause the same as a WHEN ASSOCIATED clause.

For information on creating associations, see *Using UniData*.

Using relational operators

You can use many relational operators in a UniQuery statement. Relational operators compare values with constants or other variables. Numeric values or constants are compared arithmetically. String values or constants are compared character-by-character from left to right, based on the value of each ASCII character code.

If all characters are identical in two equal-length strings, UniData evaluates the strings as equal. If one of two equal-length strings has a character that appears later in the sequence of ASCII codes, UniData evaluates that string as greater than the other. Because lowercase letters appear later in the sequence of ASCII codes than uppercase letters, UniData evaluates lowercase letters as greater than uppercase letters.

Syntax

command filename [WITH | WHEN] *attribute operator value*

where *attribute operator value* represents selection criteria with a comparison operator and a numeric or string value. The following table lists the valid comparison operators.

Operator	Description
=, EQ, EQUAL	Equal to.
#, NE	Not equal to.
>, GREATER, GT	Greater than.
>=, GE	Greater than or equal to.
<, LESS, LT	Less than.
<=, LE	Less than or equal to.
BETWEEN	Between two specified values.
LIKE	Like a literal or a string.
UNLIKE	Unlike a literal or a string.
MATCH, MATCHES, MATCHING	Matches a pattern.
NOT.MATCH, NOT.MATCHING	Does not match a pattern.

Note: Support and syntax for comparison operators vary by ECLTYPE. To determine ECLTYPE differences, see the individual keyword listing in the *UniQuery Commands Reference*.

In the following example, UniQuery lists records from the ORDERS file with an order date less than January 1, 1996, using the LT relational operator:

```
:LIST ORDERS WITH ORD_DATE LT "01/01/96" ORD_DATE
LIST ORDERS WITH ORD_DATE LT "01/01/96" ORD_DATE 13:47:12 Aug 08 2011 1
ORDERS.... Order Date

841          07/23/95
810          09/01/95
842          08/12/95
811          10/02/95
843          09/01/95
812          09/09/95
844          09/05/95
813          09/02/95
875          10/17/95
845          09/03/95
846          10/02/95
847          10/12/95
```

Using Boolean operators

Boolean operators combine or compare sets of relational data, and produce a result equal to true or false.

The following table describes the Boolean operators.

Operator	Description
AND	Expressions linked by AND must both be true. There must be two expressions.
OR	Either expression linked by OR can be true. There must be two expressions.
NOT	Returns records based on a UniQuery statement that specifies which records to exclude.

When evaluating Boolean expressions, UniData stops evaluating if the first expression is false and the Boolean operator is AND. When the Boolean operator is OR, UniData evaluates both expressions, since only one of the expressions has to be true.

The AND and OR operators have equal precedence in a UniQuery statement if the hierarchy is not specified using parentheses. Without parentheses, UniQuery evaluates the statement from left to right.

In the following example, UniQuery retrieves records where both phrase1 (PROD_NAME = "Camera") and phrase2 (INV_DATE LT "01/01/96") are true, using the AND operator:

```
:LIST INVENTORY WITH PROD_NAME = "Camera" AND WITH INV_DATE LT
"01/01/96" PROD_NAME INV_DATE
LIST INVENTORY WITH PROD_NAME = "Camera" AND WITH INV_DATE LT
"01/01/96" PROD_NAME INV_DATE 14:01:39 Apr 08 2011 1
           Product    Inventory
INVENTORY. Name...... Date......
```

```
    10140 Camera     08/26/1994
    10150 Camera     08/25/1995
    10100 Camera     07/11/1995
    10070 Camera     06/18/1995
    10060 Camera     06/15/1995
    10110 Camera     07/13/1995
    10050 Camera     06/15/1995
    10130 Camera     08/09/1995
    10090 Camera     07/10/1995
    10080 Camera     07/03/1995
    10120 Camera     07/20/1995
11 records listed
:
```

In the next example, UniQuery returns records with either phrase1 (`PROD_NAME = "Camera"`) or phrase2 (`INV_DATE LT "01/01/96"`) is true, using the `OR` operator:

```
:LIST INVENTORY WITH PROD_NAME = "Camera" OR WITH INV_DATE LT
"01/01/96" PROD_NAME INV_DATE
LIST INVENTORY WITH PROD_NAME = "Camera" OR WITH INV_DATE LT
"01/01/96" PROD_NAME INV_DATE 14:04:20 Apr 08 2011 1
           Product    Inventory
INVENTORY. Name...... Date......

    57030 Scanner    12/15/1995
    31000 CD System  06/08/1995
           2
    10140 Camera     08/26/1994
    11001 Computer   06/25/1995
    10150 Camera     08/25/1995
    10100 Camera     07/11/1995
    55040 Cable      12/13/1995
    11020 TV         06/01/1995
    10070 Camera     06/18/1995
    55050 Cable      11/11/1995
    39300 Cassette   08/29/1995
           System
.
.
.
```

You can execute more complex UniQuery statements that contain both the `AND` and `OR` operators. In the following example, UniQuery retrieves records where both phrase1 and phrase2 are true, or phrase3 is true:

```
:LIST INVENTORY WITH PROD_NAME = "TV" AND INV_DATE LT "01/01/96"
OR WITH FEATURES LIKE "...Portable..." PROD_NAME INV_DATE FEATURES
LIST INVENTORY WITH PROD_NAME = "TV" AND INV_DATE LT "01/01/96"
OR WITH FEATURES LIKE "...Portable..." PROD_NAME INV_DATE FEATURES
11:09:33 Apr 25 2011 1
           Product    Inventory
INVENTORY. Name...... Date...... Features.....................

52070      Printer    01/23/1996 Portable Color, 3 ppm
11070      TV         06/21/1995 2" Color, Portable
11080      TV         07/13/1995 9" Color, Portable, Remote
                                 Control
12006      Printer    08/22/1995 Portable Color Ink Jet
11010      TV         06/20/1995 31" Color High Resolution
30000      CD System  06/01/1995 Portable Clock Radio
           1
```

.
.
.

In the previous example, UniQuery evaluates the statements in the following ways:

- Retrieves all records with PROD_NAME equal to TV AND INV_DATE less than 01/01/96. These records meet both the selection criteria in the phrase1 (PROD_NAME = "TV") and the selection criteria in phrase2 (INV_DATE LT "01/01/96").
- Retrieves all records that meet the selection criteria in phrase3 (FEATURES LIKE "...Portable..."), which is joined to phrase2 by the OR operator.

By using parentheses, you can alter the order in which UniQuery processes a statement that contains AND and OR. UniQuery still processes the statement from left to right, but evaluates conditions within parentheses as a group.

In the following example, UniQuery evaluates the same UniQuery statement used in the preceding example in a different way because parentheses are added to change the order of processing the statement:

```
:LIST INVENTORY WITH PROD_NAME = "TV" AND (INV_DATE LT "01/01/96"
OR FEATURES LIKE "...Portable...") PROD_NAME INV_DATE FEATURES
LIST INVENTORY WITH PROD_NAME = "TV" AND (INV_DATE LT "01/01/96"
OR FEATURES LIKE "...Portable...") PROD_NAME INV_DATE FEATURES
11:30:57 Apr 25 2011 1
           Product    Inventory
INVENTORY. Name...... Date...... Features.....................

11070      TV         06/21/1995 2" Color, Portable
11080      TV         07/13/1995 9" Color, Portable, Remote
                                 Control
11010      TV         06/20/1995 31" Color High Resolution
11020      TV         06/01/1995 32" Color With Tri-lingual
                                 Screen
11030      TV         06/23/1995 27" Color, Stereo, Remote
11040      TV         06/04/1995 19" Color, Remote Control
11050      TV         06/27/1995 13" Color, Remote Control
11060      TV         07/03/1995 5" B/W Portable
8 records listed
```

In the previous example, every record must first meet the selection criteria in phrase1 (PROD_NAME = "TV"). Then, UniQuery evaluates this subset of records and retrieves only those that meet the selection criteria in phrase2 (INV_DATE LT "01/01/96") or that meet the selection criteria in phrase3 (FEATURES LIKE "...Portable..."). The records UniQuery selects differ from the previous example because phrase2 and phrase3 are evaluated as a group by adding parentheses, and only one of the conditions has to be true to satisfy the selection criteria.

Using sorting criteria in a UniQuery statement

Sorting criteria specifies how to order the data retrieved from a UniQuery statement. You can sort data using the SORT or SSELECT commands, or by specifying sorting keywords in a UniQuery statement.

The UniQuery SORT and SSELECT commands select, sort, and display data from the database by record ID, if no other sorting criteria is defined. If you define sorting criteria, UniQuery sorts the records accordingly, with the final sort level the record ID.

Syntax

```
command [DICT] filename [record_IDs] [selection_criteria]
[sorting_criteria]...
```

The following keywords are used in sorting criteria.

Keyword	Description
BY	Sorts in ascending order of the specified attribute.
BY.DSND	Sorts in descending order of the specified attribute.
BY.EXP	Sorts in ascending order of the specified multivalued or multi-subvalued attribute. UniQuery sorts each value separately, producing a separate line for each value.
BY.EXP.DSND	Sorts in descending order of the specified multivalued or multi-subvalued attribute. UniQuery sorts each value separately, producing a separate line for each value.

The BY and BY.DSND keywords sort only the first value of a multivalued or multi-subvalued attribute. BY.EXP and BY.EXP.DSND sort all values of a multivalued or multi-subvalued attribute.

Several factors can affect the way UniData sorts data, including:

- The null value — The null value is always the smallest value. Arithmetic operations that involve the null value always result in the null value.
- ECLTYPE — Controls whether UniData interprets a query based on the UniData parser or the Pick parser.
- SORT.TYPE — Controls the sorting algorithm that UniData uses.
- COLLATIONS file — Defines the default sorting order for UniQuery reports.
- The setting of UDT.OPTIONS 69 — Determines how alphanumeric data is sorted when the dictionary item specifies a right-justified sort.

For more information about ECLTYPE and SORT.TYPE, see *Using UniData*. For more information about UDT.OPTIONS, see the *UDT.OPTIONS Commands Reference*.

Whether UniQuery sorts an attribute using left justification or right justification is determined by the format defined in attribute five of the dictionary record for the attribute.

You can define multiple sorting criteria in a UniQuery statement. Sorting criteria is evaluated left to right.

In the following example, records from the CLIENTS file are sorted by state. Notice that you must specify the STATE attribute both in the sorting criteria and as a display attribute:

```
:LIST CLIENTS WITH COUNTRY = "USA" BY STATE NAME STATE
LIST CLIENTS WITH COUNTRY = "USA" BY STATE NAME STATE 15:29:37 Apr 08 2011 1
CLIENTS... Name..................... State/Territory

10094      Steve Barry                AR
10000      Andre Halligan             AZ
10086      Al Elliott                 AZ
9982       Marc Willette              CA
9986       Sam Gunter                 CA
10018      Mary Johnson               CO
10047      Ray Parker                 CO
9987       Glen Asakawa               CO
10063      Jan Elliott                CT
10019      Sally Jackson              FL
.
.
```

```
.
```

In the next example, the UniQuery statement specifies to sort records from the INVENTORY file by COLOR:

```
:LIST INVENTORY BY COLOR PROD_NAME COLOR
LIST INVENTORY BY COLOR PROD_NAME COLOR 15:32:45 Apr 08 2011 1
           Product
INVENTORY. Name...... Color.....

    40003 Telephone  Almond
                      Black
    40001 Telephone  Almond
                      Cranberry
    40015 Telephone  Almond
                      Cranberry
    40004 Telephone  Almond
                      Gray
    39100 CD System  Almond
           13
                      White
    40014 Telephone  Almond
                      White
                      Black
    53030 Photocopie Beige
           r
    53040 Photocopie Beige
           r
.
.
.
```

In the previous example, UniQuery sorts the records based upon the first value of COLOR. Because COLOR is multivalued field, the `BY.EXP` keyword should be used to produce a proper sort, as shown in the following example:

```
:LIST INVENTORY BY.EXP COLOR PROD_NAME COLOR
LIST INVENTORY BY.EXP COLOR PROD_NAME COLOR 16:00:23 Apr 08 2011 1
           Product
INVENTORY. Name...... Color.....

    39100 CD System  Almond
           13
    40001 Telephone  Almond
    40003 Telephone  Almond
    40004 Telephone  Almond
    40005 Telephone  Almond
    40010 Telephone  Almond
    40012 Telephone  Almond
    40014 Telephone  Almond
    40015 Telephone  Almond
    51020 Telephone  Beige
    53030 Photocopie Beige
           r
    53040 Photocopie Beige
           r
    53050 Photocopie Beige
.
.
.
```

In the previous example, each multi-subvalue is “exploded” and treated as a singlevalued field in the report.

Chapter 4: Customizing output

UniQuery provides keywords to customize report output, enabling you to specify column heading names, spacing, width, justification, conversion codes, break levels, margins, footers, and other aspects of formatting.

In addition, UniData provides UDT.OPTIONS and ECLTYPE, which alter the behavior of certain commands.

The following table lists the UniQuery keywords that affect report output.

Report formatting keywords		
ALL	BREAK.ON	BREAK.SUP
CNV	COL.HDR	COL.HDR.SUPP
COL.SPCS	COL.SUP	COUNT.SUP
DBL.SPC	DET.SUPP	EVAL
FOOTING	FMT	HDR.SUPP
HEADING	ID.SUPP	MARGIN
NO.SPLIT	ONLY	VERTICAL

ECLTYPE

In certain circumstances, some UniQuery commands, keywords, and functions behave differently based on the kind of parser you use with UniData. The kind of parser UniData uses is determined by the ECLTYPE. If the behavior of a command is different among the types, the UniData documentation displays the following icon, and details the difference in behavior:

Note: ECLTYPE is U or P. When it is U, UniData interprets commands and keywords consistent with the UniData parser. When it is P, UniData interprets commands and keywords consistent with the Pick parser.

UDT.OPTIONS

A UDT.OPTION is a toggle, or switch, that alters UniData's personality. This section discusses the commands and functions that behave differently in UniQuery, depending on whether a UDT.OPTION is on or off. For example, if UDT.OPTIONS 2 (U_PSTYLEECL) is on, UniData evaluates commands and keywords consistent with the Pick parser rather than the UniData parser.

UniData has default settings for all UDT.OPTIONS. To view the current setting of each option, enter the ECL command `UDT.OPTIONS` at the UniData colon (:) prompt, as shown in the following example:

```
:UDT.OPTIONS
1   U_NULLTOZERO       OFF
2   U_PSTYLEECL        OFF
3   U_SHLNOPAGE        OFF
4   U_MONTHUPCASE      OFF
5   U_USTYLEPRT        OFF
6   U_NOPROCCHAIN      OFF
7   U_NOMAKEPAGE       OFF
8   U_PASSSYSCODE      OFF
9   U_PTROFFSTK        OFF
10   U_TRIMNBR          OFF
```

```
11  U_DATACOMMAND     OFF
12  U_PRIMEDATAQ      OFF
13  U_MCDMDOCONV      OFF
14  U_BASICABORT      OFF
15  U_DYNAMICNUL      OFF
16  U_PRIMEDELETE     OFF
17  U_IGNORE_DOTS     OFF
18  U_NO_DISPDATA     OFF
19  U_VERIFY_VKEY     ON
20  U_IGNLGN_LGTO     ON
21  U_LIST_FPAUSE     OFF
22  U_FMT_COMP        OFF
23  U_PK_READNEXT     OFF
24  U_HUSH_DIVBYZERO  OFF
25  U_PK_BREAKON_L    OFF
26  U_CHK_UDT_DIR     OFF
27  U_DATACOMMAND1    OFF
28  U_BK_VHEAD_SUP    OFF
29  U_DW_SUNDAY7      OFF
30  U_BK_VLINE_SUP    OFF
31  U_VLINE_FMT       OFF
32  U_PI_PRINT_AT     OFF
33  U_RAW_DATA        OFF
34  U_HEADING_DATE    OFF
35  U_EXEC_LOCK       OFF
36  U_QPRINT_ON       OFF
37  U_MENUPAUSE       OFF
38  U_BREAKTOECL      OFF
39  U_CNAME_ALL       OFF
40  U_NOEXECCHAIN     OFF
41  U_UDT_SERVER      OFF
42  U_CHECKREMOTE     OFF
43  U_PRM_DETSUP       OFF
44  U_ERR_JRNL_SUS    OFF
45  U_PROMPTDATA      OFF
46  U_UNFLUSHDATA     ON
47  U_PCT_ROUND_SUP     OFF
48  U_UNBOUNDARY     OFF
49  U_LINEFEED_AT80     OFF
50  U_ULTIMATE_TLOAD     OFF
51  U_ALT_DATEFORMAT     OFF
52  U_KP_DIRFILEPERM     OFF
53  U_PMOD_THROWAWAY     OFF
54  U_PROC_KPTSELECT     OFF
55  U_SUPP_NOIDMSG       OFF
56  U_CONV_BADRETURN     OFF
57  U_USE_POUND       OFF
58  U_USE_COLON       OFF
59  U_NONULL_FIELDS   OFF
60  U_NODFLT_DATE     OFF
61  U_BNULLTOZERO     OFF
62  U_NEG_XDCONV      OFF
63  U_MDNP_ALLEXTL    OFF
64  U_BASIC_FINISH    OFF
65  U_LEN_BELL        OFF
66  U_PICK_NUMERIC_FILES        OFF
67  U_SPECIAL_CHAR    OFF
68  U_USER_EXITS      OFF
69  U_PICK_NCMP       OFF
70  U_PICK_DYNAMIC    OFF
71  U_ULTI_READNEXT   OFF
72  U_ULTI_SEMAPHORE  OFF
```

```
 73  U_PRIME_VERTFORM OFF
 74  U_PHANTOM_LOGOUT   OFF
 75  U_PROC_DELIMITER   OFF
 76  U_VF_ON_RAWDATA_POST_BYEXP OFF
 77  U_PROMPT_QUIT_RETURN OFF
 78  U_PICK_LOCK       OFF
 79  U_PRIME_BREAK_P       OFF
 80  U_PRIME_NOSPLIT       OFF
 81  U_PRIME_NULL_KEY       OFF
 82  U_ICONV_DIGIT_DATE       OFF
 83  U_INPUT_ESC              OFF
 84  U_DISPLAY_HOLD_NAME      OFF
 85  U_NUMERIC_SORT      OFF
 86  U_SCMD_FORADDS      OFF
 87  U_REMOTE_DELETE     OFF
 88  U_CALLC_CDECL     OFF
 89  U_PICKSTYLE_MVSORT     OFF
 90  U_MESSAGE_RAW     OFF
 91  U_LIST_TO_CONV     OFF
 92  U_INSENSITIVE_MATCH     OFF
 93  U_LEVEL_PROCBUFF     OFF
 94  U_PRIME_LIKE     OFF
 95  U_NO_TRANSLATE_NEWLINE     OFF
 96  U_PQN_LINK_RETURN    OFF
 97  U_CORRECT_PLINE    OFF
 98  U_BREAK_LINE_VALUE    OFF
 99  U_GLOBAL_ECHO    OFF
100  U_LINE_COUNTER    OFF
101  U_ALLSPACE_INPUTAT OFF
102  U_ONE_PROCREAD OFF
103  U_INPUT_TAB OFF
104  U_TRAIL_FM_TLOAD OFF
105  U_EXECUTE_ONABORT OFF
106  U_PQN_REFERENCE OFF
107  U_TRANS_MULTIVALUE OFF
108  U_PICK_REPORT OFF
109  U_TELNET_NODELAY OFF
110  U_OCONV_EMPTY_STR OFF
111  U_NT_CTRL_C_IGNORE OFF
112  U_DO_UNLINK OFF
113  U_SPOOL_BINARY OFF
114  U_NOFORMFEED OFF
115  U_MIXED_LOCATE OFF
116  U_WINDOWS_SPOOL64 OFF
117  U_BLOCK_ECL_INLINE_PROMPT OFF
118  U_NO_STACKED_EXECUTE OFF
:
```

If you want to change the value of an option, use the `UDT.OPTIONS` ECL command as shown below:

Syntax:

`UDT.OPTIONS` [*n* {ON | OFF}]

n is the number of the option you want to change.

When you exit UniData, all UDT.OPTIONS return to the default settings.

If you want certain options in effect for users when they enter a UniData account, set the desired UDT.OPTION in the login paragraph for each user.

When a UDT.OPTION affects the behavior of a command, you will see the UDT.OPTIONS icon following the description of the command in UniData manuals.

The following table defines those UDT.OPTIONS that affect UniQuery commands.

UDT.OPTIONS	Command behavior
1 U_NULLTOZERO	Determines how UniQuery evaluates empty strings. **ON** — Empty strings (“ ”) and zero are equivalent. **OFF** — Empty strings (“ ”) and zero are not equivalent.
2 U_PSTYLEECL	Determines which parser to use when interpreting UniQuery commands. **ON** — UniQuery evaluates commands using the Pick parser. **OFF** — UniQuery evaluates commands using the UniData parser.
3 U_SHLNOPAGE	Determines if UniQuery pauses at every page when the `LIST` or `SORT` command are executed against an active select list containing record IDS that do not exist in the file. **ON** — UniQuery does not pause at every page. **OFF** — UniQuery pauses at every page.
4 U_MONTHUPCASE	Determines whether UniQuery displays the month in all uppercase or in initial caps in date conversions. **ON** — Month is displayed in all uppercase. **OFF** — Month is displayed in initial caps.
7 U_NOMAKEPAGE	Determines how UniQuery fills an incomplete page when printing a report. **ON** — UniQuery performs a page feed or returns to the colon prompt after the last line of data instead of adding filler line feeds. **OFF** — UniQuery adds enough line feeds before it returns to the colon prompt or performs a page feed.
13 U_MCDMDOCONV	Determines how UniQuery performs an MD (masked decimal) conversion when a decimal point is present in the data. **ON** — UniQuery returns the original data without performing a conversion. **OFF** — UniQuery converts the data according to the conversion code, and rounds the result.
21 U_LIST_FPAUSE	Determines whether you need to enter a carriage return at the end of report displayed to the terminal screen to return to the colon prompt. **ON** — UniQuery positions the cursor at the last line of the terminal and waits for a carriage return before returning to the colon prompt. **OFF** — UniQuery positions the cursor at the ECL prompt.
22 U_FMT_COMP	Determines whether UniQuery `WITH` and `WHEN` comparisons use the numeric value or the string value of data. **ON** — Under certain conditions, uses the string value of the data. **OFF** — Uses standard comparisons.
24 U_HUSH_DIVBYZERO	Controls the display of error messages when an arithmetic error is encountered in a virtual attribute. **ON** — Does not display arithmetic error conditions. **OFF** — Displays arithmetic error conditions.

UDT.OPTIONS	Command behavior
25 U_PK_BREAKON_L	Determines how UniQuery reports print when `BREAK.ON` is used with the 'L' option and the `DET.SUP` keyword. **ON** — Overrides the 'L' option and prints the break line text. **OFF** — Suppresses the break line text and inserts a blank line every time the break value changes.
28 U_BK_VHEAD_SUP	Determines how a UniQuery report with a `BREAK.ON` clause and vertical output displays the breakpoint values. **ON** — Break section of the report displays only the value producing the breakpoint. **OFF** — Break section of the report displays the break section as well as all column headers specified in the UniQuery statement.
29 U_DW_SUNDAY7	Determines how UniQuery converts dates with the DW (date–weekday) conversion code, which converts weekdays to integers. **ON** — Converts Monday through Saturday to integers 1 through 6, respectively. Sunday converts to 7. **OFF** — Converts Monday through Saturday to integers 1 through 6, respectively. Sunday converts to 0.
30 U_BK_VLINE_SUP	Determines if UniQuery displays "start to break" and "finish breaking" when a report containing a `BREAK.ON` clause prints vertically. **ON** — Does not print either breakpoint message. **OFF** — Displays both messages.
31 U_VLINE_FMT	Determines how UniQuery formats a report when the dictionary item for an attribute has a vertical T (text) display format. **ON** — UniQuery formats output according the dictionary display format. **OFF** — UniQuery overrides the dictionary display format and prints the output on one line up to the width of the screen.
34 U_HEADING_DATE	Determines the format of the system date used in `HEADING` and `FOOTING` statements when you use the 'D' option in the header or footer, and whether you execute the `DATE.FORMAT` command during the UniData session. **ON** — Formats dates in `HEADING` and `FOOTING` statements in alphanumerics. **OFF** — Formats dates in numerics with separators in `HEADING` and `FOOTING` statements.
43 U_PRM_DETSUP	Determines how detail lines are displayed in a UniQuery statement when you use the `DET.SUP` keyword. **ON** — Displays the breakpoint value and the detail of the last value accessed before the breakpoint. **OFF** — Does not display the breakpoint detail for the last value accessed.

UDT.OPTIONS	Command behavior
47 U_PCT_ROUND_SUP	Determines how UniQuery calculates percentages for breakpoints and total lines. **ON** — Calculates breakpoint and total line percentages before rounding detail lines. **OFF** — Calculates breakpoint and total line percentages after rounding detail lines.
48 U_UNBOUNDARY	Allows you to print right-justified data beyond the format specified in the dictionary record for an attribute. **ON** — Prints right-justified data as far to the left as necessary to display the entire attribute. **OFF** — Prints right-justified data within the format length defined by the dictionary item for an attribute.
49 U_LINEFEED_AT80	Determines when UniData wraps text to a new line. **ON** — Inserts a line feed at the end of a line of 80 characters. **OFF** — Defaults to the terminal line length to wrap text.
51 U_ALT_DATEFORMAT	Determines how UniQuery displays a date when you use the ECL `DATE.FORMAT` command. **ON** — Displays dates in European format. **OFF** — Displays dates in United States format.
53 U_PMOD_THROWAWAY	Determines how UniQuery treats throwaway keywords in a non-English language. **ON** — Searches the VOC file for the keyword, allowing you to customize the VOC file for a non-English language. **OFF** — Does not search the VOC file for the keyword. If the keyword does not exist in the internal parser table, UniQuery does not recognize the keyword.
55 U_SUPP_NOIDMSG	Determines whether a message indicating UniQuery encountered nonexistent records is printed. **ON** — Does not display a message. **OFF** — Displays the nonexistent record IDs and a message.
59 U_NONULL_FIELDS	Determines if UniQuery saves attributes containing an empty string when using the `BSELECT` command. **ON** — Does not create a blank line for an empty string. **OFF** — Creates a blank line for an empty string.
69 U_PICK_NCMP	Determines how UniQuery sorts alphanumeric data when the dictionary item specifies a right-justified sort. **ON** — Useful in sort type 1 and 2. Uses a comparison algorithm consistent with the way the data is sorted, regardless of `SORT.TYPE`. **OFF** — UniQuery uses an algorithm when selecting records that returns data consistent with the order it is sorted in when `SORT.TYPE` is 0.

UDT.OPTIONS	Command behavior
73 U_PRIME_VERTFORM	Changes the display for a vertical form when the display name defined in the dictionary record for an attribute exceeds the assigned format for the attribute, and the format is right-justified. **ON** — Honors the format defined in the dictionary record of the attribute. **OFF** — Widens the display column by inserting leading spaces up to the number of characters in the display name.
76 U_VF_ON_RAWDATA_ POST_BYEXP	Controls how UniQuery treats a `BY.EXP` clause in a statement when a virtual attribute does not contain an association. **ON** — Calculates the virtual attribute according to the raw data, then extracts the values and subvalues according the `BY.EXP` clause. **OFF** — Calculates the virtual attribute after extracting the values and subvalues. Ignores the `BY.EXP` clause.
79 U_PRIME_BREAK_P	Determines how UniQuery groups data when using the `BREAK.ON` 'P' option. **ON** — Break levels stay together at the end of the report for a group. **OFF** — Two break levels may be separated for a group.
80 U_PRIME_NOSPLIT	Determines how UniQuery groups data when using the `NO.SPLIT` keyword. **ON** — Keeps two break levels together on the same page. **OFF** — May separate two break levels across a page.
89 U_PICKSTYLE_MVSORT	Determines if UniQuery maintains the relative positions of multivalues and multi-subvalues when using a `BY.EXP` clause. **ON** — UniQuery maintains the original position of the value or subvalue in the `BY.EXP` sort. **OFF** — UniQuery does not maintain the original position of the value or subvalue in the `BY.EXP` sort.
91 U_LIST_TO_CONV	Determines if UniQuery performs a conversion on a date field before saving the query to an ASCII file. **ON** — UniQuery performs the conversion defined in the dictionary record for an attribute before it saves the ASCII file. **OFF** — UniQuery does not convert a date field prior to saving the ASCII file.
92 U_INSENSITIVE_MATCH	Determines how UniQuery evaluates data whose dictionary records contain a conversion code of MCL, MCT, or MCU when using the `LIKE` keyword. **ON** — UniQuery ignores the case of the characters when performing the LIKE comparison. **OFF** — UniQuery considers the case of the characters when performing the LIKE comparison.
94 U_PRIME_LIKE	Determines if a `WHEN` clause in a UniQuery statement is treated as `WHEN` or `WHEN ASSOCIATED`. **ON** — `WHEN` functions as `WHEN ASSOCIATED` if dictionary records contain an association. **OFF** — UniQuery treats `WHEN` and `WHEN ASSOCIATED` differently.

UDT.OPTIONS	Command behavior
98 U_BREAK_LINE_VALUE	Determines if the breakpoint value prints on the subtotal line at each breakpoint. **ON** — The breakpoint value does not print on the subtotal line, only on the total line at each breakpoint. **OFF** — The breakpoint value prints on both the subtotal line and total line at each breakpoint.
107 U_TRANS_MULTIVALUE	Determines which multivalue UniData returns when executing a virtual attribute that contains a `TRANS` function with the *n* option. **ON** — Returns the correct multivalue from the target file. **OFF** — Returns the incorrect multivalue from the target file.
108 U_PICK_REPORT	Provides a technique for defining a virtual attribute that displays the number of records at each breakpoint without displaying detail lines. ON — With ECLTYPE P, you can display the number of records at each breakpoint without display detail lines. Does not work in ECLTYPE U. OFF — Virtual attribute does not display breakpoint subtotals and totals.
UDT.OPTIONS 114 U_NOFORMFEED	Determines if a form feed is included on the first page of a UniQuery report. ON UniData suppresses the form feed on the first page of a UniQuery report. OFF UniData includes the form feed on the first page of a UniQuery report.
UDT.OPTIONS 117 U_BLOCK_ECL_INLINE_PROMPT	**ON** ECL inline prompting is prohibited. **OFF** ECL inline prompting is allowed.

Grouping data

UniQuery provides the `BREAK.ON` and `BREAK.SUP` keywords to group data in a report.

Using BREAK.ON

The UniQuery `BREAK.ON` keyword is used with the `LIST` and `SORT` commands to display attributes and create a break in a report when the value of an attribute changes. UniQuery can display subtotals, averages, percentages, and other calculated values at each breakpoint. You can specify multiple breakpoints in one UniQuery statement.

Syntax

```
...BY attribute BREAK.ON ["['option' ...][text]"] attribute
```

When you use `BREAK.ON`, UniQuery, by default, displays a line of asterisks (***), equal to the length of *attribute*, following the last record before the breakpoint. The value of the last breakpoint prints below the asterisks. If you use the `TOTAL`, `AVERAGE`, `PERCENT`, or `CALCULATE` keyword in conjunction with `BREAK.ON`, a row of dashes (---) displays on the same line as the asterisks to indicate a subtotal. You can substitute text of your choice for the line of asterisks by using the *text* option.

Although not required, BREAK.ON is almost always used in conjunction with sorting criteria. The BY, BY.DSND, BY.EXP, and BY.EXP.DSND keywords produce a report in a sorted order. BREAK.ON creates a separation in the display of the data.

The following table describes the *option* parameters available with the BREAK.ON keyword.

Parameter	Description
B	Used in conjunction with the HEADING or FOOTING keywords. When you specify the "'B'" option in the heading or footing definition, UniQuery takes the value of the attribute name at the breakpoint and inserts it at the designated position in the heading or footing definition. If you specify more than one BREAK.ON "'B'" option, UniQuery only uses the first one it encounters in the statement. Each time the breakpoint value changes, UniQuery creates a new page.
D	Suppresses the breakpoint line if only one line of detail is present when the attribute value changes.
L	Suppresses the display of the line of asterisks at the breakpoint, but generates one line feed when an attribute value changes.
O	Only displays the breakpoint value once in the detail portion of the report.
P or ^	Creates a new page for each breakpoint value.
V	Inserts the value of the breakpoint attribute on the breakpoint line. The breakpoint value is printed in the breakline at the V position.
T	Displays a line of dashes (---) before the subtotal when you use more than one BREAK.ON with the TOTAL keywords and the DET.SUP keyword.
text	Specifies a text string to display at the breakpoint, replacing display of the default asterisk string.

Note: The syntax for BREAK.ON varies between ECLTYPE U and ECLTYPE P, and many UDT.OPTIONS affect the BREAK.ON keyword. For detailed information, see the *UniQuery Commands Reference*.

In the following example, information is listed from the INVENTORY file. Records are grouped by product name and color, with a breakpoint on color:

```
:LIST INVENTORY BY PROD_NAME BY.EXP COLOR PROD_NAME BREAK.ON COLOR
LIST INVENTORY BY PROD_NAME BY.EXP COLOR PROD_NAME BREAK.ON COLOR
10:07:55 Apr 13 2011 1
           Product
INVENTORY. Name..... Color.....

    10007 Adapter    N/A
    13001 Adapter    N/A
    13002 Adapter    N/A
                      **********
                      N/A

    39400 CD Player  Black
                      **********
                      Black

    39400 CD Player  Gray
    39500 CD Player  Gray
                      **********
                      Gray

.
```

```
.
.
```

You can create subtotals in a report using the TOTAL and BREAK.ON keywords. In the following example, UniQuery creates subtotals and a grand total for the QTY attribute using the TOTAL keyword:

```
:LIST INVENTORY BY PROD_NAME BY.EXP COLOR PROD_NAME BREAK.ON COLOR
TOTAL QTY
LIST INVENTORY BY PROD_NAME BY.EXP COLOR PROD_NAME BREAK.ON COLOR TOTAL QTY
 10:09:47 Apr 13 2011 1
           Product
INVENTORY. Name..... Color..... Quantity

    10007 Adapter    N/A              544
    13001 Adapter    N/A              467
    13002 Adapter    N/A              104
                      ********** --------
                      N/A             1115

    39400 CD Player  Black            399
                      ********** --------
                      Black            399

    39400 CD Player  Gray             499
    39500 CD Player  Gray            -551
                      ********** --------
                      Gray             -52
.
.
.
56090 Wrist Rest Red              500
                      ********** --------
                      Red              500

    56090 Wrist Rest Rose             499
                      ********** --------
                      Rose             499

                                 ========
TOTAL                              343316
261 records listed
```

Displaying the number of records at each breakpoint

UniData supports the use of a special virtual attribute which you can use to display the number of records contained within each breakpoint. This virtual attribute contains the value “1” in attribute two, a display value of “\” in attribute four, and a format of “0R” or “0L” in attribute five, as shown in the following example:

```
:AE DICT ORDERS COUNT
Top of "COUNT" in "DICT ORDERS", 6 lines, 12 characters.
*--: P
001: V
002: 1
003:
004: \
005: 0L
006: S
Bottom.
```

```
*--:
```

When you total this attribute in a UniQuery statement that also contains a `BREAK.ON` clause, the detail lines do not display "1" due to the "0L" format, but UniQuery does display subtotals at each breakpoint and a total at the end of the report, indicating the number of records at each breakpoint and a grand total for the report.

Note: UniQuery only supports this special virtual attribute in ECLTYPE P with UDT.OPTIONS 108 on. UDT.OPTIONS 48 must be on if the attribute is right-justified.

In the following example, UniQuery displays records from the ORDERS file, breaking on PROD_NAME, totalling the QTY attribute, and displaying the number of records at each breakpoint:

```
:LIST ORDERS BY.EXP PROD_NAME BREAK.ON PROD_NAME TOTAL QTY TOTAL
COUNT
.
.
.
788        CD Player         1
789        CD Player       850
808        CD Player         1
848        CD Player         1

           CD Player       853 4

813        CD System 1       1
829        CD System 1      50
                            50
892        CD System 1       2

           CD System 1     103 3

829        CD System 10     50
922        CD System 10      1
CD System 10     51 2

794        CD System 11    999

           CD System 11    999 1

789        CD System 12     50
789        CD System 12     50
852        CD System 12      1
933        CD System 12      2

           CD System 12    103 4
.
.
.
```

Note: When you use this special virtual attribute, UniQuery suppresses the default line of asterisks (****) and dashes (----) at each breakpoint.

Using BREAK.SUP

The UniQuery BREAK.SUP keyword is used with the LIST and SORT commands to create a break in a report when the value of an attribute changes, but suppresses the printing of the breakpoint value. Even though UniQuery does not display the breakpoint value, the output is separated by a blank line when a breakpoint occurs.

Syntax

...BY *attribute* **BREAK.SUP** ["['*option*' ...][*text*]"] *attribute*

You can use all of the BREAK.ON options with the BREAK.SUP keyword, although only the 'B', 'P', and 'T' options make sense with BREAK.SUP, since the breakpoint value is suppressed.

The following table describes the *option* parameters available with the BREAK.SUP keyword.

Parameter	Description
B	Used in conjunction with the HEADING or FOOTING keyword. When the "'B'" option is specified in the heading or footing definition, UniQuery takes the value of the attribute name at the breakpoint and inserts it at the position you designate in the heading or footing definition. If you specify more than one BREAK.ON "'B'" option, UniQuery only uses the first one it encounters in the statement. Each time the breakpoint value changes, UniQuery creates a new page.
P	Creates a new page for each breakpoint value.
T	Displays a line of dashes (---) before the subtotal when you use BREAK.SUP with the TOTAL keyword and the DET.SUP keyword.

In the following example, the BREAK.SUP keyword creates a breakpoint on the STATE attribute, but does not print the value of STATE. The 'B' and the 'P' options are used to print the breakpoint value in the header, and to create a new page each time the breakpoint value changes:

```
:LIST CLIENTS WITH COUNTRY = "USA" BY STATE BREAK.SUP "'BP'" STATE
CITY PHONE_NUM HEADING "LIST OF CLIENTS IN 'B'"
LIST OF CLIENTS IN AR
CLIENTS... City.......... Phone Number..

10094      Grand Forks         5182035692
                               5182036042
(Page Break)
LIST OF CLIENTS IN AZ
CLIENTS... City.......... Phone Number..

10000      Phoenix             6025895874
                               6025898745
10086      St. Louis           4012907597
                               4012908352
(Page Break)
LIST OF CLIENTS IN CA
CLIENTS... City.......... Phone Number..

9982       Los Angeles         3102584569
                               3102584568
9986       Fresno              2095874563
                               2095875589
.
.
```

.

Customizing columns

UniQuery supplies several keywords you can use to alter the default column headers and spacing in a UniQuery report.

Defining column headings

By default, UniQuery uses the column heading defined in attribute 4 of the dictionary record for an attribute. If no column heading is defined in the dictionary record, UniQuery uses the name of the dictionary record for the column heading.

To change the column heading for a specific attribute in the current execution of UniQuery statement only, use the `COL.HDG` keyword.

Syntax

...attribute **COL.HDG** "*column_heading*"

You must enclose *column_heading* in single or double quotation marks.

In the following example, the dictionary record for the QTY attribute in the INVENTORY file has a column heading of “Quantity” defined in attribute four. The UniQuery statement lists all records from the INVENTORY file with a quantity less that zero. The column heading for QTY displays as “Back Ordered” by using the `COL.HDG` keyword:

```
:AE DICT INVENTORY QTY
Top of "QTY" in "DICT INVENTORY", 7 lines, 33 characters.
*--: P
001: D
002: 6
003: MD0
004: Quantity
005: 6R
006: MV
007: LINE_ITEMS
Bottom.
*--:

:LIST INVENTORY WITH QTY LT '0' PROD_NAME QTY COL.HDG "Back
 Ordered"
LIST INVENTORY WITH QTY LT '0' PROD_NAME QTY COL.HDG "Back
 Ordered" 14:44:14 Apr 13 2011 1
           Product
INVENTORY. Name...... Back Ordered

    11130 Video 12            -67
    38000 CD System           -49
           11
    11050 TV                 -794
                               391
    39500 CD Player          -551
4 records listed
.
.
```

.

Suppressing column headings

If you do not want column headings to display in the current execution of UniQuery report, use the COL.HDR.SUPP keyword. The UniQuery COL.HDR.SUPP keyword suppresses all column headings defined in attribute 4 of the dictionary attributes you display in a UniQuery statement.

Syntax

...COL.HDR.SUPP

In the following example, records are listed from the CLIENTS file. No column headings appear in the report because COL.HDR.SUPP is specified in the UniQuery statement:

```
:LIST CLIENTS WITH COUNTRY = "USA" NAME CSZ COL.HDR.SUPP
LIST CLIENTS WITH COUNTRY = "USA" NAME  CSZ COL.HDR.SUPP
 14:57:18 Apr 13 2011 1
10055      Cathy Gross                    Lowell, NY  10572
10059      Weiming Wade                   Mount Holly, SC  25092
10063      Jan Elliott                    Youngstown, CT  02021
10066      Joe Guiney                     Kalamazoo, NE  19999
10067      Larry Singh                    Peekskill, MD  27226
10071      Elyse Klecker                  Flint, PA  12867
10074      Bob Barry                      Peoria, TN  21662
10075      Larry Harrell                  Flagstaff, OH  43116
10078      Scott Lacharite                Mount Pleasant, RI  03159
10079      Peggy Zhu                      Providence, KS  61140
10082      Bruce Harper                   Pomona, NM  88801
10083      Mike Venkatesan                Riverside, HI  92597
.
.
.
```

There are several synonyms for the COL.HDR.SUPP keyword. For more information about these synonyms, see the *UniQuery Commands Reference*.

Altering column spacing

By default, UniQuery places one space between columns in a report. UniQuery determines the width of the column from the format option in the attribute five of the dictionary record for an attribute. The UniQuery COL.SPACES keyword defines the spacing between each column in a horizontal report when you use the LIST or SORT commands.

Syntax

...COL.SPACES *n*

where *n* is the number of spaces between columns.

In the following example, the PROD_NAME and REORDER attributes from the INVENTORY file are printed. The PROD_NAME attribute has a format of 10T, and the REORDER attribute has a format of 6R. One space appears between the columns.

```
:LIST INVENTORY PROD_NAME REORDER
```

```
LIST INVENTORY PROD_NAME REORDER 15:15:18 Apr 13 2011 1
           Product    Reorder
INVENTORY. Name..... Point..

    53050 Photocopie      50
           r
    56060 Trackball       70
    57030 Scanner         30
    31000 CD System       70
           2
    10140 Camera          50
                           50
    11001 Computer        40
.
.
.
```

In the next example, 10 spaces separate each column by use of the `COL.SPACES` keyword:

```
:LIST INVENTORY PROD_NAME REORDER COL.SPACES 10
LIST INVENTORY PROD_NAME REORDER COL.SPACES 10 15:17:10 Apr 13 2011 1
                    Product             Reorder
INVENTORY.          Name......          Point..

    53050          Photocopie               50
                    r
    56060          Trackball                70
    57030          Scanner                  30
    31000          CD System                70
                    2
    10140          Camera                   50
                                             50
    11001          Computer                 40
    10150          Camera                   60
                                             60
.
.
.
```

Creating headers and footers

You can create report headers and/or footers for a UniQuery report by using the UniQuery `HEADING` and `FOOTING` keywords.

Creating headers

The UniQuery `HEADING` keyword creates a header for every page of a report when used with the `LIST` or `SORT` commands. The text in the header should always be contained within double quotation marks. Options for the `HEADING` keyword must be enclosed in single quotation marks. You can intersperse header text and options throughout the header.

Syntax

```
...HEADING "[text] ['option...']"]
```

The following table describes the valid `HEADING` options.

Option	Description
B	Takes the value of the attribute name at the breakpoint and inserts it at the designated position in the heading definition when you specify `BREAK.ON "'B'"` in the UniQuery statement. If you specify more than one `BREAK.ON "'B'"` option, UniQuery only uses the first one it encounters in the statement. Each time the breakpoint value changes, UniQuery creates a new page.
C[*n*]	Centers header text within a line. When you specify the *n* option, UniQuery centers the text is according to this line length.
D	Inserts the current system date at the location of the "'D'" option in the header.
F	Inserts the file name at the location of the "'F'" option in the header.
G	Distributes the words in a header evenly across the line length by inserting gaps.
L	Creates a line feed at the location of the "'L'" option. You can produce multiple header lines using the "'L'" option.
N	Suppresses the default “Enter new line to continue...” statement and causes the report to scroll without pausing until the end of the report.
P or ^	Inserts the current page number in the header. The page number increments with each new page.
T	Inserts the current time and date at the location of the "'T'" option in the header. The time and date are equal to their values at the time the UniQuery statement was executed.

The following example shows a report with a header created with `HEADING` keyword. The ‘T’ option displays the time and date the UniQuery report was executed in the header:

```
:LIST INVENTORY BY PROD_NAME BREAK.ON PROD_NAME TOTAL QTY HEADING
"Inventory on Hand As of 'T'"
Inventory on Hand As of 16:00:33 04-13-11
           Product
INVENTORY. Name...... Quantity

    10007 Adapter         544
    13001 Adapter         467
    13002 Adapter         104
           ********** ---------
           Adapter        1115

    39400 CD Player       399
                           499
    39500 CD Player      -551
           ********** ---------
           CD Player       347

    30000 CD System       310
           1
                           197
           ********** ---------
           CD System       507
           1
.
.
.
```

Note: UDT.OPTIONS 34 determines the format of the system date when using the "'D'" option in a HEADING statement. When this option is on, UniQuery prints the system date in alphanumerics in the footer. When this option is off, UniQuery prints the system date in numeric format. If you issue the DATE.FORMAT command, UniQuery prints the system date in European format either in alphanumeric format or numeric format, depending upon the setting of UDT.OPTIONS 34.

Creating footers

The UniQuery FOOTING keyword creates a footer for every page of a report when used with the LIST or SORT commands. The text in the footer should always be contained within double quotation marks. The options must be enclosed in single quotation marks. You can intersperse footer text and options throughout the footer.

Syntax

```
...FOOTING "[text] ['option...']"]
```

The following table describes the valid FOOTING options.

Option	Description
B	Takes the value of the attribute name at the breakpoint and inserts it at the designated position in the footing definition. If you specify more than one BREAK.ON "'B'" option, UniQuery only uses the first one it encounters in the statement. Each time the breakpoint value changes, UniQuery creates a new page.
C[*n*]	Centers footer text within a line. When you specify the *n* option, UniQuery centers the text according to this line length.
D	Inserts the current system date at the location of the "'D'" option in the footer.
F	Inserts the file name at the location of the "'F'" option in the footer.
G	Distributes the words in a footer evenly across the line length by inserting gaps.
L	Creates a line feed at the location of the "'L'" option. You can produce multiple footer lines using the "'L'" option.
N	Suppresses the default “Enter new line to continue...” statement and causes the report to scroll without pausing until the end of the report.
P or ^	Inserts the current page number in the footer. The page number increments with each new page.
T	Inserts the current time and date at the location of the "'T'" option in the footer. The time and date are equal to their values at the time the UniQuery statement was executed.

The following report contains a footer created with the FOOTING keyword. The ‘L’ option creates a line feed in the footer:

```
:LIST CLIENTS WITH STATE = 'NY' NAME ADDRESS CSZ FOOTING
"CLIENTS IN THE STATE OF 'L'NEW YORK"
LIST CLIENTS WITH STATE = 'NY' NAME ADDRESS CSZ FOOTING
"CLIENTS IN THE STATE OF 'L'NEW YORK" 16:08:03 Apr 13 2011 1
CLIENTS   10055
Name      Cathy Gross
```

```
Address   963 South A Boulevard
          Lowell, NY  10572

CLIENTS   9965
Name      Gary Phillips
Address   8899 S. Taylor St.
          New York, NY  00213

2 records listed



CLIENTS IN THE STATE OF
NEW YORK
:
```

Note:

UDT.OPTIONS 21 determines whether UniQuery executes a carriage return at the end of a report directed to the terminal screen. When the "'L'" option is present in a `FOOTING`statement and UDT.OPTIONS 21 is on, UniQuery pauses and waits for a carriage return before returning to the ECL command line prompt at the end of the report. If UDT.OPTIONS 21 is off, UniQuery automatically executes a carriage return and returns to the ECL command line prompt at the end of the report. If you use the "'N'" option with the "'L'" option, you automatically return to the ECL command line prompt regardless of the setting of UDT.OPTIONS 21.

UDT.OPTIONS 34 determines the format of the system date when using the D option in a `FOOTING` statement. When this option in on, UniQuery prints the system date in alphanumerics in the footer. When this option is off, UniQuery prints the system date in numeric format. If you issue the `DATE.FORMAT` command, UniQuery prints the system date in European format either in alphanumeric format or numeric format, depending upon the setting of UDT.OPTIONS 34.

Suppressing UniQuery defaults

Suppressing the default header

By default, UniQuery prints a header at the top of each UniQuery report, showing the UniQuery statement executed, the time and date the statement was executed, and the page number. You can suppress this default header by using the `HDR.SUPP` keyword.

Syntax

`...HDR.SUPP`

In the following example, `HDR.SUPP` suppresses the default header:

```
:LIST CLIENTS NAME HDR.SUPP
CLIENTS... Name.........................

9999       Paul Castiglione
10052      Paul O'Hare
10053      Gino Lee
10054      Gregory Alps
.
.
```

```
.
```

HDR.SUPP has several synonyms, which differ by ECLTYPE. For more information about HDR.SUPP synonyms, see the *UniQuery Commands Reference*.

Suppressing the record ID

By default, UniQuery prints the record ID (@ID) of each record in the left-most column of a report. To suppress the printing of the @ID, use the ID.SUPP keyword.

If you want to display the record IDs in a column other than the first, you can use ID.SUPP, then specify @ID as a display attribute in the desired location in a UniQuery statement.

Syntax

...ID.SUPP

In the following example, ID.SUPP suppresses the @ID in the UniQuery report:

```
:LIST INVENTORY BY PROD_NAME PROD_NAME QTY ID.SUPP
LIST INVENTORY BY PROD_NAME PROD_NAME QTY ID.SUPP 15:59:30 Apr 10 2011 1
Product
Name...... Quantity

Adapter          544
Adapter          467
Adapter          104
CD Player        399
                 499
.
.
.
```

ID.SUPP has several synonyms which differ by ECLTYPE. For more information about ID.SUPP synonyms, see the *UniQuery Commands Reference*.

Suppressing the number of items listed

When you execute a UniQuery statement, UniQuery displays the number of items listed in the report at the end of the report by default. If you do not want to display the number of items listed, use the UniQuery COUNT.SUP keyword.

Syntax

...COUNT.SUP

The following example illustrates output from a UniQuery statement, where UniQuery displays the number of records listed by default:

```
:LIST ORDERS PROD_NAME COLOR PRICE SAMPLE 8
LIST ORDERS PROD_NAME COLOR PRICE SAMPLE 8 10:26:52 Apr 14 2011 1
ORDERS.... Product Name Color..... Price.....

903        Cassette     Black         $228.82
           Recorder
965        Computer     Black       $1,799.99
```

```
           Telephone    Black         $139.99
841        Video 16     Black          $99.97
           Telephone    Tan            $29.99
872        TV           Black         $129.87
934        Adapter      N/A            $94.00
810        Telephone    White          $69.92
           Telephone    Silver         $47.72
           Hard Drive   Gray          $500.00
873        Computer     Black      $17,499.99
           Modem        Black         $120.99
           Cable        None            $9.99
           Printer      Gray          $399.00
966        Computer     Black          $99.95
           Case
8 records listed


:
```

In the following example, the `COUNT.SUP` keyword is added to the UniQuery statement. Notice that the number of records listed, as shown in the previous example, is suppressed:

```
:LIST ORDERS PROD_NAME COLOR PRICE SAMPLE 8 COUNT.SUP
LIST ORDERS PROD_NAME COLOR PRICE SAMPLE 8 COUNT.SUP 11:39:09
 Apr 14 2011 1

ORDERS.... Product Name Color..... Price.....

903        Cassette     Black         $228.82
           Recorder
965        Computer     Black       $1,799.99
           Telephone    Black         $139.99
841        Video 16     Black          $99.97
           Telephone    Tan            $29.99
872        TV           Black         $129.87
934        Adapter      N/A            $94.00
810        Telephone    White          $69.92
           Telephone    Silver         $47.72
           Hard Drive   Gray          $500.00
873        Computer     Black      $17,499.99
           Modem        Black         $120.99
           Cable        None            $9.99
           Printer      Gray          $399.00
966        Computer     Black          $99.95
           Case


:
```

Changing the margin

By default, a UniQuery report begins printing flush with the left-most margin (column 0). You can use the `MARGIN` keyword to change this margin. `MARGIN` affects only the data display and column headings; it does not affect the default page headers, defined headers, or defined footers.

Syntax

MARGIN *width*

In the following example, the MARGIN keyword indents the data five spaces from the left, but prints the header flush with the left margin:

```
:LIST ORDERS WITH ORD_DATE GE "01/01/96" AND WITH ORD_DATE LE
"03/31/96" PROD_NAME QTY ORD_DATE MARGIN 5 HEADING "First Quarter
 Orders"
First Quarter Orders
    ORDERS.... Product Name Qty... Order Date

    903        Cassette          3   01/13/96
                Recorder
    965        Computer          1   01/15/96
                Telephone         2
    872        TV               45   01/22/96
    934        Adapter          25   01/14/96
    873        Computer          3   01/21/96
                Modem             3
                Cable             6
                Printer           3
    966        Computer          1   01/15/96
                Case
.
.
.
```

Printing a report vertically

By default, UniQuery prints reports horizontally. If the information is too wide to fix across the page or screen, UniQuery prints reports vertically. To force a report to print vertically, use the VERTICAL keyword.

Syntax

...VERTICAL

In the following example, UniQuery prints the names and address from records in the CLIENTS file vertically:

```
:LIST CLIENTS NAME ADDRESS CSZ VERTICAL
LIST CLIENTS NAME ADDRESS CSZ VERTICAL 09:51:40 Jun 23 2011 1
CLIENTS   9999
Name      Paul Castiglione
Address   45, reu de Rivoli
          Paris,   75008

CLIENTS   10052
Name      Paul O'Hare
Address   918 W. Alta St.
          Perth,   8569

CLIENTS   10053
Name      Gino Lee
Address   483 E. Silverman St.
          Fonthill, Ontario  L0S1E5
.
```

```
.
.
```

Double-spacing a report

By default, UniQuery skips a single line between records in a report. The `DBL.SPC` keyword forces two lines between records in a report.

Syntax

`...DBL.SPC`

In the following example, the `DBL.SPC` keyword skips two lines between each record in the UniQuery report:

```
:LIST INVENTORY PROD_NAME COLOR DBL.SPC
LIST INVENTORY PROD_NAME COLOR DBL.SPC 12:25:03 Jun 04 2011 1
           Product
INVENTORY. Name..... Color.....

15001      Modem     N/A

35000      Speaker   Black
                     Charcoal

15002      Modem     Gray

54090      Disk Drive N/A

52070      Printer   Black

50050      Computer  Black

15003      Modem     Gray
.
.
.
```

Suppressing detail in a report

The UniQuery `DET.SUP` keyword suppresses all detail lines in a report, displaying only breakpoint lines and total lines.

In the following example, UniQuery lists records from the ORDERS file by color, and displays the total price. Because `DET.SUP` is used in the UniQuery statement, UniQuery suppresses the detail lines:

```
:LIST ORDERS BY.EXP COLOR BREAK.ON COLOR TOTAL PRICE DET.SUP
LIST ORDERS BY.EXP COLOR BREAK.ON COLOR TOTAL PRICE DET.SUP 08:43:08 Jun 07 2005 1
Color..... Price.......
Almond      $1,209.01
Beige      $12,215.70
Black     $201,500.36
Blue          $687.72
Brown          $39.97
Burgundy       $79.94
Charcoal      $427.80
```

```
Cherry       $1,799.88
Cordovan       $699.65
Cranberry       $69.85
Fuschia          $0.00
Gray        $82,116.59
Green          $200.52
N/A         $27,609.89
NA           $1,094.90
None             $9.99
Oak          $1,199.92
Oak grain      $799.48
Red            $717.65
Rose            $25.98
Silver       $2,143.63
Standard     $1,289.82
Tan         $34,299.61
White        $1,044.55
Yellow         $159.84
           ============
TOTAL       $371,442.25
508 records listed
```

Suppressing @UQ and @LPTR phrases

If you do not include any attributes in the UniQuery statement, the `LIST` and `SORT` commands display the attributes you define in the @UQ record in the dictionary portion of the file, if one exists. If you include the `LPTR` keyword in the UniQuery statement to send the report to a printer, `LIST` and `SORT` display the attributes you define in the @LPTR phrase in the dictionary portion of the file, if one exists. If you do not define @UQ or @LPTR, UniQuery displays only the record IDs. If you specify at least one display attribute, @UQ and @LPTR are ignored.

The UniQuery `ONLY` keyword suppresses the use of the @UQ and @LPTR phrases, if they exist.

Syntax

`...ONLY`

The following example displays the @UQ dictionary record in INVENTORY file. UniQuery displays the attributes defined in the @UQ dictionary record when no other attributes are defined in the UniQuery statement.

```
:AE DICT INVENTORY @UQ
Top of "@UQ" in "DICT INVENTORY", 7 lines, 55 characters.
001: PH
002: INV_DATE INV_TIME PROD_NAME FEATURES LINE_ITEMS
003:
:
:LIST INVENTORY
LIST INVENTORY INV_DATE INV_TIME PROD_NAME FEATURES LINE_ITEMS
16:43:03 Apr 21 2011 1
           Inventory  Inventory Product
INVENTORY. Date...... Time..... Name...... Features.....................


15001      08/20/1995   01:00PM Modem      14.4K Internal V34
35000      07/09/1995   10:00AM Speaker    250W, Direct/reflecting
15002      08/12/1995   07:00AM Modem      14.4K External V34
54090      01/03/1995   10:00AM Disk Drive 5.25" Floppy
52070      01/23/1996   02:50PM Printer    Portable Color, 3 ppm
```

```
50050      01/24/1995   09:39AM Computer   486SL133 CPU, 4MB, 250MB
                                           Laptop
15003      08/15/1995   06:00PM Modem      28.8K External V34
.
..
```

In the next example, the ONLY keyword suppresses the use of @UQ:

```
:LIST ONLY INVENTORY
LIST ONLY INVENTORY 16:44:43 Jun 21 2011 1
INVENTORY.
15001
35000
15002
54090
52070
50050
15003
53080
51060
15004
56010
.
.
.
```

ONLY has several synonyms which differ by ECLTYPE. For more information about ONLY synonyms, see the *UniQuery Commands Reference*.

Controlling page splitting

UniQuery does not prevent data that you may want grouped together from splitting across pages, unless you use the NO.SPLIT keyword.

The NO.SPLIT keyword prevents records from being split across page boundaries in a UniQuery report. If an entire record does not fit on the remaining lines of a page, UniQuery prints the record on the following page.

NO.SPLIT also prevents breakpoints and totals from printing at the beginning of a page. UniQuery prints at least one associated record with the breakpoint or total on the same page.

Syntax

...NO.SPLIT

In the following example, the NO.SPLIT keyword forces UniQuery to keep all values for a record on the same page. The “Enter <New line> to continue” prompt indicates the end of a page.

```
:LIST INVENTORY PROD_NAME QTY NO.SPLIT
LIST INVENTORY PROD_NAME QTY NO.SPLIT 11:15:25 Jun 21 2011 1
           Product
INVENTORY. Name..... Quantity

15001      Modem          7486
35000      Speaker         148
15002      Modem          3988
54090      Disk Drive      575
52070      Printer        4598
50050      Computer         15
```

```
15003       Modem          4913
53080       Photocopie     3965
            r
            Cartridge
51060       Telephone
15004       Modem           146
56010       Keyboard        473
58030       Monitor         487
55000       Cable          5709
10060       Camera         3356
                           3795
11070       TV              685
                            559
Enter <New line> to continue...
LIST INVENTORY PROD_NAME QTY NO.SPLIT 11:15:25 Jun 21 2011 2
            Product
INVENTORY. Name...... Quantity

14001       Memory         6131
34000       Speaker         197
14002       Memory         6415
50060       Computer        325
.
.
.
```

Overriding dictionary attributes

UniQuery, by default, uses the conversion codes and the format defined in the dictionary record for each attribute when displaying data. UniQuery provides some keywords that allow you to override the conversion codes and formats.

Overriding conversion codes

The UniQuery CNV keyword applies a conversion to an attribute or expression during the current execution of the UniQuery statement only. When you use the CNV keyword, the conversion code you specify overrides the conversion code defined in attribute 3 of the dictionary record for a specified attribute. You must enclose the conversion code you are specifying in single or double quotation marks.

Syntax

```
...attribute CNV "conversion_code"
```

If the *conversion_code* you specify is an empty string or invalid, UniQuery does not perform a conversion, even if one is present in the dictionary record for *attribute*. *conversion_code* can be any valid dictionary conversion code. For a list of valid conversion codes, see *Using UniData*.

In the following example, the CNV keyword is used to display ORD_DATE from the ORDERS file in the D4- format. The ORD_DATE dictionary record has a conversion code of D2/.

```
:LIST ORDERS ORD_DATE CNV "D4-"
LIST ORDERS ORD_DATE CNV "D4-" 14:25:59 Jun 03 2011 1
ORDERS.... Order Date

903         01-13-1996
965         01-15-1996
```

```
841         07-23-1995
872         01-22-1996
934         01-14-1996
810         09-01-1995
873         01-21-1996
966         01-15-1996
```

Overriding format definitions

The UniQuery FMT keyword formats an attribute according to the format you specify. FMT overrides the format specified in attribute five of the dictionary record for the attribute you specify.

Syntax

...attribute **FMT** "*format*"

The format options specify the column width and justification for text display. The following table lists the format options. *n* represents the column width you are assigning.

Option	Description
*n*L	Left-justified text in a column of *n* width. If the text exceeds the column width, text breaks after *n* characters.
*n*R	Right-justified text in a column of *n* width. If the text exceeds the column width, text breaks after *n* characters.
*n*C	Centered text in a column of *n* width. If the text exceeds the column width, text breaks after *n* characters.
*n*T	Text justification in a column of *n* width. If the text exceeds the column width, text breaks at a space between words.

In the following example, the FMT keyword formats PROD_NAME in a left-justified column 20 characters in length. The dictionary record for PROD_NAME specifies a format of “10T”.

```
:LIST INVENTORY PROD_NAME FMT "20L"
LIST INVENTORY PROD_NAME FMT "20L" 11:31:37 Jun 09 2011 1
           Product
INVENTORY. Name................

15001      Modem
35000      Speaker
15002      Modem
54090      Disk Drive
52070      Printer
50050      Computer
15003      Modem
53080      Photocopier Cartridg
           e
51060      Telephone
.
.
.
```

Creating temporary virtual attributes

The UniQuery EVAL keyword allows you to define a virtual attribute expression for the current execution of a UniQuery statement only. The expression you specify can be any expression valid in a virtual attribute. You must enclose the expression in single or double quotation marks.

For information about creating virtual attributes, see *Using UniData*.

Note: UniQuery only supports the EVAL keyword in ECLTYPE U.

Syntax

EVAL "*expression*"

In the following example, the EVAL keyword lists the city name for clients in the United States, and lists “Foreign” if the country does not equal USA.

```
:LIST CLIENTS NAME COUNTRY EVAL "IF COUNTRY = USA THEN CITY ELSE
'Foreign'"
LIST CLIENTS NAME COUNTRY EVAL "IF COUNTRY = USA THEN CITY ELSE
'Foreign'" 16:25:14 Jun 08 2011 1
                                                          IF COUNTRY =
                                                          USA THEN CITY
CLIENTS... Name.......................... Country........ ELSE 'Foreign'

9999       Paul Castiglione              France          Foreign
10052      Paul O'Hare Australia Foreign
10053      Gino Lee                      Canada          Foreign
10054      Gregory Alps                  France          Foreign
10055      Cathy Gross                   USA             Lowell
10056      Samuel Morrison               Australia       Foreign
10057      Subrina Iguano                Canada          Foreign
10058      Antonette Larnelle            France          Foreign
10059      Weiming Wade                  USA             Mount Holly
10060      George Duncan                 Australia       Foreign
.
.
.
```

Creating labels

UniQuery provides two commands, LIST.LABEL and SORT.LABEL, to create labels in any size you choose. SORT.LABEL is the same as LIST.LABEL, except SORT.LABEL prints data from the *filename* you specify in label format by record ID, if no other sorting criteria is defined. If you define sorting criteria, UniQuery sorts the records accordingly, so the record ID is the final sort level.

Syntax

LIST.LABEL [DICT] *filename* [*record_IDs*] [*selection_criteria*] [*sorting_criteria*] [*attributes* | ALL] [*format_options*] [*report_options*]

SORT.LABEL [DICT] *filename* [*record_IDs*] [*selection_criteria*] [*sorting_criteria*] [*attributes* | ALL] [*format_options*] [*report_options*]

The following table describes the parameters of the LIST.LABEL and SORT.LABEL commands.

Parameter	Description
DICT	Specifies the dictionary portion, rather than the data portion, of *filename*.
filename	Specifies the file on which the operations are to be performed. You can only specify one *filename* for each UniQuery command.
record_IDs	Specifies record_IDs to test against the selection criteria. If you specify more than one *record_ID*, separate them with spaces. You should enclose *record_IDs* in quotation marks so they will not be interpreted as keywords or field names.
selection_criteria	States conditions for bypassing or retrieving records.
sorting_criteria	Defines the order to display the records in the report.
attributes | ALL	Specifies which attributes within *filename* to include in the report. Each attribute must be defined in the dictionary of *filename*. Use `ALL` to display all D-type attributes in a record.
format_options	Specifies how to format the report, including page breaks, breakpoint values, headers, and footers.
report_options	Specifies keywords that control report output, including `LPTR`, `NO.PAGE`, `SAMPLE`, and `SAMPLED`.

After you enter the `LIST.LABEL` or `SORT.LABEL` statement, UniData prompts for the following information to format the labels.

Option	Description
COL	The number of columns across the page.
ROW	The number of lines included on each label. ROW should be equal to the number of attributes specified, plus one if ID_SUPP is not specified.
SKIP	The number of lines between each label.
INDNT	The number of spaces to indent on the left side of each label. If INDNT is greater than 0, you can specify a header at the beginning of each row.
SIZ	The width of each line on a label. If SIZ is greater than the width of the label, the line is truncated.
SPACE	The number of spaces between each column of labels.
C	Specifies to ignore empty strings. If the UniQuery statement does not contain C, a blank line is inserted for each empty string.

If INDNT is greater than 0 and you do not suppress column headings, UniData prompts for a header for each row of the label. The text you enter appears adjacent to each row in the first column of the labels. If the setting of INDNT is not wide enough to accommodate the length of the header, the header for the row may overwrite the data in the label. If you do not want to print a header for each row, press RETURN at each header prompt.

In the following example, `SORT.LABEL` creates labels from data in the CLIENTS file.

```
:SORT.LABEL CLIENTS NAME ADDRESS CSZ ID.SUPP
COL,ROW,SKIP,INDNT,SIZ,SPACE(,C) :1,8,1,0,40,1,C
Andre Halligan
854 Ivy St.
Suite 4200
Phoenix, AZ 80598
```

```
Patricia Halberg
62 LaSalle Ave.
South Bend, IN 20687


Aude Grenelle
Av. Bourgailh
Bordeaux, 75001
.
.
.
```

Chapter 5: Creating and using select lists

This chapter provides information about creating, saving, deleting, and activating select lists. This chapter also discusses manipulating and merging select lists.

Defining select lists

A select list is a collection of items, usually record IDs, that satisfy the selection criteria you define in a UniQuery statement.

When you issue a UniQuery statement containing selection criteria without defining specific record IDs, UniQuery evaluates each record in the file to see if it meets the selection criteria you define. Normally, you do not want to view all the records in a file, but rather a subset of data.

For example, you may want to create several reports for items from the ORDERS file with a price less than $100.00. Rather than executing several UniQuery statements against the entire file, you can select the records with a price less than $100.00 and save them in a list. Then, you can activate the saved list and create reports using the qualified records. This can significantly reduce processing time.

Creating select lists

You create select lists with the UniQuery `BSELECT`, `ESEARCH`, `FORM.LIST`, `QSELECT`, `SELECT`, `SSELECT` or `USHOW` commands.

Using the SELECT command

The `SELECT` command is the most common command used to create select lists.

Syntax

`SELECT` *`filename`* `[`*`record_IDs`*`]` `[`*`selection_criteria`*`]` `[`*`sorting_criteria`*`]` `[SAVING [UNIQUE]` *`attribute`* `[NO.NULLS]] [TO` *`list_num`*`]`

The UniQuery `SELECT` command creates up to 10 active select lists (0-9) or record IDs (or keys) that may be used by other UniQuery commands or other UniData processes.

If you do not specify the TO option with *list_num*, UniQuery saves the select list to the default list number of 0. The greater than (>) prompt indicates that a select list is active.

In the following example, the first `SELECT` statement selects accounts with an ID of 123456 and is then saved to select list 0. The second select statement saved to select list 3 by use of the TO option.

```
:SELECT CUSTOMER WITH STATE = "CO"

11 records selected to list 0.

>SAVE.LIST CO
11 key(s) saved to 1 record(s).
:SELECT CUSTOMER WITH STATE = "NY" TO 3

1 records selected to list 3.

:
```

UniQuery accepts attribute names in selection and sorting criteria, but you cannot specify attributes for display purposes in a SELECT statement. SELECT statements do not produce output, and therefore do not accept any format specifications.

The following table describes the parameters of the SELECT syntax.

Parameter	Description
filename	Name of the file from which to retrieve data. You may specify only one file name for each UniQuery statement.
record_IDs	Specifies record IDs to test against the selection criteria. You can specify up to 128 record IDs in a SELECT statement.
selection_criteria	Specifies conditions for bypassing or retrieving a particular record.
sorting_criteria	Specifies how to sort the retrieved attributes.
SAVING [UNIQUE] *attribute* [NO.NULLS]	Saves specified attribute to an active select list, rather than the primary key. The UNIQUE option prevents UniQuery from saving duplicate values. The NO.NULLS be option prevents UniQuery from saving empty strings, but does not prevent UniQuery from saving the null value. If you specify SAVING UNIQUE, the select list is sorted in BY.EXP order.
TO *list_num*	Specifies the active select list number to which records are to be saved. The list can be numbered from 0 through 9.

SELECT creates a list of records IDs called an active select list. Any UniQuery command that uses an active list can then be executed subsequent to the creation of an active list. Programs written in UniBasic can use the READNEXT function against an active select list to read successive record IDs.

In the following example, UniQuery creates an active select list of record IDs in the ORDERS file with a price less than $100.00, then lists the records by price.

```
:SELECT ORDERS WITH EVERY PRICE < "100.00"

39 records selected to list 0.

>LIST ORDERS BY PRICE PROD_NAME PRICE
LIST ORDERS BY PRICE PROD_NAME PRICE 11:42:24
Aug 07 2011 1
ORDERS.... Product Name Price.......

969        Mouse Pad           $3.99
                               $3.99
                               $3.99
1000       Cable               $7.95
           Cable              $24.99
           Cable               $8.99
819        Cable               $7.95
           Cable               $8.99
           Cable               $9.99
           Cable              $17.99
           Cable               $8.99
           Cable               $8.99
.
.
.
```

In the preceding example, the greater than sign (>) indicates an active select list. The lists you create with the SELECT command can be stored in a the SAVEDLISTS file using the SAVE.LIST command.

To activate a saved list, use the GET.LIST command. To deactivate the list, use the CLEARSELECT command.

For more information about the SAVE.LIST, GET.LIST, SELECT, and CLEARSELECT commands, see the *UniQuery Commands Reference*.

Using the SSELECT command

SSELECT creates a list of records IDs in ascending order, called an active select list. Any UniQuery command that uses an active list can be executed subsequent to the creation of an active list. SSELECT is the same as the SELECT command, except records are selected by ascending record ID if you do not specify any other selection criteria. If you define selection criteria, UniQuery sorts the records accordingly, with the final sort level the record ID.

Syntax

SSELECT *filename* [*record_IDs*] [*selection_criteria*] [*sorting_criteria*] [SAVING [UNIQUE] *attribute* [NO.NULLS]] [TO *list_num*]

The following table describes the parameters for the SSELECT command.

Parameter	Description
filename	Name of the file from which to retrieve data. You can specify only one file name for each UniQuery statement.
record_IDs	Specifies record IDs to test against the selection criteria. You can specify up to 128 record IDs in a SELECT statement.
selection_criteria	Specifies conditions for bypassing or retrieving a particular record.
sorting_criteria	Specifies how to sort the retrieved records. If you specify sorting criteria, the final sort is by record ID when you use the SSELECT command.
SAVING [UNIQUE] *attribute* [NO.NULLS]	Saves specified attribute to an active select list, rather than the primary key. The UNIQUE option prevents duplicate values from being saved. The NO.NULLS option prevents empty strings from being saved, but does not prevent the null value from being saved.
TO *list_num*	Specifies the active select list number to which records are to be saved. The list can be numbered from 0 through 9.

Using the BSELECT command

The UniQuery BSELECT command retrieves data from a file into an active select list. Unlike the SELECT command, which retrieves only record IDs, BSELECT builds a list of the attributes you name in the UniQuery statement. You must name at least one attribute in the statement.

Note: UDT.OPTIONS 59 determines if a BSELECT statement creates a blank line in a select list if the designated attribute does not exist in a record. When UDT.OPTIONS 59 is on, UniQuery does not create a blank line for each non-existing attribute. When UDT.OPTIONS 59 is off, a blank line is created for each non-existing attribute.

Syntax

BSELECT *filename* ['*record_IDs*'] [*selection_criteria*] *attribute* [*attribute...*]

The following table describes the parameters of the BSELECT command.

Parameter	Description
filename	Name of file on which operations are to be performed. You may specify only one file name in a UniQuery statement.
record_IDs	Record IDs to test against the selection criteria. The record IDs must appear in single or double quotation marks. If you do not specify any record IDs, UniQuery searches the entire data file.
selection_criteria	Conditions for retrieving or bypassing a record. UniQuery only retrieves records meeting the selection criteria.
attribute	One or more attributes within *filename* to include in the result. You must specify at least one attribute. If you specify an attribute that is multivalued or multi-subvalued, UniQuery stores each value or subvalue on a separate line.

BSELECT is useful when selecting attributes from one file that are record IDs in another file. In the following example, BSELECT retrieves the client numbers from the ORDERS file, then a UniQuery statement lists the names and addresses of those clients from the CLIENTS file.

```
:BSELECT ORDERS CLIENT_NO

192 records selected to list 0.

>LIST CLIENTS NAME ADDRESS
LIST CLIENTS NAME ADDRESS 08:50:00 Jun 02 2005 1
CLIENTS... Name.......................... Address..................

9988       Dominic Warner                7235 Laguna Blvd
                                         Suite 720
9987       Glen Asakawa                  220 Pearl
10002      Aude Grenelle                 Av. Bourgailh
9979       Andrea Herriot                91, promenade Plage
9978       Mike Vidulich                 165 Market Street
.
.
.
```

Using the ESEARCH command

The UniQuery ESEARCH command enables you to search a file for specific values. The ESEARCH command creates an active select list of record IDs that satisfy the selection criteria you define in a UniQuery statement, and also contain occurrences of the character strings you stipulate.

After you enter the ESEARCH statement, UniQuery prompts for a string. At the STRING prompt, enter the string you are searching for. You can specify more than one string. When you finish entering the strings, press ENTER at the STRING prompt. The total length of all the strings you specify cannot exceed 500 characters.

Syntax

ESEARCH *filename* [*record_IDs*] [*selection_criteria*] [(*option*]

The follow table describes the parameters of the ESEARCH command.

Parameter	Description
filename	Specifies the name of the file on which to perform operations. You can designate only one file name for each UniQuery command. The file name must follow the command name, except where the DICT keyword is used.
record_IDs	Specifies the record_IDs UniQuery tests against the selection criteria.
selection_criteria	States conditions for retrieving or bypassing a particular record.
(option	Specifies additional functions to be performed or output conditions to be met during the primary operation. If you do not specify an option, UniQuery selects a record if it contains any of the specified string(s).

The following table lists the ESEARCH options.

Option	Description
(A	Selected records must contain all specified strings.
(I	Display record IDs as they are selected.
(N	Selected records must not contain any of the specified string(s).
(S	Display only the record IDs of selected records; does not form the select list.

In the following example, the ESEARCH command forms a select list of all records in the CLIENTS file that do not contain the string “Gross” in the record using the (N option. The (I option displays the record IDs as they are selected.

```
:ESEARCH CLIENTS WITH STATE = "NY" (NI
   STRING : Gross
    STRING :
9965

1 records selected to list 0.

>
```

Note: You cannot use the LIKE keyword in conjunction with ESEARCH.

Using the FORM.LIST command

The UniQuery FORM.LIST command creates an active select list consisting of each attribute from a record. FORM.LIST reads the entire record of the record ID you specify. You can only specify one *record_ID* with this command. FORM.LIST converts all special delimiters into attribute marks so that each value is on a separate line in the resulting select list. The specified record ID is not saved in the list.

Syntax

FORM.LIST *filename record_ID*

The following table describes the parameters of the FORM.LIST syntax.

Parameter	Description
filename	Specifies the name of the file that contains the record ID `FORM.LIST` reads to form the select list. You can designate only one file name for each UniQuery command. The file name must follow the command name, except where the `DICT` keyword is used.
record_ID	Specifies the record_ID `FORM.LIST` reads to convert to an active select list.

In the following example, `FORM.LIST` creates an active select list consisting of each attribute from a record in the INVENTORY file. Notice how each multivalue is saved to a separate line in the select list.

```
:FORM.LIST INVENTORY 51070
20 records formed to list 0.
>SAVE.LIST TEST
20 key(s) saved to 1 record(s).
:EDIT.LIST TEST
10209
46740
Telephone
Economy Trimline
Black
White
Green
Red
547
1000
33
289
999
999
999
999
50
50
50
50
~
~
~
"SAVEDLISTS/TEST000" 20 lines, 105 characters
```

Using the QSELECT command

The UniQuery `QSELECT` command extracts data from specified records in a file and creates an active select list.

Syntax:

QSELECT *filename* [*record IDs... | *]* [*(n]*

The following table describes the parameters of the `QSELECT` command.

Parameter	Description
filename	Specifies the name of the file from which UniQuery creates the select list. You can only specify one file name in a UniQuery statement.
record_IDs	Specifies the records from which UniData extracts data.

Parameter	Description
*	Specifies to extract the defined attributes from all records in the file. If you do not specify any attributes, extracts all attributes from all records.
n	Stores the *n*th attribute in the active select list. Must be the location of the attribute in the data file, not the name of the attribute.

QSELECT is useful when selecting attributes from one file that are record IDs in another file. In the following example, QSELECT retrieves the client numbers, which are located in attribute 3, from the ORDERS file, then a UniQuery statement lists the names and addresses of those clients from the CLIENTS file.

```
:QSELECT ORDERS * (3

192 records selected to list 0.

>LIST CLIENTS NAME ADDRESS
LIST CLIENTS NAME ADDRESS 08:50:00 Jun 02 2011 1
CLIENTS... Name..................... Address.................

9988       Dominic Warner               7235 Laguna Blvd
                                        Suite 720
9987       Glen Asakawa                 220 Pearl
10002      Aude Grenelle                Av. Bourgailh
9979       Andrea Herriot               91, promenade Plage
9978       Mike Vidulich                165 Market Street
.
.
.
```

Using the USHOW command

The ECL USHOW command generates lists of selected attributes from UniData files. This command is an implementation of the Prime Information SHOW command.

Syntax

USHOW [DICT] *filename* [*attribute* [*attributeN*...]]

The following table lists the USHOW parameters.

Parameter	Description
DICT	Lists the dictionary file.
filename	A UniData file.
attribute	Name of an attribute to display.

The following example shows the result of USHOW with the ORDERS demo file:

```
:USHOW ORDERS PRODUCT_NO
Page no : 1 of 13                       No. of recs. selected 0 of 193

         ORDERS|Product Number|
 ------------------------------------------------------------------------------
  1|       912| |     55040| |
  2|       801| |     11000| |
```

```
  3|       941| |     50000| |
  4|       805| |     11140| |
  5|       830| |     55090| |
  6|       970| |     13003| |
  7|       863| |     40005| |
  8|       834| |     40007| |
  9|       861| |     56080| |
 10|       890| |     54090| |
 11|       914| |     40007| |
 12|       803| |     10004| |
 13|       832| |     10020| |
 14|       972| |     10090| |
 15|       860| |     57010| |
 ------------------------------------------------------------------------------
Command :
S (range) - Select, C (range) - Clear, F - forwars, B - backwards
Range - ALL, VISIBLE, nn-nn, nn, nn-
```

At the colon (:) prompt, you can do any of the following:

- S — Save a range of record IDs to a select list
- C — Clear a range of record IDs
- F — Move forward through the USHOW display
- B — Move backward through the USHOW display

After creating a select list, UniData displays the active select list prompt (>). At this prompt, you can operate on the active select list or enter `quit` or `QUIT` to exit the `USHOW` process and end the UniData session.

To return to the UniData colon prompt, enter `CLEARSELECT` at the active select list prompt, or press your interrupt key (enable the interrupt key with `PTERM -BREAK`).

Saving select lists

Once UniQuery has selected records meeting the specified selection criteria, UniQuery displays the greater than sign (>), indicating an active select list. From this prompt, you can save the list to a file using the `SAVE.LIST` command immediately after the list is created. You can specify the name of the list or save it to the default name.

Syntax

`SAVE.LIST` [*`list.name`*] [FROM *`list.num`*]

If you do not specify a list name, `SAVE.LIST` names the list by using the operating system process ID or the name set in the UDT_SAVELIST environment variable. To find the process ID, use the `LISTUSER` command. UniData displays the process ID under the USRNBR column. For information on setting environment variables, see *Administering UniData on UNIX* or *Administering UniData on Windows Platforms*.

If you create a list using one of the above commands and want to save the list for later processing, or if you want to use the selected records in multiple processes, save the list with the `SAVE.LIST` command. You can retrieve the list repeatedly using the `GET.LIST` command. The list remains available for use until you delete it with the `DELETE.LIST` command, or remove it with the appropriate system-level command from the SAVEDLISTS directory.

Important: By default at UniData 8.1.0 and later, the saved list is stored as one item in the SAVEDLISTS directory. You may see a performance improvement during a `DELETE.LIST` operation, due to fewer files being involved. This behavior can be returned to the original behavior in UniData 7.3.x and earlier by setting the `udtconfig` parameter SINGLE_SAVEDLIST from 1 (default) to 0.

With SINGLE_SAVEDLIST set to 0, or using UniData 7.3.x or earlier, a saved list that exceeds approximately 34,810 characters on UniData for UNIX or 29,408 on UniData for Windows platforms is saved in multiple parts. Each part has an extension to the specified saved list name, beginning at 000 and incrementing sequentially (001, 002, and so forth).

The following table describes the parameters of the `SAVE.LIST` command.

Parameter	Description
list.name	Specifies name of record to which the active select list is to be saved. UniQuery overwrites an existing saved list of the same name without warning.
FROM *list.num*	If more than one saved list is active, specifies which list to save. If you do not specify *list.num*, UniQuery saves the default list 0.

In the following example, the `SELECT` command retrieves records from the ORDERS file. The `SAVE.LIST` command then saves the records to a list named ALL.ORDERS.

```
:SELECT ORDERS
192 records selected to list 0.
>SAVE.LIST ALL.ORDERS
192 key(s) saved to 1 record(s).
:
```

For more information about the `SAVE.LIST` command, see the *UniQuery Commands Reference*.

Retrieving a saved list

The UniQuery `GET.LIST` command retrieves a select list previously saved with the `SAVE.LIST` command. Any process that can be executed after a `SELECT` statement can also be executed after a `GET.LIST` statement.

Syntax

`GET.LIST` [*account*] [*list_name*] [TO *list_num*]

Once you retrieve a list with the `GET.LIST` command, UniQuery assigns it the default list number of 0 and displays the active select list prompt (>), unless you assign the list a specific number differing from the default. Unlike UniBasic, UniQuery cannot directly process commands against numbered select lists. For information about using select list in UniBasic, see *Developing UniBasic Applications*.

The following table describes the parameters of the `GET.LIST` syntax.

Parameter	Description
account	Full path to the account where the SAVEDLISTS file resides if the list does not reside in the SAVEDLISTS file in the current account.
list_name	The name of the saved list to retrieve. If you do not specify *list_name*, UniQuery retrieves the default list 0.

Parameter	Description
TO [*list_num*]	List number (0 through 9) to which GET.LIST retrieves a saved list. If you do not specify TO l*ist_num*, UniQuery retrieves the saved list to the default list 0.

In the following example, the `GET.LIST` keyword retrieves the list called TEST from the SAVEDLISTS file and stores it in the default list 0. Then the `LIST` command displays the names and addresses of the clients whose record IDs are contained in the list.

```
:GET.LIST TEST
4 records retrieved to list 0.
>LIST CLIENTS NAME ADDRESS
LIST CLIENTS NAME ADDRESS 13:38:15 Jun 10 2011 1
CLIENTS... Name.......................... Address..................

9982       Marc Willette                 1800 Center Street
9986       Sam Gunter                    92 Barracks Road
9987       Glen Asakawa                  220 Pearl
10047      Ray Parker                    2021 Glenwood Boulevard
4 records listed
```

Deleting a select list

The UniQuery `DELETE.LIST` command deletes a saved list from the SAVEDLISTS file. If the list you specify does not exist, or you do not have permission to delete the list, UniQuery displays an error message.

Syntax

`DELETE.LIST` *`list_name`*

list_name is the name of the list you want to delete.

> **Warning:** `DELETE.LIST` deletes the saved list you specify without prompting for confirmation that you want to delete the list.

In the following example, the `DELETE.LIST` command removes the NEW.CLIENTS list from the SAVEDLISTS file:

```
:DELETE.LIST NEW.CLIENTS
'NEW.CLIENTS' deleted.
:
```

Editing and copying saved lists

UniData provides commands to edit and copy existing saved lists.

Editing a saved list

The UniQuery `EDIT.LIST` command enables you to edit a saved list that was previously saved to the SAVEDLISTS file. UniQuery opens the saved list to the system editor. If you do not specify *savedlist_name*, UniQuery opens the system editor anyway.

Syntax

EDIT.LIST [*savedlist_name*]

In the following example, EDIT.LIST opens a saved list called PRODUCTS from the SAVEDLISTS file:

```
:EDIT.LIST PRODUCTS
15001
35000
15002
54090
52070
.
.
.
"SAVEDLISTS/PRODUCTS000" 175 lines, 1050 characters
```

You can also edit a saved list using the Alternate Editor (AE.) A saved list that exceeds approximately 34,810 characters is saved in multiple parts. Each part has an extension to the specified saved list name, beginning at 000 and incrementing sequentially (001, 002, and so forth). The following example illustrates selecting a large number of records, saving those records to a list, and the resulting records in the SAVEDLIST file.

```
SELECT FAMILY_FILE1
40000 records selected to list 0.
>SAVE.LIST FAMILY.RECORDS
40000 key(s) saved to 7 record(s).
:
:LIST SAVEDLISTS WITH @ID LIKE "...FAMILY..."
LIST SAVEDLISTS WITH @ID LIKE "...FAMILY..." 16:05:49 Jun 09 2011 1
SAVEDLISTS
FAMILY.RECORDS000
FAMILY.RECORDS001
FAMILY.RECORDS002
FAMILY.RECORDS003
FAMILY.RECORDS004
FAMILY.RECORDS005
FAMILY.RECORDS006
7 records listed
:
```

Although you do not have to specify each part of the SAVEDLIST record ID when you execute the GET.LIST command, you must specify each part if you use the COPY command or the AE command. In the following example, AE is used to edit one part of the FAMILY.RECORDS saved list:

```
:AE SAVEDLISTS FAMILY.RECORDS000
Top of "FAMILY.RECORDS000" in "SAVEDLISTS", 6,087 lines, 34,809 characters.
0001: 9230
0002: 1158
0003: 10523
0004: 11532
0005: 12541
0006: 2167
0007: 13550
0008: 14559
0009: 15568
0010: 3176
0011: 16577
0012: 17586
0013: 18595
0014: 19604
```

```
.
.
.
*---:
```

Copying a saved list

The UniQuery COPY.LIST command copies an existing saved list in the SAVEDLISTS file to a new list within SAVEDLISTS, or to a new file within the same UniData account. If you do not specify a new list name on the command line, UniQuery prompts you for the new list name. COPY.LIST can also store a list as a record in a UniData file, send the list to a printer, or display the list on the terminal screen.

Syntax

COPY.LIST *saved_list new_list* [D | N | O | P | T]

The following table describes the parameters of the COPY.LIST command.

Parameter	Description
saved_list	Name of the existing list in the SAVEDLISTS file you are copying.
new_list	New list name to which UniData copies the contents of *saved_list*.
D	Deletes the original list after it has been copied. Cannot use with the P or T option.
N	Suppresses automatic paging. Valid only with the T option.
O	Overwrites the contents of *new_list* if *new_list* already exists. Cannot use with the P or T option.
P	Sends contents of *saved_list* to a printer. Valid only with the T option.
T	Sends contents of *saved_list* to the terminal screen. Valid only with the N or P option.

In the following example, the COPY.LIST command copies the existing saved list ALL.CLIENTS to a saved list called NEW.CLIENTS. COPY.LIST also deletes the ALL.CLIENTS saved list and overwrites the existing NEW.CLIENTS saved list by using the O and D options.

```
:COPY.LIST ALL.CLIENTS NEW.CLIENTS -OD
:
```

For more information about the COPY.LIST command, see the *UniQuery Commands Reference*.

Comparing saved lists

UniQuery provides the MERGE.LIST command to create a select list from the difference, intersection, or union of two numbered select lists. To merge saved lists, use the GET.LIST command to assign a list number to each saved list.

Note: MERGE.LIST is only supported in ECLTYPE U.

Syntax

MERGE.LIST *list1* {DIFF | INTERSECTION | UNION} *list2* TO *list3* [COUNT.SUP]

The following table describes the parameters of the MERGE.LIST command.

Parameter	Description
list1...list2	Specifies the two active lists to be merged. Must be a numbered list (0 through 9) rather than select list names. To merge named lists, use the GET.LIST command to assign a select list number to each saved list, then execute MERGE.LIST.
DIFF	Retrieves items that exist in *list1* but not in *list2*.
INTERSECTION	Retrieves items the exist in both *list1* and *list2*.
UNION	Retrieves all items from both lists.
TO *list3*	Specifies the active select list number to store the results of merging *list1* and *list2*. If you do not specify a list number, MERGE.LIST uses the default list 0. If *list3* exists, MERGE.LISToverwrites the list without a warning message.
COUNT.SUP	Suppresses the message that reports the number of items in the resulting list.

Note: The INTERSECTION and UNION operators build a list of unique items. If an item exists in *list1* and *list2*, it is saved only once in *list3*.

Complete the following steps when using the MERGE.LIST command:

1. Create and save the SELECT lists you want to merge. If you saved the lists to a list name, or if the lists already exist in the SAVEDLISTS file, use the GET.LIST command to retrieve the lists to a numbered list.

   In the following example, two lists are created and saved, then retrieved to a numbered list with the GET.LIST command:

   ```
   :SELECT INVENTORY WITH COLOR = "Blue"

   8 records selected to list 0.
   >SAVE.LIST BLUE

   8 key(s) saved to 1 record(s).
   :
   :SELECT INVENTORY WITH COLOR = "Green"

   9 records selected to list 0.
   >SAVE.LIST GREEN
   Overwriting existing saved list.

   9 key(s) saved to 1 record(s).
   :
   :GET.LIST BLUE TO 1
   8 records retrieved to list 1.
   :GET.LIST GREEN TO 2
   9 records retrieved to list 2.
   ```

2. Execute the MERGE.LIST command to merge the two active select lists.

   In the following example, MERGE.LIST creates an active select list of the INTERSECTION between list 1 and list 2.

   ```
   :MERGE.LIST 1 INTERSECTION 2
   3 record(s) selected.
   >SAVE.LIST DIFF
   ```

In the previous example, no list number was specified to save the MERGE.LIST results, so MERGE.LIST stores the results to list 0. From the > prompt, you can execute the SAVE.LIST command to name an active select list to save.

The next example illustrates how to specify a select list number to save the results of the MERGE.LIST command, then save the numbered list to a named list.

```
:GET.LIST BLUE TO 1
Overwriting existing select list.
8 records retrieved to list 1.
:GET.LIST GREEN TO 2
Overwriting existing select list.
9 records retrieved to list 2.
:MERGE.LIST 1 INTERSECTION 2 TO 3
3 record(s) selected.
:SAVE.LIST DIFF FROM 3
```

3. View the merged lists.

   You can view the contents of the list MERGE.LIST creates in one of the following ways:

   - Use the COPY.LIST command with the -T option.
   - Use the EDIT.LIST command.
   - From the operating system prompt, change to the SAVEDLISTS directory and use a system editor.

   In the following example, the COPY.LIST command displays the contents of the DIFF saved list.

```
:COPY.LIST DIFF -T
DIFF
001 10030
002 51020
003 56090
:
```

Chapter 6: Directing report output

UniQuery enables you to direct output from a UniQuery statement to the terminal, to a printer, to the _HOLD_ file, or to another file. This chapter explains generating reports to the terminal, to a file for later use, to a tape, and creating new files with requested attributes using the `REFORMAT` and `SREFORMAT` commands.

Directing output to the terminal

By default, UniQuery directs output from a UniQuery statement to the terminal. UniQuery pauses after 23 lines of output and displays the “Enter <New Line> to continue...” prompt. The following example illustrates the default output from a UniQuery statement:

```
:LIST CLIENTS NAME
LIST CLIENTS NAME 11:30:51 Apr 08 2011 1
CLIENTS... Name.........................

9999       Paul Castiglione
10052      Paul O'Hare
10053      Gino Lee
10054      Gregory Alps
10055      Cathy Gross
10056      Samuel Morrison
10057      Subrina Iguano
10058      Antonette Larnelle
10059      Weiming Wade
10060      George Duncan
10061      Kelly Donalley
10062      Michelle Wickerman
10063      Jan Elliott
10064      Carlos Martinez
10065      Chu Lang
10066      Joe Guiney
10067      Larry Singh
10068      Mitchell Benson
10069      Dean Bronson
10071      Elyse Klecker
Enter <New line> to continue...
LIST CLIENTS NAME 11:30:51 Apr 08 2005 2
CLIENTS... Name.........................

10072      Franklin Sears
10073      Harold Gustano
10074      Bob Barry
.
.
.
```

Suppressing page break pausing

UniQuery provides the `NO.PAGE` keyword to suppress pausing for page breaks when printing a report to the terminal screen. When using the `LIST` or `SORT` command, UniQuery does not display the “Enter <New line> to continue...” prompt.

Syntax

...NO.PAGE

In the following example, the NOPAGE keyword prints a report to the terminal screen without pausing for page breaks:

```
:LIST CLIENTS NAME NOPAGE
LIST CLIENTS NAME 10:59:44 Apr 21 2011 1
CLIENTS... Name.........................

9999       Paul Castiglione
10052      Paul O'Hare

10053      Gino Lee
10054      Gregory Alps
10055      Cathy Gross
10056      Samuel Morrison
10057      Subrina Iguano
10058      Antonette Larnelle
.
.
.
9993       Kathleen Donohue
10047      Ray Parker
9994       Edouard  Nielsen
10048      Thomas Montero
9995       Omar Saulnier
10049      Jennifer Vaughn
9996       Wei Chin
10050      David Silvers
9997       Carol Haig
10051      Tim Knoblauch
9998       Brian Douglass
131 records listed
:
```

Directing output to a printer or HOLD file

You can direct output of UniQuery report to a printer or the _HOLD_ file by using the LPTR command.

Setting printer options in UniData for UNIX

Prior to printing a report to a printer, user the SETPTR command to set printer options. The SETPTR command allows you to define “logical printer units” within a UniData session. A logical printer unit is a combination of a printer destination, a form type, page dimensions, and additional options. By varying form type and options, you can define more than one logical printer unit for a single physical printer.

With SETPTR, you can define up to 31 logical printer units in a single UniData session. Throughout UniData, you can define up to 255, but only 31 can be defined in a single user session.

Syntax

SETPTR *unit*[,*width*,*length*,*topmargin*,*bottommargin*] [,*mode*] [“*spooler_options*”[,*options*]]

The following table lists the parameters of the SETPTR syntax.

Parameter	Description
unit	Logical printer unit number; internal to UniData; you can map this to a UNIX printer or queue with the DEST and FORM options. Must range from 0 through 254; default is 0.
[*width*]	Number of characters per line; must be from 0 to 256; default is 132.
[*length*]	Number of lines per page; must be from 1 to 32,767 lines; default is 60.
[*topmargin*]	The number of lines to leave blank at the top of each page; must be from 0 to 25; default is 3.
[*bottommargin*]	The number of lines to leave blank at the bottom of each page; must be from 0 to 25; default is 3.
[*mode*]	Allows you additional flexibility to direct output; default is 1; see separate table.
["*spooleroptions*"]	Allows you to specify desired spooler options as a quoted string, which UniData then passes directly to the UNIX spooler.
[*options*]	Allows you to specify printing options that UniData then interprets and passes to the UNIX spooler. See separate table.

Note: Users familiar with Pick conventions should be aware that printer unit numbers set with SETPTR are not the same as Pick printer numbers. SETPTR enables you to define logical printer units, which may be, but are not necessarily, linked to specific printers. UniData printer unit numbers are used with the PRINT ON statement in UniBasic to allow multiple concurrent jobs. Use the DEST option of SP.ASSIGN to specify Pick printers and forms.

The next table describes modes for SETPTR.

Mode	Description
1	Directs output to a printer only. Default mode.
2	Must be used with DEVICE option. Directs output to the serial device specified by the DEVICE option.
3	Directs output to a _HOLD_ file only.
6	Directs output to both a _HOLD_ file and a printer.
9	Directs output to a printer. Suppresses display of the _HOLD_ entry name.

The next table describes options for the SETPTR command.

Option	Description
BANNER [*string*]	Modifies the default banner line (which is the UNIX user id). Depends on MODE setting; also modifies _HOLD_ entry name.
BANNER UNIQUE [*string*]	Modifies the default banner line, and automatically uses attribute 1 (NEXT.HOLD) in the dictionary for the _HOLD_ file to create unique entry names for jobs sent to _HOLD_.
BRIEF	Directs UniData not to prompt for verification upon execution of SETPTR.
COPIES *n*	Prints *n* copies of the print job.
DEFER [*time*]	Delays printing until the specified *time*. Consult your host operating system documentation for the correct syntax for specifying time. You will need the documentation for the UNIX at command.

Option	Description
DEST *unit* (or AT *unit*)	Directs output to a specific printer or queue. The *unit* must be a valid destination at your site. Consult your spooler documentation and use the UNIX `lpstat` command for information about valid destinations.
DEVICE *filename*	Used with mode 2 only. Directs output to the UNIX device whose special file is *filename*.
EJECT	Ejects a blank page at the end of each print job.
NOEJECT	Suppresses the form feed at the end of each print job.
FORM {*form*}	Assigns a specified *form* to each print job. The *form* must be defined to your spooler before you use this option.
LNUM	Prints line numbers in the left margin of each print job.
NFMT or NOFMT	Suspends all UniData print formatting.
NHEAD or NOHEAD	Suppresses the banner for each print job.
NOMESSAGE	Suppresses all messages from your UNIX spooler.
OPEN	Opens a print file, and directs output to this file until the file is closed by the `SP.CLOSE` command.

For a complete discussion about managing printers in UniData, see *Using UniData*.

Setting printer options in UniData for Windows platforms

On UniData for Windows platforms, the `SETPTR` command maps printers defined in Windows systems (either local printers or network print devices) to logical unit numbers.

With `SETPTR`, you can define up to 31 logical printer units in a single UniData session. Throughout UniData, you can define up to 255, but only 31 can be defined in a single user session.

The default print unit in UniData is unit 0. You can map this default unit to a particular device with `SETPTR`. If you do not map it explicitly, unit 0 is automatically mapped to one of two printers:

- The default printer for your Windows system. Check **Settings > Printers** to determine which printer is the default.
- A printer identified by the system environment variable UDT_DEFAULT_PRINTER. This definition overrides the default printer for the Windows system. Use the MS-DOS `SET` command or select **Settings > Control Panel > SystemEnvironment** to display or modify UDT_DEFAULT_PRINTER.

The following table lists the parameters of the `SETPTR` syntax.

Parameter	Description
unit	Logical printer unit number; internal to UniData; you can map this to a Windows printer with the DEST option. Valid values range from 0 through 254. The default is 0.
[width]	The number of characters per line: must be from 0 to 256. The default is 132.
[length]	The number of lines per page. Valid values range from 1 to 32,767 lines. The default is 60.
[topmargin]	The number of lines to leave blank at the top of each page. Valid values range from 0 to 25. The default is 3.
[bottommargin]	The number of lines to leave blank at the bottom of each page; must be from 0 to 25. The default is 3.
[mode]	The output direction. The default is 1. See the following table.

Parameter	Description
["*spooler_options*"]	Options that are valid with the Windows spooler. See separate table for list of supported options. Enclose these options in quotation marks.
[options]	Report formatting and printer control options. See the options table.

Note: Users familiar with Pick conventions should be aware that printer unit numbers set with `SETPTR` are not the same as Pick printer numbers. `SETPTR` enables you to define logical printer units, which may be, but are not necessarily, linked to specific printers. UniData printer unit numbers are used with the `PRINT ON` statement in UniBasic to allow multiple concurrent jobs. Use the DEST option of `SP.ASSIGN` to specify Pick printers and forms.

The next table describes modes for `SETPTR`.

Mode	Description
1	Directs output to a printer only. Default mode.
2	Must be used with DEVICE option. Directs output to the serial device specified by the DEVICE option.
3	Directs output to a _HOLD_ file only.
6	Directs output to both a _HOLD_ file and a printer.
9	Directs output to a printer. Suppresses display of the _HOLD_ entry name.

The next table describes options for the `SETPTR` command.

Option	Description
BANNER [*string*]	Modifies the default banner line (which is the Windows user id). Depends on MODE setting; also modifies _HOLD_ entry name.
BANNER UNIQUE (*string*)	Modifies the default banner line and automatically uses attribute 1 (NEXT.HOLD) in the dictionary for the _HOLD_ file to create unique entry names for jobs sent to _HOLD_.
BRIEF	Suppresses the verification prompt.
COPIES *n*	Prints *n* copies. Does not work with mode 3. Default is 1.
DEFER [*time*]	Delays printing until the specified *time*. Specify the time in HH:MM format. Does not work with mode 3.
[DEST | AT] *unit*	Directs output to a specific printer or queue. The *unit* may be either a local printer or a network printer.
DEVICE *name*	Used with mode 2 only. Directs output to the Windows device (for instance, a COM port) identified by *name*.
EJECT	Ejects a blank page at the end of the print job.
NOEJECT	Suppresses the form feed at the end of the print job.
LNUM	Prints line numbers in the left margin.
NFMT | NOFMT	Suspends all UniData print formatting.
NHEAD | NOHEAD	Suppresses the banner.
OPEN	Opens a print file, and directs output to this file until the file is closed by the `SP.CLOSE` command.

The next table describes spooler options you can specify in a quoted string.

Option	Description
Orientation	The paper orientation. Must be PORTRAIT or LANDSCAPE. Defaults to the setting in the Default Document Properties sheet for the printer.
PaperSource	The default paper source; must match an available paper source listed on the Device Settings tab of the printer's Properties Sheet.
Duplex	Must be NONE, HORIZONTAL, or VERTICAL; default is NONE. **Note:** If the print device does not support duplex printing, this option is ignored. Jobs print single-sided and no error message displays.
Form	The form to use (for instance, Letter). Must match an available paper size listed on the Device Settings tab of the printer's Properties Sheet.
Mode	RAW or WINDOW. Default is RAW, meaning that printer-specific escape sequences are required for all formatting. **Note:** Specifying formatting options (Form, Font, FontSize, Orientation, FontStyle, DefaultSource, or Duplex) in a quoted string automatically switches Mode to WINDOW.
Prefix	The printer-specific escape sequence, specified as the literal ASCII characters. Valid in RAW mode only.
Font	The font name, for instance, "Courier New." **Note:** The UniData spooler creates a "logical font" using the values you provide for Font, FontSize, and FontStyle. Windows platforms attempt to find an appropriate font to use from the ones installed on your computer.
FontSize	The font size in points (for instance, 8, 9, 10, 11). **Note:** The UniData spooler creates a "logical font" using the values you provide for Font, FontSize, and FontStyle. Windows platforms attempt to find an appropriate font to use from the ones installed on your computer.
FontStyle	Must be Regular, Italic, Bold, Underline, or StrikeOut. Default is Regular. **Note:** The UniData spooler creates a "logical font" using the values you provide for Font, FontSize, and FontStyle. Windows platforms attempt to find an appropriate font to use from the ones installed on your computer.
LeftMargin	The left margin of the page, in inches.
RightMargin	The right margin of the page, in inches.
TopMargin	The top margin of the page, in inches. **Note:** TopMargin is measured beginning at the value of the `SETPTR topmargin` option (default is 3 lines). If *topmargin* is 3 lines (the default) and TopMargin = 1, the first printed line is one inch below the third line of the page.
BottomMargin	Bottom margin of the page, in inches. **Note:** BottomMargin is measured beginning at the value of the `SETPTR bottommargin` option (default is 3 lines). If *bottommargin* is 3 lines (the default) and BottomMargin = 1, the first printed line is one inch above the third line from the end of the page.

Option	Description
Priority	Must be from 1 to 99, where 1 is minimum priority and 99 is maximum priority.
JobState	The only valid value is PAUSE, which stops all jobs to the print unit. There is no way to reverse this action.

To display information about printers on your Windows system, from the **Start** menu, click **Printers and Faxes**. The local printers may point to the same physical print device or to different physical print devices.

Tip: You can print from UniData to any network print device available to you. A print device does not need to be visible in the Printers dialog box.

You can define local or network printers to UniData by using the `SETPTR` command, as shown in the following examples.

```
:SETPTR
0,,,,,1,AT
LETTER,”TopMargin=1,BottomMargin=1,Font=Courier,FontSize=12”
Unit 0
Mode 1

Options are:
Destination LETTER
Lp options : TopMargin=1,BottomMargin=1,Font=Courier,FontSize=12
OK to set parameters as displayed?(enter y/n)y

:SETPTR 0
Unit 0
Width 105
Length 31
Top margin 3
Bot margin 3
Mode 1
Options are:
Destination LETTER
Lp options : TopMargin=1,BottomMargin=1,Font=Courier,FontSize=12

:SETPTR 1,,,0,0,1,AT \\DENVER4\hpzone3,”Priority=99”
Unit 1
Top margin 0
Bot margin 0
Mode 1
Options are:
Destination \\DENVER4\hpzone3
Lp options : Priority=99
OK to set parameters as displayed?(enter y/n) y

:SETPTR 2,,,,,1,AT LEGAL
Unit 2
Mode 1
Options are:
Destination LEGAL
OK to set parameters as displayed?(enter y/n) Y

:SETPTR 3,,,,,1,AT \\DENVER4\hpzone2,”Form=A4”
Unit 3
Mode 1
Options are:
```

```
Destination \\DENVER4\hpzone2
Lp options : Form=A4
OK to set parameters as displayed?(enter y/n) y
:
```

Notice the following points:

- The default print device (printer unit 0) is now mapped to the local printer LETTER. If you use the PRINT command or LPTR with no print unit specified, your print job is directed to LETTER.
- Use SETPTR unit to display the current settings for a print unit.
- When you specify spooler options (TopMargin, BottomMargin), UniData automatically recalculates the width and length, taking these into account. Also, when you specify formatting options in a quoted string, UniData implicitly changes the spooler Mode from RAW (the default) to WINDOW.
- You can specify spooler options in a quoted string either before or after SETPTR options like AT, DEFER.
- You can map a printer unit to a network print device even if that device is not displayed in your Printers dialog.

After you have defined printers with SETPTR, you can display a list with the LISTPTR command, as shown below:

```
:LISTPTR
Unit.. Printer.................. Port.....................Status..
0   LETTER                      \\DENVER4\hpzone3 'Running
1   \\DENVER4\hpzone3 hpzone3 Running
2   LEGAL                           \\DENVER4\hpzone3 Running
3   \\DENVER4\hpzone2   hpzone2 Running
```

Notice that, in the previous example, the two local printers point to the same network print device.

Use PTRDISABLE and PTRENABLE (STOPPTR and STARTPTR) to control the local printers:

```
:PTRDISABLE LETTER
:LISTPTR
Unit.. Printer.................. Port.....................Status..
0 LETTER \\DENVER4\hpzone3 Paused
1 \\DENVER4\hpzone3 hpzone3 Running
2 LEGAL \\DENVER4\hpzone3 Running
3 \\DENVER4\hpzone2 hpzone2 Running
:PTRENABLE LETTER
:LISTPTR
Unit.. Printer.................. Port.....................Status..
0 LETTER \\DENVER4\hpzone3 Running
1 \\DENVER4\hpzone3 hpzone3 Running
2 LEGAL \\DENVER4\hpzone3 Running
3 \\DENVER4\hpzone2 hpzone2 Running
:
```

Only users with Full Control permissions on a printer can control the printer with PTRDISABLE and PTRENABLE. Check **Permissions** on the **Security** tab of the printers **Properties** sheet to determine who has permissions.

Sending output to a printer

You direct output to a logical printer only by using mode 1 with the SETPTR command, and by specifying LPTR in a UniQuery statement.

In the following example, mode 1 is selected for SETPTR. Then, UniQuery executes a statement directing output to the printer through LPTR.

```
:SETPTR 0,132,60,3,3,1
Unit 0
Width 132
Length 60
Top margin 3
Bot margin 3
Mode 1

Options are:
OK to set parameters as displayed?(enter Y/N) Y
:
:LIST ORDERS PROD_NAME COLOR PRICE SAMPLE 10 LPTR
request id is hpzone3-281 (1 file)
```

Sending output to the _HOLD_ file

You direct output to the _HOLD_ file only by selecting mode 3 with the SETPTR command, and by specifying LPTR in a UniQuery statement.

In the following example, mode 3 is selected with the SETPTR command. The SETPTR BANNER option specifies a _HOLD_ file name of TEST.

```
:SETPTR 0,132,60,3,3,3,BANNER TEST
Unit 0
Width 132
Length 60
Top margin 3
Bot margin 3
Mode 3

Options are:
Banner TEST
OK to set parameters as displayed?(enter Y/N) Y
Hold Entry _HOLD_/TEST
:
:LIST ORDERS PROD_NAME COLOR PRICE SAMPLE 10 LPTR
:
```

Note:

Mode 6 of the SETPTR command directs output to both the printer and the _HOLD_ file.

UDT.OPTIONS 84 determines how UniData handles print jobs that you direct to a _HOLD_ file. When this option is on, UniData displays the name of each _HOLD_ file name to the terminal as a process creates the file. When this option is off, UniData displays a _HOLD_ file name only when a process executes SETPTR or SP.ASSIGN.

Printing a _HOLD_ file using SP.EDIT

To print a _HOLD_ file, use the SP.EDIT command. This command starts a system editor from which you can display, edit, or print *record* in the _HOLD_ file. If you do not enter a record name, UniData prompts for it.

Syntax

SP.EDIT [*record*]

After you enter SP.EDIT and a record ID, UniData prompts for an action code. After each action except quit, UniData returns to the action code prompt (?). If you do not indicate filename, UniData prompts for each file in the _HOLD_ file in sequence, starting with the earliest entry first.

Action codes

The following table lists the SP.EDIT action codes.

Code	Name	Description
T | t	terminal	Displays file on terminal.
F | f	find	Prompts for a search string. After you enter the string, UniData displays the file, beginning with the line containing the string, and then returns to the ? prompt. **Note:** Do not enclose the string in quotation marks.
S | s	spool	Spools the file to the printer.
D | d	delete	Deletes the file
Q | q	quit	Returns to the ECL colon prompt (:).

In the following example, UniData opens a record in the _HOLD_ file, then prompts for an action code. The user responds by entering s to spool the file to the printer:

```
:SP.EDIT TEST

Hold item TEST - (t) terminal (f) find (s) spool (d) delete or (Q) quit ?
s
request id is hpzone3-282 (1 file)

Hold item TEST - (t) terminal (f) find (s) spool (d) delete or (Q) quit ?
q
```

Printing a _HOLD_ file using SPOOL

You can also print a record in the _HOLD_ file using the ECL SPOOL command. The ECL SPOOL command prints the contents of a record or records.

Even though SETPTR mode may be set to 3 or 6 (route to _HOLD_ file), SPOOL directs output only to the print queue or terminal.

Syntax

SPOOL *filename record* [*recordM...recordN*][-O][-T]

The following table lists the SPOOL parameters.

Parameter	Description
filename	The UniData file to be printed.
record	The record ID in *filename*. You can list more than one record by separating the record IDs with a space.

Parameter	Description
-O	Suppresses display of the file name and the record ID in the output.
-T	Displays output to the terminal rather than the printer.

In the following example, UniQuery prints the TEST record in the _HOLD_ file using the SPOOL command:

```
:SPOOL _HOLD_ TEST
request id is hpzone3-283 (1 file)
:
```

Directing output to another file

UniQuery provides two commands, REFORMAT and SREFORMAT, to direct output from a UniQuery statement to another file.

Using REFORMAT

The UniQuery REFORMAT command copies record attributes you specify from one data file to another data file. The destination file must already exist. REFORMAT uses the first attribute named in the UniQuery statement as the record ID in the destination file. The remaining attributes in the UniQuery statement become record attributes in the destination file. UniQuery prompts for the name of the destination file after you enter the REFORMAT command.

Syntax

REFORMAT *filename attributes* [*selection_criteria*]

The following table describes the parameters of the REFORMAT command.

Parameter	Description
filename	Name of the file from which record attributes are selected. You may only specify one file name in a UniQuery statement.
attributes	Specifies the record attributes to construct the new file. The first attribute you specify becomes the record ID of the new record.
selection_criteria	Specifies conditions for selecting or bypassing a record. UniQuery only selects records meeting the selection criteria.

In the following example, the REFORMAT command copies the name, city, state and phone number attributes from the CLIENTS file to a new file called COLORADO_CLIENTS.

```
:REFORMAT CLIENTS NAME CITY STATE PHONE_NUM WITH STATE = "CO"
File name : COLORADO_CLIENTS
:
```

To list the contents of the new file, you must create dictionary records for each attribute if they do not already exist. In the following example, UniQuery lists each attribute in the new file using dictionary items previously created.

```
:LIST COLORADO_CLIENTS CITY STATE PHONE
LIST COLORADO_CLIENTS CITY STATE PHONE 13:52:44 Jun 20 2011 1
COLORADO_CLIENTS  ..................  .  .........
```

```
Ray Parker       Portland             CO 4087695340
                                         4087698834
Glen Asakawa     Colo Spgs.           CO 7198569584
                                         7195868554
2 records listed
:
```

For information about creating dictionary records, see *Using UniData*.

Using SREFORMAT

The UniQuery SREFORMAT command sorts specified records by their record IDs and copies the record attributes from one file to another file. SREFORMAT uses the first attribute named in the UniQuery statement as the record ID in the destination file. The remaining attributes in the UniQuery statement become record attributes in the destination file. UniQuery prompts for the name of the destination file after you enter the SREFORMAT command.

Syntax

SREFORMAT *filename attributes* [*selection_criteria*]

The following table describes the parameters of the SREFORMAT command.

Parameter	Description
filename	Name of the file from which record attributes are selected. You may only specify one file name in a UniQuery statement.
attributes	Specifies the record attributes to construct the new file. The first attribute you specify becomes the record ID of the new record.
selection_criteria	Specifies conditions for selecting or bypassing a record. UniQuery only selects records meeting the selection criteria.

In the following example, the SREFORMAT command copies the name, city, state and phone number attributes from the CLIENTS file to a new file called COLORADO_CLIENTS.

```
:SREFORMAT CLIENTS NAME CITY STATE PHONE_NUM WITH STATE = "CO"
File name : COLORADO_CLIENTS
:
```

Directing output to tape

UniData provides a number of ECL commands for use in managing tape devices. For more information about managing tape devices, see *Administering UniData*.

Defining tape units

The SETTAPE command allows you to define logical tape units in your UniData environment. This command establishes a link between a UniData internal tape unit number and a system-level file. You can use SETTAPE to relate unit numbers to tape devices, or to system-level disk files.

Syntax

SETTAPE *unit.no* [*dn.path.nr*][*dn.path.r*][*blocksize*]

On UniData for Windows platforms, the SETTAPE command establishes a link between a UniData internal tape unit number and an NTFS tape device. You can use SETTAPE to relate unit number to tape devices, or to NTFS or FAT disk files.

Note: If you are using an NTFS tape drive on a Windows platform, you must identify the tape drive with its name in UNC format. If you are using a disk file, you may identify it by its path and file name. The disk file must already exist.

SETTAPE creates an editable ASCII file located in *udthome*/sys/tapeinfo on UniData for UNIX and *udthome*\sys\tapeinfo on UniData for Windows platforms. If you attach a tape and change the block size from that specified in tapeinfo, UniData creates another file in the same directory, tapeatt, which takes precedence over tapeinfo.

Any user can execute SETTAPE *unit.no* to display the current settings for a tape unit. However, you must log in as root on UniData for UNIX or Administrator on UniData for Windows platforms to define a tape unit or modify settings.

Once a tape unit has been defined using SETTAPE, it can be accessed by users in any UniData account on your system. The tape unit definition remains the same unless it is changed.

The following table describes the parameters of the SETTAPE syntax.

Parameter	Description
unit.no	Internal UniData tape unit number. Must be from 0 to 9.
no_rewind_driver	Path and device name of the “no rewind” device driver for *unit*. On UniData for Windows platforms, the driver must be specified in the UNC format if the device is a tape drive.
rewind_driver	Path and name of the “rewind” device driver for *unit*. On UniData for Windows platforms, the driver must be specified in the UNC format if the device is a tape drive.
[*blocksize*]	Tape block size in bytes; must be a multiple of 512. If you do not specify *blocksize*, the default value is 4096.

Note: When defining tape units, be certain to define unit 0. Some of the UniData tape handling commands require unit 0 to be defined so that it can be used as a default.

When you define a tape device or modify a definition, you create or update an entry in the ASCII text file *udthome*/sys/tapeinfo.

Attaching a tape device

Before you can access a tape device, you must attach to it. The ECL T.ATT command attaches a tape drive for exclusive use by the current process.

Syntax

T.ATT [*cn*] [BLKSIZE *block*] [TAPELEN *length*]

The following table lists the T.ATT parameters.

Parameter	Description
nn	Indicates conversion and tape unit. *c* – Conversion code number. Valid conversion codes are: ▪ 0 – Default. No conversion. ASCII is assumed. ▪ 1 – EBCDIC conversion. ▪ 2 – Invert high-bit. ▪ 3 – Swap bytes. *n* – Tape unit number. UniData allows up to 10 unit numbers, 0–9. If you do not indicate the tape unit number, UniData uses tape unit 0 (zero). Do not separate the conversion code from the tape unit with a space.
BLKSIZE *block*	Indicates block size. *block* is a valid block size. If you do not stipulate BLKSIZE, UniData uses the block size set by the SETTAPE command.
TAPELEN *length*	Indicates a tape length for multi-reel tape processing. *length* is the desired tape length in megabytes. **Note:** TAPELEN applies only to tapes created in UniData. UniData cannot read multi-reel TDUMP tapes made on legacy systems.

In the following example, UniData attaches tape unit 4 without indicating a block size. (For the block size, UniData uses the block size set by the SETTAPE command.) The T.STATUS command displays the status of the tape devices.

```
:T.ATT 4
tape unit 4 blocksize = 16384.
:T.STATUS
UNIT STATUS UDTNO USER CHANNEL ASSIGNED
NUMBER NAME NAME BLOCKSIZE
1 AVAILABLE
2 AVAILABLE
3 AVAILABLE
5 AVAILABLE
8 AVAILABLE
4 ASSIGNED 3 root /tmp/diskfile1 16384
:
```

Remember the following points about T.ATT.

- You cannot attach a tape unit with T.ATT unless the unit was previously defined with SETTAPE.
- You can execute T.ATT successive times to change the tape blocksize and also the tape length. If you don't specify BLKSIZE, T.ATT uses the default tape blocksize specified in SETTAPE.
- Only one process can attach a tape unit at any time. You can attach more than one tape unit to a single process, but you cannot attach the same tape unit to more than one process.
- You can use the ECL T.STATUS command to list all defined tape units, and see which ones are attached and which are available.

Writing records to tape

UniData provides the T.DUMP command to write records to tape. The ECL T.DUMP command copies the contents of a file to a tape that was attached with the T.ATT command. UniData writes an end-of-file mark at the end of the file.

Syntax

```
T.DUMP [DICT] filename [MU cn]
[record[recordM...recordN]|select_criteria] [[PICK | pick] [HDR.SUP]]
```

T.DUMP works with an active select list. If you wish to copy a sorted subset of records, create a select list before using T.DUMP. If a record ID is included in a saved list that does not exist in the file, UniData displays a message that the record was not found and not copied.

Before you can execute any tape commands, the tape unit must be configured.

Note: UDT.OPTIONS 50 allows you to choose the ASCII characters used as the end-of-record mark. When this option is on, UniData uses character 251, a UniData text mark. When this option is off, UniData uses character 254, an attribute mark, followed by the text mark. This feature provides compatibility with Pick on Ultimate systems.

Tip: Due to the differences in Pick operating systems and manufactured tapes, we suggest that you use the HDR.SUPP keyword when using the T.DUMP command, and when using the Pick T-LOAD command to avoid inconsistencies in tape labels.

The following table lists the T.DUMP parameters.

Parameter	Description
DICT	Indicates the dictionary portion of the file. If you do not stipulate DICT, UniData copies only the data portion of the file.
filename	The UniData file to be copied.
MU *cn*	Indicates conversion and tape unit. *c* – Conversion code number. Valid conversion codes are: ▪ 0 – Default. No conversion. ASCII is assumed. ▪ 1 – EBCDIC conversion. ▪ 2 – Invert high-bit. ▪ 3 – Swap bytes. *n* – Tape unit number. UniData allows up to 10 unit numbers, 0–9. If you do not indicate the tape unit number, UniData uses tape unit 0 (zero). Do not separate the conversion code from the tape unit with a space.
record	The records within *filename* to copy.
select_criteria	The record IDs, a select list of the record IDs, or a selection condition. If you do not indicate *select_criteria*, UniData copies all records within *filename*.
PICK | pick	Produces a tape that can be loaded on a Pick system. To avoid incompatibility in tape label format, suppress the label by including HDR.SUP.
HDR.SUP	Suppresses the generation of a tape label.

The following example shows some typical outputs when a process with tape unit 4 attached executes the ECL `T.DUMP` command:

```
:T.DUMP DICT INVENTORY MU 04
16 record(s) dumped to tape
:SELECT VOC WITH F1 = PA

9 records selected to list 0.

>T.DUMP VOC
9 record(s) dumped to tape
:T.DUMP INVENTORY MU 01
Unit 1 not attached yet.

:T.DUMP INVENTORY MU 09
Unit 9 is not initialized yet.
:T.STATUS

   UNIT      STATUS     UDTNO    USER          CHANNEL          ASSIGNED
   NUMBER                        NAME          NAME             BLOCKSIZE

  1        AVAILABLE
  2        AVAILABLE
  3        AVAILABLE
  5        AVAILABLE
  8        AVAILABLE
  4        AVAILABLE
  0        AVAILABLE
:
```

Detaching the tape device

When you are done with a tape device, use the `T.DET` command to release the device so that another process can use it. If you have attached more than one device, you need to release each one separately. If you have attached only one, the `T.DET` command releases the one you have attached.

Syntax

`T.DET` `[n]`

n is the tape unit number. UniData allows up to 10 unit numbers, 0 to 9.

In the following example, UniData releases tape unit 4:

```
:T.DET 4
```

For more information about individual tape commands, see the *UniData Commands Reference*.

Chapter 7: UniQuery security

This chapter provides information about creating field level security and using UniBasic subroutines for further security. For information about UniData security, see *UniData Security Features*.

Creating field level security

UniData includes functionality to determine UniQuery access on a field-by-field basis.

System Administrators can set privileges for UniQuery access at the file level or the field level, for a single user or for all users in the UNIX group, by creating a QUERY.PRIVILEGE table in a specified format and adding records to that table.

You may also set a default for your system, defining all files as OPEN or SECURE. In an OPEN system, the ability to access a file or a field with UniQuery is a function of file permissions, other UniData security implementations, and privileges granted using the QUERY.PRIVILEGE table. In a SECURE system, unless privileges are granted in the QUERY.PRIVILEGE table, users cannot access files through UniQuery, regardless of file permissions or other implementations.

Points to remember about field level security

- Implementing and maintaining field level security is a completely manual process. You must create and populate the QUERY.PRIVILEGE file manually.
- ECL commands, such as `CREATE.FILE`, `DELETE.FILE`, and `CNAME` do not update the QUERY.PRIVILEGE table.
- ECL commands are not affected by the UniQuery security.
- The UniQuery `MODIFY` command is not affected by the UniQuery security feature. The security is imposed when a user attempts to `SELECT`.
- A default of OPEN or SECURE affects all UniData accounts that share the same *udthome*. You cannot define some accounts as OPEN and some as SECURE.
- Privileges granted on a file are not automatically applied to its dictionary. In other words, if a user has ALL access to the INVENTORY file and its dictionary, you must consider D_INVENTORY as well. If the system default is OPEN, the user can access D_INVENTORY. Otherwise, if you want the user to access D_INVENTORY, you need a QUERY.PRIVILEGE record for D_INVENTORY as well.

The QUERY.PRIVILEGE file

UniQuery security depends on the existence of a special file called QUERY.PRIVILEGE, which must be located in *udthome*`/sys` on UniData for UNIX or *udthome*`\sys` on UniData for Windows Platforms. If this file does not exist, UniQuery functions as it has previously, with no field-level security.

Warning: If you create the QUERY.PRIVILEGE file, but do not populate the file with any records, UniData will not allow any user to access any files on the system through UniQuery.

When you install UniData, the UniQuery security is not implemented. If you wish to turn this feature on, you must create QUERY.PRIVILEGE and D_QUERY.PRIVILEGE manually.

Records in the QUERY.PRIVILEGE file grant the SELECT privilege to users or groups of users, at the file level or the field level. Each QUERY.PRIVILEGE record has one attribute. The dictionary of the QUERY.PRIVILEGE file contains four items.

Following is a sample of the dictionary of the QUERY.PRIVILEGE file:

```
:LIST DICT QUERY.PRIVILEGE
LIST DICT QUERY.PRIVILEGE BY TYP BY @ID TYP LOC CONV NAME FORMAT
 SM ASSOC 15:20:26 Jun 02 2011 1
@ID............ TYP LOC.......... CONV NAME........... FORMAT SM
 ASSOC.....

@ID             D               0      QUERY.PRIVILEGE 50L    S
PRIV            D               1      PRIVILEGES      5R     M
FULLPATH        V   FIELD(@ID,'*' File Path 25T S
                    ,2)
USERNAME        V   FIELD(@ID,'*' User Name 25T S
                    ,1)
4 records listed
```

The following table describes each QUERY.PRIVILEGE attributes.

Attributes	Description
@ID	Data attribute that defines the user or domain and the file for which you are setting privileges. @ID takes the form *username*path*, or PUBLIC* *path*. On UniData for Windows Platforms, the @ID can also be *domain \username*pathname*. If you are setting up a system default, @ID is DEFAULT.
PRIV	Data attribute that indicates the attributes to which you are granting privileges by location. PRIV is a multivalued attribute. To grant privileges to all attributes in a table, set PRIV to ALL. If you are setting a system default, set PRIV to OPEN to grant privileges. To restrict privileges to every attribute in a file, set PRIV to SECURE.
FULLPATH	Virtual attribute formula that designates the full path of the file affected by PRIV. This formula has the format FIELD(@ID,'*',2).
USERNAME	Virtual attribute formula that designates the user affected by PRIV. This formula has the format FIELD(@ID,'*',1).

Note: You can customize the length of the dictionary attributes in the QUERY.PRIVILEGE file. The length of @ID should be sufficient to contain the longest user name and the longest absolute path for a UniData file on your system. FULLPATH and USERNAME should be long enough to handle the longest absolute path and longest user name, respectively.

The following table shows a very simple example of a QUERY.PRIVILEGE file on UniData for UNIX:

```
:LIST QUERY.PRIVILEGE PRIV
LIST QUERY.PRIVILEGE PRIV 10:48:15 Apr 25 2011 1
          Authorized
QUERY.PRIVILEGE........................ Locations.

claireg*/disk1/ud72/demo/INVENTORY       1 2 3 4 5
                                         11
DEFAULT                                        OPEN
claireg*/disk1/ud72/demo/CLIENTS                ALL
3 records listed
```

The next example shows a simple QUERY.PRIVILEGE data file on UniData for Windows NT:

```
:LIST QUERY.PRIVILEGE PRIV
LIST QUERY.PRIVILEGE PRIV 10:50:22 Apr 25 2011 1
                                          Authorized
QUERY.PRIVILEGE........................ Locations.

claireg*\UniData71\Demo\INVENTORY 1 2 3 4 5
                                           11
DEFAULT                                         OPEN
claireg*\UniData71\demo\CLIENTS ALL
3 records listed
```

Both of the QUERY.PRIVILEGE files mean:

- Except for INVENTORY and CLIENTS, which are in the demo database, all users have privileges to query all files in all accounts that share the same *udthome*.
- User claireg can query the fields in positions 1, 2, 3, 4, 5 and 11 only in the INVENTORY file. No other user can query this file.
- User claireg can query any field in the CLIENTS file. No other user can query the CLIENTS file.

UniQuery processing

If you have turned on the security feature by creating and populating the QUERY.PRIVILEGE file, every time a user logs in to UniData their udt process reads the contents of QUERY.PRIVILEGE and stores the information for reference. Then, when a user attempts a UniQuery access, UniData checks the stored information using the following steps:

1. Check for system privileges granted to the user group or domain.

   On UniData for UNIX, if the user's UNIX group has sufficient privileges for the requested access, allow the access. On UniData for Windows Platforms, if the user's domain has sufficient privileges for the requested access, allow the access. Otherwise, proceed to step 2.
2. Check for privileges granted specifically to the user.

   If the user has sufficient privileges for the requested access, then allow the access. Otherwise, proceed to step 3.
3. Check for privileges granted to PUBLIC.

   Privileges granted to PUBLIC apply to all system users. If PUBLIC has sufficient privileges for the requested access, grant the access. Otherwise, proceed to step 4.
4. Check for a DEFAULT entry

   If there is a DEFAULT record in QUERY.PRIVILEGE, and if the default is set to OPEN, allow the requested access. If there is no DEFAULT, or if the DEFAULT is SECURE, disallow the access, displaying the following message: "No privilege on *filename.*"

Turning on field-level security

Complete the following steps to implement the UniQuery field-level security feature:

1. Log on to your system as root or Administrator. UniData must be running. Users do not need to log off.
2. Create QUERY.PRIVILEGE.

Change your working directory to *udthome*/sys on UniData for UNIX or *udthome*\sys on UniData for Windows platforms, and enter udt (or udtts if you are using device licensing) to start a UniData session. Use the ECL CREATE.FILE command as follows:

```
:CREATE.FILE QUERY.PRIVILEGE 101
Create file D_QUERY.PRIVILEGE, modulo/1,blocksize/1024
Hash type = 0
Create file QUERY.PRIVILEGE, modulo/101,blocksize/1024
Hash type = 0
Added "@ID", the default record for UniData to DICT QUERY.PRIVILEGE.
```

Make the QUERY.PRIVILEGE file a static hashed file.

3. Set permissions on QUERY.PRIVILEGE.

   The QUERY.PRIVILEGE file and its dictionary should be read-only to all users except root on UniData for UNIX or Administrator on UniData for Windows platforms.

4. Edit the dictionary.

   Use UniEntry, AE, or ED to edit D_QUERY.PRIVILEGE. The dictionary should look like the following example:

```
@ID............ TYP LOC.......... CONV NAME........... FORMAT SM
ASSOC.....

@ID             D               0      QUERY.PRIVILEGE 40L    S
PRIV            D               1      PRIVILEGES      5R     M
FULLPATH        V   FIELD(@ID,'*' File Path 25T S
                    ,2)
USERNAME        V   FIELD(@ID,'*' User Name 25T S
                    ,1)
```

   Note: You can customize the format for the dictionary items to specify lengths for the attributes that match your system.

5. Add records to QUERY.PRIVILEGE.

   For this step, you may prefer to have users logged out of UniData. As you add records to the QUERY.PRIVILEGE file, users logging into UniData will access whatever records are present at the time they log in, which may cause unexpected results.

   Use AE, UniEntry, or ED to populate the QUERY.PRIVILEGE file.

Remote items

You can further customize security by replacing some command entries in your VOC file with remote items. A remote item (VOC record type R) allows a record definition to be stored in a location other than the VOC file. You can substitute remote items for sentences, paragraphs, verbs (commands), locally cataloged programs, or menus. Refer to *Using UniData* for more information about R-type items.

R type items allow you to customize security in two ways:

- You can use a remote item as a pointer to a location with different file permissions from the current account, limiting access to the item.
- You can supply a “security routine” for the remote item. R type items enable you to name a cataloged subroutine that is executed when a user invokes the remote item. The subroutine must have one argument, and return a value of 1 (true) or 0 (false). When a user invokes a remote item with a security subroutine, the remote item will not execute unless the subroutine returns 1 (true).

The following screen shows an example of a remote item you could create for the ECL `LIST` command:

```
:LIST VOC F1 F2 F3 F4 WITH @ID = LIST 11:05:23 Apr 24 2011 1
VOC..... F1..... F2..... F3..... F4.....

LIST      R          OTHER_LIST      LIST            SECTEST2
1 record listed
```

When a user executes the `LIST` command, UniData executes a security subroutine called SECTEST2. If that subroutine returns a value of “true”, UniData executes the item called LIST in a file called OTHER_VOC.

The next screen shows the security subroutine:

```
:AE BP SECTEST2
Top of "SECTEST2" in "BP", 4 lines, 66 characters.
*--: P
001: SUBROUTINE SECTEST2(OKAY)
002: COMMON /SECUR/ VALID
003: OKAY = VALID
004: RETURN
Bottom.
```

In this example, the subroutine obtains the value of VALID from named COMMON. The value can be set by another subroutine or program. The following screen shows what happens if VALID is zero (false) and a user executes the ECL `LIST` command:

```
:LIST VOC WITH F1 = PA
Not a verb
LIST
```

The next screen shows what happens if VALID is 1 (true):

```
:LIST VOC WITH F1 = PA
LIST VOC WITH F1 = PA 11:13:27 Apr 24 2011 1
VOC.....
ECLTYPE
CP
CT
SP.OPEN
listdict
LISTDICT
6 records
```

Chapter 8: Null value handling

This chapter examines the UniQuery handling of the null value in UniQuery. See *Using UniData* for an overview of null value handling across UniData products.

Introduction to the null value

With null value handling on, the null value represents an unknown value, making the UniData RDBMS more compliant with the standards defined by ANSI SQL '92. Compliance improves compatibility with client and desktop tools.

Turning on null value handling

Turn on null value handling with the UniData configuration parameter NULL_FLAG. UniData must be stopped and restarted for `udtconfig` parameters to take effect.

Turning off null value handling

After turning null value handling on, we recommend that you not turn it off, as the null character may have been introduced to your data. If you must turn null value handling off, be sure to check your data and convert the null value to another string (using a UniBasic program or virtual attribute) before attempting to execute queries, virtual attributes, or UniBasic programs.

Warning: If null value handling is turned off, UniData may produce unpredictable results when it encounters the null value in data or as a parameter in a virtual attribute.

Representing the null value

If you accept the default language group when installing UniData, the null value is represented by the ASCII character 129. When you change language groups, a different character may be assigned to represent the null value. For this reason, we recommend that you use the @NULL variable to represent the null value in UniData and UniQuery. See *Administering UniData on UNIX* or *Administering UniData on Windows Platforms* for instructions on setting the UniData configuration parameter NULL_FLAG and selecting a language group.

Tip: The ASCII character that represents the null value is nonprinting. Use the UniData command NVLMARK to specify a printable character to represent the null value for display or printing. Use the UniData SQL command `NVL`, or a UniBasic conversion function in a virtual attribute to permanently convert the null value to another string.

Inserting the null value

You can insert the null value using the following tools:

- A text editor — Enter the ASCII character.
- The UniData Alternative Editor (AE) — See AE online help, or *Using UniData*.

- UniEntry — You cannot use this tool to enter the null value.
- UniData SQL — Use the keyword NULL.
- A UniBasic program — Can be called by a virtual attribute, paragraph, or trigger.

The null value in UniQuery

The following table summarizes the effects of the null value on UniQuery operations.

Operation	Effect
Aggregation	Regardless of whether null value handing is turned on, UniData ignores all nonnumeric values, including the null value, when calculating aggregate functions: ▪ SUM — The result is not affected, but the number of records listed is. See the example. ▪ TOTAL — The result is not affected, but the number of records listed is. See the example. ▪ AVERAGE — The result IS affected, because nulls contribute to the number of values used to calculate average. ▪ PERCENT — Null values are listed as contributing 0% to the TOTAL. The keyword NO.NULLS, used with AVERAGE refers to empty strings, not null values.
Numeric and Date	UniData interprets the null value as 0; arithmetic operations produce a result of the null value.
Conditional	Comparisons with the null value yields a false result; the operation returns a zero.
Sorting and Indexing	The null value is the smallest value, lower than all negative numbers. It is sorted first in a descending sort in the absence of selection criteria.
Printing and Displaying	Displays as a space by default; print character can be changed with the UniData configuration parameter NVLMARK.
Selecting	The keywords IS NULLVAL in a WITH or WHEN clause selects the null value.

Examples

Data values in the demo database have been changed before executing these examples to facilitate illustration of the effects of the null value on UniQuery operations.

Aggregation operations

Aggregation operations consist of SUM, TOTAL, AVERAGE, and PERCENT. UniData ignores nonnumeric values when performing aggregate operations whether null value handling is turned on or off. UniData also ignores the null value when calculating subtotals, subpercents, and subaverages.

Examples

This series of examples demonstrates the handling of null values and empty strings in aggregate functions. First, here are the records we are going to be working with. INVENTORY records 55020 and 55050 contain empty strings in the PRICE attribute.

```
:LIST INVENTORY PRICE WHEN PRICE < 10
LIST INVENTORY PRICE WHEN PRICE < 10 14:43:46 Apr 16 2011 1
INVENTORY. Price.....

55020
55050
55090           $8.99
55060           $5.99
56080           $3.99

                $3.99
                $3.99
51070           $9.99
                $9.99
                $9.99
                $9.99
55000           $7.95
55040           $9.99
55010           $8.99
9 records listed
```

In this example, the PRICE attribute in record 55050 has been changed to the null value, but an empty string is still stored in PRICE for record 55020. As expected, when `PERCENT` is calculated, the null value in 55050 is NOT included, but the empty string in 55020 is:

```
:LIST INVENTORY TOTAL PRICE PERCENT PRICE WHEN PRICE < 10
LIST INVENTORY TOTAL PRICE PERCENT PRICE WHEN PRICE < 10 15:30:29 Apr 16 2011 1
INVENTORY. Price..... Price.....

55020                          0.00
55090           $8.99          9.58
55060           $5.99          6.38
56080           $3.99          4.25
                $3.99          4.25
                $3.99          4.25
51070           $9.99         10.65
                $9.99         10.65
                $9.99         10.65
                $9.99         10.65
55000           $7.95          8.47
55040           $9.99         10.65
55010           $8.99          9.58
           ========== ==========
               $93.84        100.00
8 records listed
```

Before executing this next example, we changed values in the PRICE attribute in records 55050 and 55020 of the INVENTORY file to empty strings. Compare this result from the preceding example, in which PRICE contained the null value in record 55050. Because empty strings are included in calculation of averages, the number of records increases from 7 to 9, and the average decreases from $7.82 to $6.70.

```
:LIST INVENTORY AVERAGE PRICE WHEN PRICE < 10
LIST INVENTORY AVERAGE PRICE WHEN PRICE < 10 11:13:18 Apr 16 2011 1
INVENTORY. Price.....
```

```
55020
55050
55090           $8.99
55060           $5.99
56080           $3.99
                $3.99
                $3.99
51070           $9.99
                $9.99
                $9.99
                $9.99
55000           $7.95
55040           $9.99
55010           $8.99
           ==========
AVERAGE         $6.70
9 records listed
```

With empty strings in the PRICE attribute of records 55020 and 55050, UniData includes the values when calculating `SUM` (Note the last line, 9 records summed.)

```
:SUM INVENTORY PRICE WHEN PRICE < 10
Total PRICE = $93.84
9 records summed
```

However, with null values in these attributes, they are excluded in the `SUM` (7 records summed):

```
:SUM INVENTORY PRICE WHEN PRICE < 10
SUM INVENTORY PRICE WHEN PRICE < 10
Total PRICE = $93.84
7 records summed
```

Numeric and date calculations

UniData supports the arithmetic operators +, /, *, -, as well as other mathematic symbols.

Null value handling on

When arithmetic operations encounter the null value, they produce a result of the null value.

Examples

Before executing this example, the value in INV_DATE for record 50020 in the INVENTORY file was changed to the null value; for record 50050, this attribute was changed to 0. Record 50020 is not selected, because the result of the null value + 1 is the null value. Record 50050 is selected, however, because 0 + 1 is 1.

```
:SORT INVENTORY WITH ID BETWEEN 50000 51000 AND EVAL "INV_DATE+1"
 > "0" INV_DATE
SORT INVENTORY WITH ID BETWEEN 50000 51000 AND EVAL "INV_DATE+1" >
 "0" INV_DATE 15:02:51 Apr 19 2011 1
          Inventory
INVENTORY. Date......

50000           10196
```

```
50010           10250
50030           10250
50040           10250
50050               0
50060            9997
50070           10189
50080           10209
50090           10247
51000           10005
10 records listed
```

The following virtual attribute, DATE_PLUS, adds 1 to INV_DATE (the alternate editor is used to display the virtual attribute):

```
:AE DICT INVENTORY DATE_PLUS
Top of "DATE_PLUS" in "DICT INVENTORY", 6 lines, 27 characters.
*--: P
001: V
002: INV_DATE+1
003: Date+1
004:
005: 10R
006: S
Bottom.
*--:
```

This next UniQuery statement displays the preceding virtual attribute. Notice that INV_DATE for record 50020 is X, because the `udtconfig` parameter NVLMARK is set to display X instead of the nonprinting null value. This example demonstrates that the null value in INV_DATE for record 50020 plus 1 is the null value:

```
:LIST INVENTORY INV_DATE DATE_PLUS WHEN ID BETWEEN 50000 51000LIST INVENTORY
INV_DATE DATE_PLUS WHEN ID BETWEEN 50000 51000
 16:07:15 Apr 19 2011 1
           Inventory
INVENTORY. Date......  .........

50070           10189      10190
51000           10005      10006
50040           10250      10251
50010           10250      10251
50020               X          X
50050               0          1
50080           10209      10210
50030           10250      10251
50060            9997       9998
50090           10247      10248
50000           10196      10197
11 records listed
```

Null value handling off

The character that would represent the null value is processed as is any other nonnumeric value. UniQuery replaces it with 0.

This example demonstrates UniData's handling of ASCII character 129 with null value handling turned off (it is converted to 0 for numeric calculations).

```
:SORT INVENTORY WITH EVAL "INV_DATE+1" > "0" INV_DATE BY INV_DATE
```

```
SORT INVENTORY WITH EVAL "INV_DATE+1" > "0" INV_DATE BY INV_DATE
14:23:49 Apr 19 2011 1
           Inventory
INVENTORY. Date......

50020
50050               0
10140            9735
54090            9865
54070            9885
50060            9997
51000           10005
51020           10006
...
```

The null value in conditional tests

Null value handling on

Use the keywords IS [NOT] NULL to test for the null value.

Before executing the next examples, we added a value to the PRICE attribute in record 50050. The multivalued attribute now contains two values, 0 and 99999. In record 50020, we also added a value. This attribute now contains 1349999, ^129 (the ASCII character that represents the null value for the English language group), 1349999, and 1349999. We also created a paragraph, TRYNULL, that selects all existing multivalues in attribute PRICE, and SORTs those multivalues in ascending order by PRICE.

As you can see, record 55020 is NOT selected because any test of the null value produces a negative result:

```
:TRYNULL
SORT INVENTORY PRICE WHEN PRICE AND WHEN ID BETWEEN 50000 51000
BY.EXP PRICE 16:52:07 Apr 19 2011 1
INVENTORY. Price.....

50050           $0.00
51000          $59.99
51000          $59.99
51000          $59.99
50090         $799.99
50050         $999.99
```

To retrieve the null value use the keywords IS NULLVAL, as in the following example:

```
:LIST INVENTORY PRICE WHEN PRICE IS NULLVAL
LIST INVENTORY PRICE WHEN PRICE IS NULLVAL 16:59:20 Apr 19 2011 1
INVENTORY. Price.....

55050                      X
1 record listed
```

Null value handling off

All characters are tested according to their ASCII value. The keyword IS NULLVAL in paragraphs or virtual attributes produces a runtime error.

Examples

As you can see, record 55020, witch contains ASCII value 129 in the PRICE attribute is selected.

```
:TRYNULL
SORT INVENTORY PRICE WHEN PRICE AND WHEN ID BETWEEN 50000 51000
 BY.EXP PRICE 10:33:34 Apr 19 2011 1
INVENTORY. Price.....

50020
50050           $0.00
51000          $59.99
51000          $59.99
51000          $59.99
50090         $799.99
50050         $999.99
...
```

In this example, we changed the values in attribute INV_DATE. For record 50020, INV_DATE is now ASCII character 129. For record 50050, it is now 0. We also removed the conversion for display of INV_DATE so that sorting and selection based on this attribute is performed on the internal date. ASCII character 129 is selected in this case, and is sorted above negative numbers, but below 0:

```
:SORT INVENTORY INV_DATE WHEN INV_DATE BY INV_DATE
SORT INVENTORY INV_DATE WHEN INV_DATE BY INV_DATE 11:21:59 Apr 19
 2011 1
           Inventory
INVENTORY. Date......

10009          -72495
50020
50050               0
10140            9735
54090            9865
54070            9885
50060            9997
...
```

Sorting and indexing

Null value handling on

The null value is sorted as the lowest number, below all negative numbers.

Before executing the next examples, we added a value to the PRICE attribute in record 50050. The multivalued attribute now contains two values, 0 and 99999. In record 50020, we also added a value. This attribute now contains 1349999, ^129 (the ASCII character that represents the null value for the English language group), 1349999, and 1349999. We also created a paragraph, TRYNULL, that selects all existing multivalues in attribute PRICE, and SORTs those multivalues in ascending order by PRICE.

Examples

The null value, represented here by X, is sorted lower than negative numbers:

```
:SORT INVENTORY INV_DATE BY INV_DATE
SORT INVENTORY INV_DATE BY INV_DATE 16:29:24 Apr 19 2011 1
           Inventory
INVENTORY. Date......
```

```
50020              X
10009          -72495
50050              0
10140           9735
54090           9865
54070           9885
50060           9997
51000          10005
...
```

Null value handling off

All characters are sorted by ASCII value.

Examples

The following example demonstrates that, with null value handling off, ASCII character 129 is sorted below 0, but above negative numbers.

```
:SORT INVENTORY INV_DATE BY INV_DATE
SORT INVENTORY INV_DATE BY INV_DATE 09:48:57 Apr 19 2011 1
           Inventory
INVENTORY. Date......

10009          -72495
55020
55050              0
10140           9735
54090           9865
54070           9885
50050           9886
50060           9997
51000          10005
...
```

Printing and displaying the null value

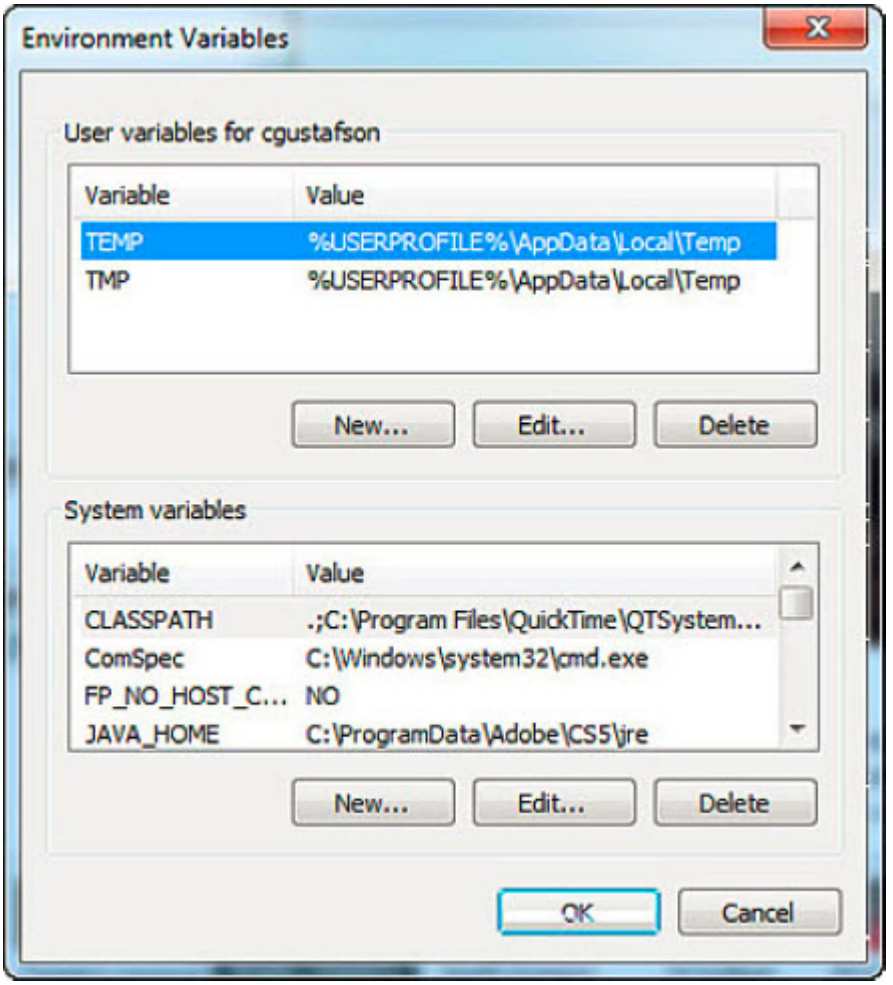
Use the UniData configuration parameter or environment variable NVLMARK to specify a character to represent the null value for print or display purposes. The following syntax is appropriate for setting the environment variable NVLMARK on UniData for UNIX running C shell.

Syntax

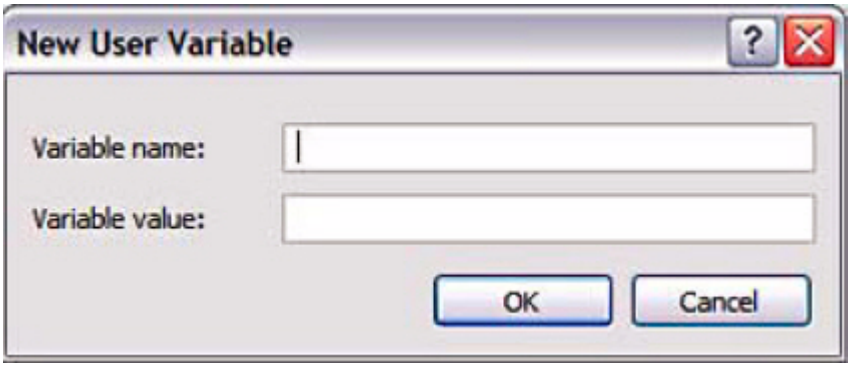
setenv NVLMARK *char*

where *char* is any single printable character.

To set an environment variable on UniData for Windows platforms, from the **Start** menu, click **Control Panel**, double-click **System**, double-click **Advanced system settings**, and then click the **Environment** tab. A window similar to the following appears:



Click **New**. The following dialog box appears:



In the **Variable name** field, enter `NVLMARK`. Enter the value for the NVLMARK in the **Variable Value** field.

Tip: Environment variables remain in effect for the current working session only. For information on setting the environment variables for all sessions, see *Administering UniData on UNIX* or *Administering UniData on Windows Platforms*.

Selecting the null value

Use the keyword IS NULLVAL to check for the null value in a WHEN or WITH clause.

Syntax

```
{LIST | SORT} filename [display_attributes] {WITH | WHEN} [IS NOT | IS]
NULLVAL
```

For more information on IS NULLVAL, see *UniQuery Commands Reference*.

Chapter 9: Creating XML documents

XML for UniData

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.

UniData enables you to receive and create XML documents, and process them through UniBasic, UniData SQL, or UniQuery. In order to work with the XML documents in UniData, 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 `$UDTHOME/sys/` directory. The `U2XMLOUT.map` file sets the flag for all UniData users.

Creating an XML document from UniQuery

You can create an XML document from UniData files through UniQuery. To create an XML document through UniQuery, complete the following steps:

1. If you are the originator of the DTD or XML Schema, use UniQuery 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 128 at the end of this section.
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. You must use the `TOXML` command to create an XML document.

Create the _XML_ file

UniData 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

UniData 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 association of multivalues and multi-subvalues will form a sub-element of the record element using the association *filename_MV.* The multi-subvalue will be a sub-element of the multivalue element. The name of the sub-element, if there is no association, is *fieldname_MV* or *_MS*.
- Within a sub-element, each multivalued field becomes an attribute of the sub-element.
  - Associated multi-subvalued fields become another nested sub-element of the sub-element. The name of this nested sub-element is *association_name*-MS.
  - If there are no associated multi-subvalued fields, the sub-element name is field_name-MV/MS.

This is the default mapping scheme. You can change the default by defining map files in the _XML_ directory.

The following example shows data created in attribute mode:

```
LIST STUDENT LNAME CGA TOXML SAMPLE 1
<?xml version="1.0"?>
<MAIN>
<STUDENT _ID = "521814564" LNAME = "Smith">
 <CGA-MV SEMESTER = "FA93">
    <CGA-MS COURSE_NBR = "CS130" COURSE_NAME = "Intro to Operating Systems"
COURSE_GRD = "A" COURSE_HOURS = "5" TEACHER = "James"/>
    <CGA-MS COURSE_NBR = "CS100" COURSE_NAME = "Intro to Computer Science"
COURSE_GRD = "B" COURSE_HOURS = "3" TEACHER = "Gibson"/>
    <CGA-MS COURSE_NBR = "PY100" COURSE_NAME = "Introduction to Psychology"
COURSE_GRD = "B" COURSE_HOURS = "3" TEACHER = "Masters"/>
 </CGA-MV>
 <CGA-MV SEMESTER = "SP94">
    <CGA-MS COURSE_NBR = "CS131" COURSE_NAME = "Intro to Operating Systems"
COURSE_GRD = "B" COURSE_HOURS = "5" TEACHER = "Aaron"/>
    <CGA-MS COURSE_NBR = "CS101" COURSE_NAME = "Intro to Computer Science"
COURSE_GRD = "B" COURSE_HOURS = "4" TEACHER = "Gibson"/>
    <CGA-MS COURSE_NBR = "PE220" COURSE_NAME = "Racquetball" COURSE_GRD = "A"
COURSE_HOURS = "3" TEACHER = "Fisher"/>
 </CGA-MV>
</STUDENT>

</MAIN>
:
```

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 and multi-subvalued 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 that may contain multi-subvalued sub-elements. 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 110.
- By default, UniData converts text marks to an empty string.

Specify that you want to use element-centric mapping by using the ELEMENTS keyword in the UniQuery 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"?>
<MAIN>
<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>PY100</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>CS131</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>
</MAIN>
```

Displaying empty values in multivalued fields in an association

UniData displays empty values in multivalued fields belonging to an association depending on the setting of the Matchelement field in the `U2XMLOUT.map` file.

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. 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.

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 12:59:04
Aug 29 2011 1

STUDENT       123-45-6789
Last Name    Martin
First Name   Sally
Course # Grade Course Name               Term
PY100          Introduction to Psychology SP94
PE100    C     Golf - I

STUDENT       987-65-4321
Last Name    Miller
First Name   Susan
Course # Grade Course Name               Term
EG110    C     Engineering Principles     FA93
MA220          Calculus- I
PY100    B     Introduction to Psychology
EG140    B     Fluid Mechanics            SP94
EG240    B     Circut Theory
MA221          Calculus - II

2 records listed
```

Notice that two of the GRADE fields are empty, while their associated values for COURSE # and COURSE NAME are not.

When Emptyattribute is set to 1, the missing values for COURSE_GRD display as empty values in the XML documents, as shown in the following example:

```
LIST STUDENT CGA TOXML XMLMAPPING student.map
<?xml version="1.0"?>
<MAIN>
<STUDENT _ID = "123456789">
 <Term SEMESTER = "SP94">
    <Courses_Taken COURSE_NBR = "PY100"
COURSE_NAME = "Introduction to Psychology"
COURSE_HOURS = "3"
TEACHER ="Masters"/>
    <Courses_Taken COURSE_NBR = "PE100" COURSE_NAME = "Golf - I" COURSE_GRD = "C"
COURSE_HOURS = "3" TEACHER = "Fisher"/>
 </Term>
</STUDENT>

<STUDENT _ID = "987654321">
 <Term SEMESTER = "FA93">
    <Courses_Taken COURSE_NBR = "EG110" COURSE_NAME = "Engineering Principles"
COURSE_GRD = "C" COURSE_HOURS = "5" TEACHER = "Carnes"/>
    <Courses_Taken COURSE_NBR = "MA220"
COURSE_NAME = "Calculus-I"
COURSE_HOURS = "5" TEACHER = "Otis"/>
<Courses_Taken COURSE_NBR = "PY100" COURSE_NAME = "Introduction to Psychology"
 COURSE_GRD = "B" COURSE_HOURS = "3" TEACHER = "Masters"/>
 </Term>
 <Term SEMESTER = "SP94">
    <Courses_Taken COURSE_NBR = "EG140" COURSE_NAME = "Fluid Mechanics"
COURSE_GRD = "B" COURSE_HOURS = "3" TEACHER = "Aaron"/>
    <Courses_Taken COURSE_NBR = "EG240" COURSE_NAME = "Circut Theory"
COURSE_GRD = "B" COURSE_HOURS = "3" TEACHER = "Carnes"/>
    <Courses_Taken COURSE_NBR = "MA221"
COURSE_NAME = "Calculus - II"
COURSE_HOURS = "5" TEACHER = "Otis"/>
 </Term>
</STUDENT>

</MAIN>
:
```

When Matchelement is set to 1, the missing values for COURSE _GRD, `<COURSE_GRD></COURSE_GRD>`, display as an empty value in the XML document, as shown in the following example:

```
LIST STUDENT LNAME CGA TOXML ELEMENTS

<?xml version="1.0"?>
<MAIN>
<STUDENT>
 <_ID>123456789</_ID>
 <LNAME>Martin</LNAME>
 <CGA-MV>
    <SEMESTER>SP94</SEMESTER>
   <CGA-MS>
      <COURSE_NBR>PY100</COURSE_NBR>
      <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
      <COURSE_GRD></COURSE_GRD>
      <COURSE_HOURS>3</COURSE_HOURS>
      <TEACHER>Masters</TEACHER>
```

```
    </CGA-MS>
    <CGA-MS>
       <COURSE_NBR>PE100</COURSE_NBR>
       <COURSE_NAME>Golf - I</COURSE_NAME>
       <COURSE_GRD>C</COURSE_GRD>
       <COURSE_HOURS>3</COURSE_HOURS>
       <TEACHER>Fisher</TEACHER>
    </CGA-MS>
 </CGA-MV>
</STUDENT>

<STUDENT>
 <_ID>987654321</_ID>
 <LNAME>Miller</LNAME>
 <CGA-MV>
    <SEMESTER>FA93</SEMESTER>
    <CGA-MS>
       <COURSE_NBR>EG110</COURSE_NBR>
       <COURSE_NAME>Engineering Principles</COURSE_NAME>
       <COURSE_GRD>C</COURSE_GRD>
       <COURSE_HOURS>5</COURSE_HOURS>
       <TEACHER>Carnes</TEACHER>
    </CGA-MS>
    <CGA-MS>
       <COURSE_NBR>MA220</COURSE_NBR>
       <COURSE_NAME>Calculus- I</COURSE_NAME>
       <COURSE_GRD></COURSE_GRD>
       <COURSE_HOURS>5</COURSE_HOURS>
       <TEACHER>Otis</TEACHER>
    </CGA-MS>
    <CGA-MS>
       <COURSE_NBR>PY100</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>EG140</COURSE_NBR>
       <COURSE_NAME>Fluid Mechanics</COURSE_NAME>
       <COURSE_GRD>B</COURSE_GRD>
       <COURSE_HOURS>3</COURSE_HOURS>
       <TEACHER>Aaron</TEACHER>
    </CGA-MS>
    <CGA-MS>
       <COURSE_NBR>EG240</COURSE_NBR>
       <COURSE_NAME>Circut Theory</COURSE_NAME>
       <COURSE_GRD>B</COURSE_GRD>
       <COURSE_HOURS>3</COURSE_HOURS>
       <TEACHER>Carnes</TEACHER>
    </CGA-MS>
    <CGA-MS>
       <COURSE_NBR>MA221</COURSE_NBR>
       <COURSE_NAME>Calculus - II</COURSE_NAME>
       <COURSE_GRD></COURSE_GRD>
       <COURSE_HOURS>5</COURSE_HOURS>
       <TEACHER>Otis</TEACHER>
    </CGA-MS>
 </CGA-MV>
```

```
</STUDENT>

</MAIN>
:
```

This is the default behavior.

When Matchelement is set to 0, the missing value for COURSE_GRD, `<COURSE_GRD></COURSE_GRD>`, is ignored in the XML document, as shown in the following example:

```
LIST STUDENT LNAME CGA TOXML ELEMENTS
<?xml version="1.0"?>
<MAIN>
<STUDENT>
 <_ID>123456789</_ID>
 <LNAME>Martin</LNAME>
 <CGA-MV>
    <SEMESTER>SP94</SEMESTER>
   <CGA-MS>
      <COURSE_NBR>PY100</COURSE_NBR>
      <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
      <COURSE_HOURS>3</COURSE_HOURS>
      <TEACHER>Masters</TEACHER>
   </CGA-MS>
   <CGA-MS>
      <COURSE_NBR>PE100</COURSE_NBR>
      <COURSE_NAME>Golf - I</COURSE_NAME>
      <COURSE_GRD>C</COURSE_GRD>
      <COURSE_HOURS>3</COURSE_HOURS>
      <TEACHER>Fisher</TEACHER>
   </CGA-MS>
 </CGA-MV>
</STUDENT>

<STUDENT>
 <_ID>987654321</_ID>
 <LNAME>Miller</LNAME>
 <CGA-MV>
    <SEMESTER>FA93</SEMESTER>
   <CGA-MS>
      <COURSE_NBR>EG110</COURSE_NBR>
      <COURSE_NAME>Engineering Principles</COURSE_NAME>
      <COURSE_GRD>C</COURSE_GRD>
      <COURSE_HOURS>5</COURSE_HOURS>
      <TEACHER>Carnes</TEACHER>
   </CGA-MS>
   <CGA-MS>
      <COURSE_NBR>MA220</COURSE_NBR>
      <COURSE_NAME>Calculus- I</COURSE_NAME>
      <COURSE_HOURS>5</COURSE_HOURS>
      <TEACHER>Otis</TEACHER>
   </CGA-MS>
   <CGA-MS>
      <COURSE_NBR>PY100</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>EG140</COURSE_NBR>
      <COURSE_NAME>Fluid Mechanics</COURSE_NAME>
      <COURSE_GRD>B</COURSE_GRD>
      <COURSE_HOURS>3</COURSE_HOURS>
      <TEACHER>Aaron</TEACHER>
   </CGA-MS>
   <CGA-MS>
      <COURSE_NBR>EG240</COURSE_NBR>
      <COURSE_NAME>Circut Theory</COURSE_NAME>
      <COURSE_GRD>B</COURSE_GRD>
      <COURSE_HOURS>3</COURSE_HOURS>
      <TEACHER>Carnes</TEACHER>
   </CGA-MS>
   <CGA-MS>
      <COURSE_NBR>MA221</COURSE_NBR>
      <COURSE_NAME>Calculus - II</COURSE_NAME>
      <COURSE_HOURS>5</COURSE_HOURS>
      <TEACHER>Otis</TEACHER>
   </CGA-MS>
  </CGA-MV>
 </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 UniQuery statement.

The mapping file

You can create the `U2XMLOUT.map` file in `$UDTHOME/sys/` to define commonly used global settings for creating XML documents. UniData reads and processes this mapping file each time UniData 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 UniQuery statement.

UniData processes XML options as follows:

1. Reads options defined in the `U2XMLOUT.map` file when UniData 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 UniQuery statement, as shown in the following example:

   ```
   LIST STUDENT SEMESTER TOXML XMLMAPPING mystudent.map
   ```
3. Processes any options you specify in the UniQuery statement.

Options you specify in the UniQuery 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_doc"
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** – UniData 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** – UniData uses the association element to change the display name of an association. By default, this name is the association phrase name, together with "-MV" or "-MS."

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 121 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. 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, UniData 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.

Hidems attribute

This attribute specifies whether to hide <MS> and </MS> tags for multi-subvalued 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 - ShowMS tags for multi-subvalued fields.

1 - Hide MS tags for multi-subvalued 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.

Collapsems attribute

This attribute specifies whether to collapse <MS> and </MS> tags, using only one set of these tags for multi-subvalued 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 MS tags for multi-subvalued fields.

1 - Collapse MS tags for multi-subvalued fields.

Namespace attributes

UniData provides the following attributes for defining namespaces:

- xmlns:name-space-name=”URL”
- targetnamespace=”URL”

UniData 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, UniData checks if the explicitly defined namespace has the same URL as the targetnamespace. If it does, UniData uses the namespace name to qualify the schema element, and the XML document element name.

If there is no other namespace explicitly defined, UniData creates a defaultnamespace in the schema file as shown in the following example:

```
xmlns=”targetnamespace URL”
```

In this case, UniData does not qualify the schema element or the XML document element.

UniData 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, UniData will not use a global namespace to qualify any element in the document.

The following program illustrates the output of the TARGETNAMESPACE attribute:

```
AE BP XML3
$INCLUDE INCLUDE XML.H
 CMD = "LIST STUDENT LNAME COURSE_NBR COURSE_GRD COURSE_NAME SEMESTER FNAME"


*   test with TARGETNAMESPACE

OPTIONS ="XMLMAPPING=student.map"
OPTIONS = OPTIONS:' ELEMENTS TARGETNAMESPACE=www.rocketsoftware.com'
PRINT OPTIONS
```

```
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 4 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 following example shows the output if the TARGETNAMESPACE attribute is set to “www.rocketsoftware.com”:

```
:LIST STUDENT LNAME COURSE_NBR COURSE_GRD COURSE_NAME SEMESTER FNAME TOXML WITHSCHEMA
 XMLMAPPING student.map
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="www.rocketsoftware.com"
  xmlns:rocketsoftware="http://www.rocketsoftware.com"
 xmlns="www.rocketsoftware.com"
 elementFormDefault="qualified">
 <xsd:annotation>
 <xsd:documentation xml:lang="en">
account: C:\U2\ud73\XMLDemo\udxml
    command: LIST STUDENT LNAME COURSE_NBR COURSE_GRD COURSE_NAME SEMESTER FNAME
TOXML WITHSCHEMA XMLMAPPING student.map
 </xsd:documentation>
 </xsd:annotation>
 <xsd:element name="MAIN">
    <xsd:complexType>
   <xsd:sequence>
       <xsd:element name="STUDENT" type="STUDENTType" minOccurs="0" maxOccurs="un
bounded"/>
   </xsd:sequence>
    </xsd:complexType>
 </xsd:element>
   <xsd:complexType name="STUDENTType">
      <xsd:sequence>
        <xsd:element name="_ID" type="xsd:string"/>
        <xsd:element name="LNAME" type="xsd:string"/>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="SEMESTER" type="xsd:string"/>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <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:sequence>
```

```
        <xsd:element name="FNAME" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
<?xml version="1.0"?>
<MAIN
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="www.rocketsoftware.com"
  xmlns:rocketsoftware="http://www.rocketsoftware.com"
>
<STUDENT>
 <_ID>123456789</_ID>
 <LNAME>Martin</LNAME>
    <SEMESTER>SP94</SEMESTER>
      <COURSE_GRD></COURSE_GRD>
      <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
      <COURSE_NBR>PY100</COURSE_NBR>
      <COURSE_GRD>C</COURSE_GRD>
      <COURSE_NAME>Golf - I</COURSE_NAME>
      <COURSE_NBR>PE100</COURSE_NBR>
 <FNAME>Sally</FNAME>
</STUDENT>

<STUDENT>
 <_ID>987654321</_ID>
 <LNAME>Miller</LNAME>
    <SEMESTER>FA93</SEMESTER>
      <COURSE_GRD>C</COURSE_GRD>
      <COURSE_NAME>Engineering Principles</COURSE_NAME>
      <COURSE_NBR>EG110</COURSE_NBR>
      <COURSE_GRD></COURSE_GRD>
      <COURSE_NAME>Calculus- I</COURSE_NAME>
      <COURSE_NBR>MA220</COURSE_NBR>
      <COURSE_GRD>B</COURSE_GRD>
      <COURSE_NAME>Introduction to Psychology</COURSE_NAME>
      <COURSE_NBR>PY100</COURSE_NBR>
    <SEMESTER>SP94</SEMESTER>
      <COURSE_GRD>B</COURSE_GRD>
      <COURSE_NAME>Fluid Mechanics</COURSE_NAME>
      <COURSE_NBR>EG140</COURSE_NBR>
      <COURSE_GRD>B</COURSE_GRD>
      <COURSE_NAME>Circut Theory</COURSE_NAME>
      <COURSE_NBR>EG240</COURSE_NBR>
      <COURSE_GRD></COURSE_GRD>
      <COURSE_NAME>Calculus - II</COURSE_NAME>
      <COURSE_NBR>MA221</COURSE_NBR>
 <FNAME>Susan</FNAME>
</STUDENT>

</MAIN>
:
```

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

UniData 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

UniData uses the File attribute to process both UniQuery and UniData 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 121.

```
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, UniData 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:

```
:AE _XML_ student.map
<!-- this is for STUDENT file -->
<U2
root="MAIN"
schema1 = "type"
xmlns1: rocketsoftware="http://www.rocketsoftware.com"
collapsemv='1'
collapsems='1'
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 root="MAIN"
targetnamespace="www.rocketsoftware.com"
hidemv ="1"
hidems ="1"
/>
```

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, UniData would create the XML data as if they were multivalued attributes.

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"?>
<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\ud73\XMLDemo\udxml
    command: LIST STUDENT LNAME SEMESTER COURSE_NBR TOXML XMLMAPPING student.map
SCHEMAONLY
 </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:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="SEMESTER" type="xsd:string"/>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="COURSE_NBR" type="xsd:string"/>
```

```
        </xsd:sequence>
        </xsd:sequence>
      </xsd:sequence>
      <xsd:attribute name="_ID" type="xsd:string"/>
      <xsd:attribute name="LNAME" type="xsd:string"/>
      </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:

```
LIST STUDENT LNAME SEMESTER COURSE_NBR TOXML XMLMAPPING student.map
<?xml version="1.0"?>
<MAIN>
<STUDENT _ID = "123456789" LNAME = "Martin">
    <SEMESTER>SP94</SEMESTER>
      <COURSE_NBR>PY100</COURSE_NBR>
      <COURSE_NBR>PE100</COURSE_NBR>
</STUDENT>

<STUDENT _ID = "987654321" LNAME = "Miller">
    <SEMESTER>FA93</SEMESTER>
      <COURSE_NBR>EG110</COURSE_NBR>
      <COURSE_NBR>MA220</COURSE_NBR>
      <COURSE_NBR>PY100</COURSE_NBR>
    <SEMESTER>SP94</SEMESTER>
      <COURSE_NBR>EG140</COURSE_NBR>
      <COURSE_NBR>EG240</COURSE_NBR>
      <COURSE_NBR>MA221</COURSE_NBR>
</STUDENT>

</MAIN>
:
```

Conversion code considerations

UniData uses the following rules when extracting data from database files:

- If the dictionary record of a field you are extracting contains a conversion code, UniData 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 UniQuery 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

UniData 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 UniQuery statement, that format will override the format defined in the mapping file.

Mapping file encoding

For special characters encountered in data, UniData uses the default XML entities to encode the data. For example, '<' becomes <, '>' becomes >, '&' becomes &, and ' " ' becomes ". However, UniData 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. UniData 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, UniData 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:

```
:LISTDICT STUDENT

DICT STUDENT  15:14:23 Aug 23 2011 1
@ID............ TYP LOC.......... CONV MNAME.......... FORMAT SM ASSOC.....

@ID             D               0      STUDENT         12R### S
                                                       -##-##
                                                       ##
ID              D               0      STUDENT         12R### S
                                                       -##-##
                                                       ##
LNAME           D               1      Last Name       15T    S
FNAME           D               2      First Name      10L    S
MAJOR           D               3      Major           4L     S
MINOR           D               4      Minor           4L     S
ADVISOR         D               5      Advisor         8L     S
SEMESTER        D               6      Term            4L     MV CGA
COURSE_NBR      D               7      Crs #           5L     MS CGA
COURSE_GRD      D               8      GD              3L     MS CGA
COURSE_HOURS    I   TRANS('COURSE      Hours           5R     MS CGA

                    S',COURSE_NBR
                    ,CREDITS,'X')

COURSE_NAME     I   TRANS('COURSE      Course Name     25L    MS CGA
                    S',COURSE_NBR
Enter <New line> to continue...
23 Aug 23 2007 2

@ID............ TYP LOC.......... CONV MNAME.......... FORMAT SM ASSOC.....

                    ,'NAME','X')
CGA             PH  SEMESTER COUR
                    SE_NBR COURSE
                    _NAME COURSE_
                    GRD COURSE_HO
                    URS TEACHER
@ORIGINAL       SQ            @ID
@SYNONYM        SQ             ID

GPA1            V   SUBR('GPA1',C MD3  GPA             5R     S
                    OURSE_HOURS,C
                    OURSE_GRD)

TEACHER         V   TRANS('COURSE      Teacher         10L    MS CGA
                    S',COURSE_NBR
                    ,'TEACHER','X
                    ')
17 records listed
```

Creating an XML document

To create an XML document using UniQuery, 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, UniData creates an xmlfile.xsd in the _XML_ directory. If you specify WITHDTD, UniData creates an xmlfile.dtd as well.

For detailed information about the `LIST` command, see *Using UniQuery*.

Examples

Creating an attribute-centric XML document

Using the mapping file described in Mapping file example, on page 121, the following example creates an attribute-centric XML document. To use a mapping file, specify the XMLMAPPING keyword in the UniQuery statement.

```
LIST STUDENT LNAME FNAME SEMESTER COURSE_NBR COURSE_GRD COURSE_NAME TOXML
XMLMAPPING student.map
<?xml version="1.0"?>
<MAIN>
<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>

<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>

</MAIN>
:
```

Creating an XML document with a DTD or XML schema

If you only include the TOXML keyword in the UniQuery 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"?>
<!DOCTYPE ROOT[
<!ELEMENT ROOT (STUDENT*)>
<!ELEMENT STUDENT ( CGA-MV* )>
<!ATTLIST STUDENT
        _ID CDATA #REQUIRED
>
<!ELEMENT CGA-MV ( CGA-MS* )>
<!ATTLIST CGA-MV
        SEMESTER CDATA #IMPLIED
>
<!ELEMENT CGA-MS  EMPTY>
<!ATTLIST CGA-MS
        COURSE_NBR CDATA #IMPLIED
        COURSE_GRD CDATA #IMPLIED
        COURSE_NAME CDATA #IMPLIED
]>
<MAIN>
<STUDENT _ID = "123456789">
 <CGA-MV SEMESTER = "SP94">
    <CGA-MS COURSE_NBR = "PY100"
COURSE_NAME = "Introduction to Psychology"/>
    <CGA-MS COURSE_NBR = "PE100" COURSE_GRD = "C" COURSE_NAME = "Golf - I"/>
 </CGA-MV>
</STUDENT>

<STUDENT _ID = "987654321">
 <CGA-MV SEMESTER = "FA93">
    <CGA-MS COURSE_NBR = "EG110" COURSE_GRD = "C"
COURSE_NAME = "Engineering Principles"/>
   <CGA-MS COURSE_NBR = "MA220" COURSE_NAME = "Calculus- I"/>
    <CGA-MS COURSE_NBR = "PY100" COURSE_GRD = "B"
COURSE_NAME = "Introduction to Psychology"/>
 </CGA-MV>
 <CGA-MV SEMESTER = "SP94">
    <CGA-MS COURSE_NBR = "EG140" COURSE_GRD = "B" COURSE_NAME = "Fluid Mechanics
"/>
    <CGA-MS COURSE_NBR = "EG240" COURSE_GRD = "B" COURSE_NAME = "Circut Theory"/
>
   <CGA-MS COURSE_NBR = "MA221" COURSE_NAME = "Calculus - II"/>
```

```
 </CGA-MV>
</STUDENT>

</MAIN>
```

Using WITHSCHEMA

Use the WITHSCHEMA keyword with the UniQuery `LIST` command to create an XML schema.

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]]...`

> **Note:** If you specify both WITHDTD and WITHSCHEMA in the same UniQuery statement, UniData does not produce an XML schema.

WITHSCHEMA creates an XML schema *filename*.xsd. By default, UniData 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, UniData generates the following:

`schemaLocation="namespaceURL filename.xsd"`

In both of these cases, you can validate the files using the XML schema validator, or the UniBasic 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 UniQuery 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"
maxOccu
s="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="MS"
treated-as="attribute"
/>
<u2
file="STUDENT"
field="COURSE_NAME"
map-to="courseName"
treated-as="attribute"
type="MS"
/>
<u2
file="STUDENT"
field="COURSE_GRD"
map-to="courseGrade"
type="MS"
/>
<u2
file="STUDENT"
field="COURSE_HOURS"
map-to="courseHours"
type="MS"
/>
<u2
file="STUDENT"
field="TEACHER"
map-to="courseInstructor"
type="MS"
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 using DB.TOXML

To create an XML document from ECL, use the `DB.TOXML` command.

Syntax

DB.TOXML *"xml_doc_filename" "xmap_filename" "condition"*

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
xml_doc_filename	The name of the XML document to create. If you do not enter a full path, the file is written to the _XML_ directory.
xmap_filename	The file name for the U2XMAP file.
condition	A UniQuery condition string, for example, WITH SCHOOL = “CO002”

Example

The following example illustrates using `DB.TOXML` from ECL to create an XML document.

```
DB.TOXML SCHOOL_STUDENT.XML STUDENT.MAP WITH SCHOOLID = "CO002"
```

For detailed information about using XML with UniData, see *Using UniData* and *UniBasic Extensions*.

Chapter 10: Receiving XML documents

XML documents are text documents, intended to be processed by an application, such as a web browser. UniData enables you to receive and create XML documents, and process them through UniBasic, UniData SQL, or UniQuery.

Transferring data drom XML to the database

You can store data contained in an XML document in UniData files. This process is called shredding. You can also create XML documents from data contained in UniData files.

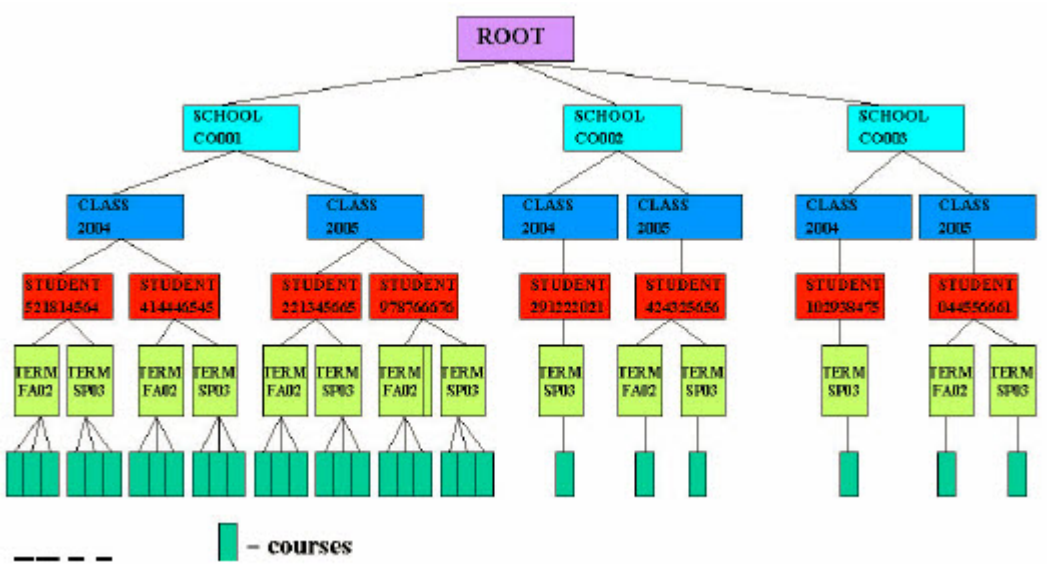
Prior to this release, transferring information from an XML document to UniData supported limited levels of nesting. At this release, you can transfer an unlimited levels of XML data to UniData files, although one UniData file can contain no more than three levels of data. If you are extracting more than three levels of data, you must write the data to more than one UniData file.

The following example shows a sample XML document containing information about students in different school districts:

```
<?xml version="1.0" ?>
<ROOT>
<SCHOOL SCHOOLID="CO001" NAME="Fairview"
     DISTRICT="BVSD">
 <CLASS CLASSOF="2004"
   <STUDENT ID="521814564" NAME="Harry Smith" DOB="1985-02-08">
    <TERM SEMESTER="FA02">
        <COURSE NAME="MA130" GRADE="A" />
        <COURSE NAME="CH100" GRADE="B" />
        <COURSE NAME="PY100" GRADE="B" />
    </TERM>
    <TERM SEMESTER="SP03">
        <COURSE NAME="MA131" GRADE="B" />
        <COURSE NAME="CH101" GRADE="B" />
        <COURSE NAME="PE220" GRADE="A" />
    </TERM>
   </STUDENT>
   <STUDENT ID="414446545" NAME="Karl Offenbach" DOB="1984-12-26">
                              ...
   </STUDENT>
  </CLASS>
  <CLASS CLASSOF="2005">
                              ...
 </SCHOOL>
<SCHOOL SCHOOLID="CO002" NAME="Golden" DISTRICT="ACSD"
 <CLASS CLASSOF="2004">
  <STUDENT ID="291222021" NAME="Jojo Smith" DOB="1985-08-06">
   <TERM SEMESTER="SP03">
    <COURSE NAME="FR101" GRADE="B" />
   </TERM>
  </STUDENT>
 </CLASS>
 <CLASS CLASSOF="2005">
  <STUDENT ID="424325656" NAME="Sally Martin" DOB="1985=12-01"
   <TERM SEMESTER="FA02"
    <COURSE NAME="PY100" GRADE="C" />
    <COURSE NAME="PE100" GRADE="C" />
   </TERM>
  </STUDENT>
```

```
 </CLASS>
<SCHOOL SCHOOLID="CO003" NAME="Cherry Creek" DISTRICT="CCSD"
                              ...
</SCHOOL>
</ROOT>
```

This document can be represented by the following XML document tree:



The following is an overview of the steps required to retrieve data from and XML database and store it in UniData files.

Generate database schema

First, you must understand the structure of the data in the incoming XML document. You can understand this structure by reviewing the DTD or Schema of the XML document.

Create UniData files

After you review the DTD or Schema from the incoming XML document, you must create the corresponding UniData data and dictionary files, and create a dictionary record for each element or attribute in the corresponding XML document.

If you want to transfer all the data from this sample XML document to UniData, you must deal with five levels of nested elements, SCHOOL, CLASS, STUDENT, TERM and COURSES. Therefore, you have to map this XML document to two UniData files. Logically, you would map the bottom three elements, STUDENT, TERM, and COURSES, to one UniData file. We will call this file the STUDENT file. We will map the top two elements, SCHOOL and CLASS, to another UniData file. We will call this file the SCHOOL file. The ID, NAME, and DOB attributes of the element STUDENT will map to singlevalued fields in the STUDENT file, the attribute SEMESTER of the subelement TERM will map to a multivalued field, and the attributes NAME and GRADE of the third-level nested element COURSES will map to multivalued fields as well. However, the fields that correspond to the NAME and GRADE attributes will be multi-subvalued.

Since the STUDENT file fields corresponding to XML attributes SEMESTER, NAME, and GRADE are all related, they will be combined into one association, called CGA.

The element CLASS serves as a link between the two UniData files, and therefore the field CLASS_OF appears in both UniData files.

The contents of the dictionary for each file follows:

```
DICT STUDENT    01:25:17pm  16 Sep 2003  Page    1

                Type &
Field..........Field. Field........ Conversion.. Column......... Output Depth &
Name...........Number Definition... Code........ Heading........ Format Assoc..

@ID             D    0                          STUDENT       10L    S
NAME            D    1                          Name          10L    S
DOB             D    2               D2         DOB           10L    S
CLASS_OF        D    3                          Class Of      10L    S
SEMESTER          D    4                                Semester        10L    M CGA
COURSE_NBR        D    5                                Course No       10L    M CGA
COURSE_GRD        D    6                                Grade           10L    M CGA

7 records listed.
>

DICT SCHOOL    01:27:14pm  16 Sep 2003  Page    1

                Type &
Field..........Field. Field........ Conversion.. Column......... Output Depth &
Name...........Number Definition... Code........ Heading........ Format Assoc..

@ID             D    0                          SCHOOL          10L    S
SCHOOLID        D    0                          SchoolId        10L    S
SCHOOL_NAME     D    1                          Name            10L    S
SCHOOL_DISTRIC D    2                       District        10L    S
T
CLASS_OF        D    3                       Class Of        10L    S

5 records listed.
```

Create the U2XMAP file

The rules for transferring data between an XML document and database files are recorded in a separate file, referred to as the U2XMAP file. This file contains such information as the starting node of the XML document, names and relationships of database files that are being used to exchange data with a specified XML document, the mapping of XML attribute names to database field names, and other optional information, such as the mapping of NULL values and date format conversions.

The following example illustrates a U2XMAP:

```
<?xml version="1.0" ?>
<!--DOCTYPE U2XMAP SYSTEM "U2XMAP.DTD"    -->
<U2XMAP Version="1.0" Name="XMAP1">
   <!-- Table/Class Map    -->
   <TABLECLASSMAP MapName="M1" StartNode="/ROOT/SCHOOL" TableName="SCHOOL">
      <ColumnMap Node="@SCHOOLID" Column="SCHOOLID" />
      <ColumnMap Node="@NAME" Column="SCHOOL_NAME" />
      <ColumnMap Node="@DISTRICT" Column="SCHOOL_DISTRICT" />
      <ColumnMap Node="CLASS, @CLASSOF" Column="CLASS_OF" />
      <TableMap Node="CLASS/STUDENT" MapName="M2" />
   </TABLECLASSMAP>
   <TABLECLASSMAP MapName="M2" StartNode="CLASS/STUDENT" TableName="STUDENT">
      <ColumnMap Node="@ID" Column="@ID" />
      <ColumnMap Node="@NAME" Column="NAME" />
      <ColumnMap Node="@DOB" Column="DOB" />
      <ColumnMap Node="TERM, @SEMESTER" Column="SEMESTER" />
      <ColumnMap Node="TERM, COURSES, @NAME" Column="COURSE_NBR" />
      <ColumnMap Node="TERM, COURSES, @GRADE" Column="COURSE_GRD" />
   </TABLECLASSMAP>
```

Each TABLECLASSMAP element defines where to find the data in the XML document, and where to place it in the UniData data file based on the dictionary definition of the field.

Syntax

```
TABLECLASSMAP MapName = "xx" StartNode = "startnode" TableName =
"UniData file name"
```

Parameters

The following table describes each parameter of the syntax:

Parameter	Description
MapName	The name of the relationship between the portion of the XML document that starts with StartNode and the UniData data file.
StartNode	The XPath expression defining the starting position in the XML document.
TableName	The name of the target UniData file.

To map a particular XML attribute to a UniData field, use the ColumnMap element.

Syntax

```
ColumnMap Node=XPath expression, Column = UniData record field
```

The ColumnMap node defines the location of the node in the XML document. The ColumnMap Column defines the field in the UniData file to which you want to map the XML data. The UniData file must exist, and the dictionary record for the field must be defined.

Mapping XML data to multivalued fields

If you want to map an XML attribute to a multivalued field in the UniData record, specify a comma (“,”) before the name of the XML attribute, as shown in the following example:

```
ColumnMap Node = “CLASS, @CLASSOF” Column = “CLASS_OF”
```

If you want the values of the corresponding UniData data files to be multi-subvalued, such as COURSE_NBR and COURSE_GRADE in the STUDENT file, specify a comma before the attribute of the next level subelement and another comma before the attribute of the next level subelement, as shown in the following example:

```
ColumnMap Node="TERM, COURSES, @NAME"
Column="COURSE_NBR"
ColumnMap Node="TERM, COURSES, @GRADE"
Column="COURSE_GRD"
```

Defining a map relationship

If you are mapping the XML attributes to more than one UniData data file, you must define a dependent map using the TableMap Node element.

```
TableMap Node=”CLASS/STUDENT” MapName=”M2”
```

In this example, MapName M2 is defined within the MapName M1 element as a dependent map to M1.

```
<TABLECLASSMAP MapName=”M1” StartNode=”/ROOT/SCHOOL” TableName=”SCHOOL”>
<ColumnMap Node=”@SCHOOLID” Column=”SCOOLID” />
<ColumnMap Node=”@NAME” Column=”SCHOOL_NAME” />
<ColumnMap Node=”@DISTRICT” Column=”SCHOOL_DISTRICT” />
<ColumnMap Node=”CLASS, @CLASSOF” Column=”CLASS_OF” />
<TableMap Node = “CLASS/STUDENT” MapName=”M2” />
</TABLECLASSMAP>
```

Defining related tables

If you are mapping more than three levels of data, you must map the data to more than one UniData file, since a UniData file can support no more than three levels of data. In the U2XMAP file, you define the files that are related to each other using the RelatedTable element.

Use the MapParentKey element to define the parent file (the file corresponding to the top portion of the XML subtree being transformed). Use the MapChildKey to define each child file of the parent file, as shown in the following example:

```
<RelatedTable>
      <MapParentKey TableName=”SCHOOL” Column=”CLASS_OF” Key Generate=”No” />
    <MapChildKey TableName=”STUDENT” Column=”CLASS_OF” />
</RelatedTable>
```

In this example, SCHOOL is the parent UniData file which contains one child file, STUDENT. You must define a field that appears in both UniData files using the Column element. In this case, CLASS_OF appears in both the SCHOOL and STUDENT files.

The KeyGenerate element determines if UniData generates the parent/child key or not.

Populating the database

After you define the U2XMAP file, you can populate the UniData database from ECL or UniBasic.

Populating the database from ECL

Use the XML.TODB command to populate the UniData database from ECL.

Syntax

XML.TODB *<XML Document> <U2XMAP File>*

The following example assumes that the XML document STUDENT.XML and the U2XMAP STUDENT.MAP are located in the _XML_ file.

```
XML.TODB  STUDENT.XML  STUDENT.MAP
LIST SCHOOL
SCHOOL.....   Name.....    District.....    Class Of...

CO001         Fairview     BVSD             2004
                                            2005
CO002         Golden       ACSD             2004
                                            2005
CO003         Cherry Creek CCSD             2004
                                            2005

LIST STUDENT
STUDENT.....  Name.............  DOB...    Class Of   Semester..  Course NO.  Grade

414446545     Karl Offenbach     24 DEC 84  2004      FA02        HY104       D
                                                                  MA101       C
                                                                  FR100       C
                                                      SP03        HY105       B
                                                                  MA102       C
                                                                  FR101       C
4243255656    Sally Martin       01 DEC 85  2005      FA02        PY100       C
                                 .........
```