Rocket UniData

Using UniData

Version 8.1.1

December 2015
UDT-811–UDTU-1

Notices

Edition

Publication date: December 2015
Book number: UDT-811–UDTU-1
Product version: Version 8.1.1

Copyright

Trademarks

Examples

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

License agreement

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

Corporate information

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

Website: www.rocketsoftware.com

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

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

Country	Toll-free telephone number
United States	1-855-577-4323
Australia	1-800-823-405
Belgium	0800-266-65
Canada	1-855-577-4323
China	800-720-1170
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 Customer Portal is the primary method of obtaining support. If you have current support and maintenance agreements with Rocket Software, you can access the Rocket Customer Portal and report a problem, download an update, or read answers to FAQs. To log in to the Rocket Customer Portal or to request a Rocket Customer Portal account, go to www.rocketsoftware.com/support.

In addition to using the Rocket Customer Portal 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: UniData relational database concepts

This book is a guide to the Rocket UniData Relational Database Management System (RDBMS). It provides information about UniData for end users, system administrators, and database administrators. This book is not intended to be a guide to designing a database; rather, it gives you the tools to create dictionary and data tables that fit your database design.

The UniData relational database management system

UniData is a relational database management system (RDBMS) designed specifically for business users. A database management system is software that enables users to control the organization, storage, retrieval, security, and integrity of data in a database.

The UniData technology is based on the nested relational database model. This nested design extends the standard relational model from a redundant, flat table structure to a three-dimensional database.

UniData uses several access methods to store, manipulate, and report data. Briefly, these are:

- **UniBasic:** UniBasic is a powerful programming language you can use to write sophisticated programs. From within UniBasic, you can run system-level processes or call C programs. You can also call UniBasic programs from C programs.
- **UniData SQL:** The UniData implementation of the SQL (Structured Query Language) data manipulation language.
- **UniQuery:** UniQuery is the UniData nonprocedural query language for creating reports and viewing your database.
- **AE:** AE is UniData's Alternate Editor. AE is used to edit records in the database and create UniBasic programs.

Note: For additional information on these features, see *Developing UniBasic Applications*, *Using UniData SQL*, and *Using UniQuery*.

In the nested relational model that UniData employs, data is stored in files, and the records in a file can have attributes that contain values that can be either singlevalued, multivalued, or multi-subvalued. This allows UniData to store relations within relations. This method of storing data accomplishes the following:

- Simplifies the data modeling process
- Reduces the number of files and indexes in a database
- Eliminates redundant data storage
- Maintains relational access to all data in the database

Elements of a relational database

Relational databases consist of several elements. The terms for these elements are:

- Database
- File

- Record
- Primary key
- Attribute

Database

A database is a collection of interrelated data. In most cases, a relational database contains a group of files. Each file contains a group of records, while each record contains attributes, and each attribute contains values.

File

A file is a collection of records that are somehow related. While files contain records, records contain one or more attributes that contain specific information about one element of the record. For example, the CLIENTS file contains records of a company's clients. Each record contains elements of information about the client, such as name, address, and phone number, each of which are stored in separate attributes.

Record

A record is a group of related attributes that contain data about a single entity. Data for a single entity, such as a client, can be stored in a database in the same way that you would store all pertinent information in a single file folder in a file cabinet. If you have four clients, you need four individual records.

Primary key

A primary key is a unique identifier for a record; it is also called a record ID, or @ID. Each record must have a unique identifier that differentiates it from other records in the database. UniData uses the primary key to locate records in files. Common record identifiers that can serve as primary keys are account numbers, social security numbers, and medical record numbers. However you decide to identify your primary keys, each one must be unique.

Variable-length attributes and records

Unlike other databases that require you to specify an attribute length, UniData does not limit the length of your data within an attribute, nor does it pad the attribute or record to achieve a fixed length. UniData does not have to rely on fixed lengths to recognize where an attribute begins and ends; instead, it inserts delimiters between attributes and between records. This is called a dynamic array structure.

The following table describes the UniData delimiters. In a record, a delimiter is displayed as an ASCII value or as the character represented by the ASCII value.

Delimiter symbol	ASCII value	Name	Description
(nonprinting)	255	Record mark (@RM)*	Marks the end of a record.
~	254	Attribute mark (@AM)*	Marks the end of an attribute.
}	253	Value mark (@VM)*	Marks the end of multivalue.

Delimiter symbol	ASCII value	Name	Description
|	252	Subvalue mark (@SM)*	Marks the end of multi-subvalue.
{	251	Text mark (@TM)*	Marks the point where values wrap to a new line.

*( ) indicates system variables used by UniQuery and UniBasic to represent these marks.

Attribute

An attribute is an element of information in a record. Attributes are sometimes referred to as fields. Taken together, all attributes for one entity constitute a record.

For example, in the CLIENTS demo database file, each client record consists of elements, such as name, street address, city, country, telephone number, and fax number. Each of these elements is an attribute of the record.

Singlevalued attributes

A singlevalued attribute can contain only one value. For instance, the NAME attribute in a CLIENT file is singlevalued because a client can have only one name.

The end of an attribute value is delimited by an attribute mark.

Multivalued attributes

UniData employs a record concept that supports multivalued attributes. A multivalued attribute is one that usually contains more than one value. For example, a CLIENT file could contain an attribute listing the contacts at the client site. This is a multivalued attribute since there may be more than one contact at the client site.

The end of a multivalued attribute is delimited by a value mark.

Multi-subvalued attributes

UniData allows multivalued attributes to contain nested values, called subvalues. A multi-subvalued attribute is normally related to another attribute. This relationship is called an association. For example, the CLIENT file could contain an attribute listing the names of the contacts' children. This attribute is multi-subvalued, since each contact could have more than one child; and this multi-subvalued attribute might be associated with the attribute containing the contacts at the client site.

The end of a multi-subvalued attribute is delimited by a subvalue mark.

UniData dictionaries

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-type records, and items that calculate or manipulate data, called virtual fields, or V-type records. A user may also define a dictionary item to store user-defined data, called X-type records.

Note: UniData reserves attributes 30 - 39 of a dictionary record for the use of customers.

Dictionary file

A dictionary file, just like a data file, is a collection of records containing attributes. DICT.DICT is the master dictionary for all dictionary files. 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 attributes 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 D-type dictionary record.

The following table describes the contents of all D-type dictionary records.

Attribute number	Contents	Optional or required	Description
0	@ID	Required	The primary key or name of the attribute.
1	D [desc]	Required	For a D-type dictionary record, this attribute must contain D. You may enter a description of the attribute for your own documentation, if desired.
2	Location	Required	The attribute ordinal position in the data record, such as 1, 2, or 3. A value of 1 indicates the first position in the data record, a value of 2 indicates the second position in the data record, and so on. A value of 0 in this attribute always specifies the @ID. You can create more than one dictionary record for each attribute in the data file. This creates synonyms, allowing different users to access data using their own preferences for headings, format, etc.
3	Conversion Code	Optional	Any valid conversion code supported by the UniBasic OCONV function. Any data attribute that is stored in internal format, such as a date, may be converted to display format. Valid conversion codes are listed in Dictionary conversion codes and format options, on page 163
4	Display Name	Optional	If a value is present in this field, UniData displays this value as the column heading in reports. If this attribute is blank, UniData uses the attribute name as the column heading. If you want to create a display name containing more than one line, you can insert a value mark (^253) in the display name. Contents: Displays As: HOLIDAY CARD HOLIDAY CARD HOLIDAY^253CARD HOLIDAY CARD

Attribute number	Contents	Optional or required	Description
5	Format	Required	This attribute specifies how you want the data attribute to be displayed. The contents of this attribute may include the length and justification of the output, along with any special characters to be included in the display. The format can be any valid parameter supported by the UniBasic FMT function. For a list of valid parameters, see Dictionary conversion codes and format options, on page 163.
6	Value Code Specifier	Required	Identifies whether the data attribute is singlevalued (S), multivalued (MV), or multi-subvalued (MS). If this attribute contains M, UniData assumes a multi-subvalued attribute. Although M is a valid entry for a multi-subvalued field in UniData, it is not by UniData SQL.
7	Association	Optional	The name of a PH-type dictionary record that contains the association of several related dictionary records. Only multivalued and multi-subvalued fields may be associated. For example, you may have one multivalued attribute containing children's names, with another multivalued attributed containing those children's ages. If these two values are associated, each value of the name relates to the same value of the age. All multivalued and multi-subvalued attributes are not necessarily associated with any other attribute. Take care in establishing associations, especially when using UniData SQL.

The following example shows a D-type dictionary record:

```
:AE DICT INVENTORY INV_DATE
Top of "INV_DATE" in "DICT INVENTORY", 7 lines, 29 characters.
*--: P
001: D
002: 1
003: D4/
004: Inventory}Date
005: 10R
006: S
007:
Bottom.
```

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.

Tools available for use in virtual attributes include:

- **Computational Operators:** Anything needed for mathematical operations, such as plus, minus, and so forth.
- **String Extraction Operators:** Operators to extract all or part of a string of data.
- **Relational Operators:** Operators needed for comparative logic, such as greater than, less than, equal to, etc.
- **Boolean Operators and Conditional Expression:** AND, OR, and IF/THEN/ELSE constructions.
- **UniBasic String, Manipulation, and Conversion Functions:** Functions that allow you to manipulate numeric data (ABS, RND, SQRT, etc.) as well as manipulate text (DOWNCASE, SPACE, STR, etc.). For more information about these functions, see *Developing UniBasic Applications.*
- **System Variables:** Called "at" (@) variables.
- **Functions:** TRANS and TOTAL functions.
- **Subroutines:** A call for a UniBasic subroutine.

In a virtual attribute, you can apply the above tools to attribute names, numeric constants, and literal strings. You can make several function calls in one virtual attribute by separating each call with a semicolon (;).

Creating virtual attributes is discussed in depth in Creating virtual attributes, on page 71. For more information about UniBasic functions, see *Developing UniBasic Applications*.

The following table describes the attributes of V-type dictionary records.

Attribute number	Contents	Optional or required	Description
0	@ID	Required	The primary key, or name, of the attribute.
1	V or I [desc]	Required	For a V-type record, this attribute must be V or I. You may enter a description of the attribute for your own documentation, if desired.
2	Location	Required	The virtual field formula to be evaluated.
3	Conversion Code	Optional	Any valid conversion code supported by the UniBasic OCONV function. Any data field that is stored in internal format, such as a date, may be converted to display format. Valid conversion codes are listed in Dictionary conversion codes and format options, on page 163.
4	Display Name	Optional	If a value is present in this field, UniData displays the value as the column heading in a report. If this attribute is blank, UniData uses the attribute name as the column heading. If you want to create a display name containing more than one line, you can insert a value mark (^253) in the display name. Contents: Displays As: HOLIDAY CARD HOLIDAY CARD HOLIDAY^253CARD HOLIDAY CARD
5	Format	Required	This attribute specifies how you want the data attribute to be displayed. The contents of this attribute may include the length and justification of the output, along with any special characters to be included in the display. The format can be any valid parameter supported by the UniBasic FMT function. For a list of valid parameters, see Dictionary conversion codes and format options, on page 163.

Attribute number	Contents	Optional or required	Description
6	Value Code Specifier	Required	Identifies whether the data attribute is singlevalued (S), multivalued (MV), or multi-subvalued (MS). If this attribute contains M, UniData assumes a multi-subvalued attribute. Although M is a valid entry for a multi-subvalued attribute in UniData, it is not supported by UniData SQL.
7	Association	Optional	The name of a PH-type dictionary record that contains the association of several related dictionary records. Only multivalued and multi-subvalued fields may be associated. For example, you may have one multivalued attribute containing children's names, with another multivalued attributed containing those children's ages. If these two values are associated, each value of the name relates to the same value of the age. All multivalued and multi-subvalued attributes are not necessarily associated with any other attribute. Take care in establishing associations, especially when using UniData SQL.
8-9	Derived	Optional	Attributes 8-9 are reserved for internal use and should not be used.

The following example shows a V-type dictionary record:

```
:AE DICT INVENTORY DIFF
Top of "DIFF" in "DICT INVENTORY", 7 lines, 44 characters.
*--: P
001: V
002: QTY - REORDER
003: MD2,
004: Difference
005: 6R
006: MV
007: LINE_ITEMS
Bottom.
```

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 to store @UQ and @LPTR information. @UQ displays a default set of dictionary attributes when you issue a LIST or SORT command. @LPTR displays a default set of dictionary attributes when you issue the LPTR command with no other dictionary items specified. Phrases can also be used to store an alias for a fragment of a UniQuery statement that you frequently use.

The following table describes the attributes of a PH-type dictionary record.

Attribute number	Contents	Optional or required	Description
0	@ID	Required	The primary key, or name, of the phrase. If used to associate related attributes, the @ID of the phrase must be the association name defined in attribute 7 of the associated attributes' dictionary records. If used to display default dictionary attributes with the LIST or SORT commands, the @ID must be @UQ. If used to print default dictionary attributes with the LPTR command, the @ID must be @LPTR.
1	PH [desc]	Required	For a PH-type record, this attribute must be PH. You may enter a description of the attribute for your own documentation, if desired.
2	Phrase	Required	The contents of the phrase may be any part of a valid UniQuery statement that does not contain a verb. If used to associate related attributes, attribute 2 of the PH-type record contains the dictionary names of the attributes that are related.

The following example shows a PH-type dictionary record for an association:

```
:AE DICT INVENTORY LINE_ITEMS
Top of "LINE_ITEMS" in "DICT INVENTORY", 2 lines, 31 characters.
*--: P
001: PH
002: COLOR PRICE QTY REORDER DIFF
Bottom.
```

X-type dictionary records

An X-type dictionary attribute stores user-defined information, and is ignored by UniData. 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.

The following table describes the attributes of an X-type dictionary record.

Attribute number	Contents	Optional or required	Description
0	@ID	Required	The primary key or name of the X-type record.
1	X [desc]	Required	For an X-type record, this field must be X. You may enter a description of the attribute for your own documentation, if desired.
2-*n*	User-defined data	Optional	These attributes contain user-defined data.

The following example shows an X-type dictionary record that stores the next available INVENTORY ID:

```
:AE DICT INVENTORY NEXT_INV_NO
Top of "NEXT_INV_NO" in "DICT INVENTORY", 7 lines, 28 characters.
```

```
*--: P
001: X
002: 58051
Bottom.
```

SQ-type 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.

The DICT.DICT dictionary

DICT.DICT is the dictionary that defines the attributes in all other UniData dictionaries, including those for the VOC (vocabulary) file. (DICT.DICT is the final default for locating dictionary names related to the data file named in a query. If UniData does not find an attribute name in the file dictionary, it searches the VOC file. If it does not find the item there, it searches DICT.DICT. The DICT.DICT file resides in `udthome/sys` on UniData for UNIX, or `udthome\sys` on UniData for Windows platforms.)

UniData includes a global DICT.DICT, shown in the following example:

```
SORT DICT DICT.DICT TYP LOC CONV MNAME FORMAT SM ASSOC BY TYP BY
LOC BY @ID 16:47:32 Jun 07 2011 1
@ID............ TYP LOC.......... CONV MNAME.......... FORMAT SM
ASSOC.....

@ID             D              0                      15L    S
TYPE            D              1                      3L     S
LOC             D              2                      13R    S
CONV            D              3                      4L     S
NAME            D              4                      15L    S
FORMAT          D              5                      6L     S
SM              D              6                      2L     S
ASSOC           D              7                      10T    S
MNAME           I          NAME                       15L    M
@UQ             PH  BY TYP BY @ID
                     TYP LOC CONV
                     NAME FORMAT
                    SM ASSOC
TYP             V   TRIM(TYPE[1,2                     3L     S
                    ])
11 records listed
```

You can create a local DICT.DICT that overrides the global DICT.DICT. When you start a UniData session, UniData searches the VOC file for a local DICT.DICT before it reads the global DICT.DICT in the sys directory. When you start a UniData session, DICT.DICT is loaded into memory.

Modifying DICT.DICT

To modify the global DICT.DICT, you must create a VOC pointer, as shown in the following example:

```
:AE VOC DICT.DICT
Top of "DICT.DICT" in "VOC", 3 lines, 23 characters.
001: F
002: @UDTHOME/sys/DICT.DICT
003: @UDTHOME/sys/DICT.DICT
```

```
Bottom.
```

If a local DICT.DICT file exists, but is empty, UniData displays the error message “The current DICT.DICT is empty” when you attempt to execute a UniQuery command against the dictionary portion of any file, such as LIST DICT VOC.

You can add or modify records in the global DICT.DICT file, or in the local DICT.DICT file to suit your needs. If you modify DICT.DICT, users who are already in a UniData session must issue the `READDICT.DICT` command to override the version of DICT.DICT that was loaded into memory when they began their UniData sessions.

Data type enforcement

Beginning at UniData 7.3, UniData provides a data type check on data you write to a file in an effort to avoid writing bad data to the database. This data type check is file-based.

You can specify a field in a file on which you want to run a data type check. In each record write, UniData then checks that field against the data type defined by the metadata. If a write fails, UniData reports the record ID and the data that failed the check.

For information about the U2Metadata Manager tool (U2 MDM), see the online help available from the **Help** > **Help Contents** option in the U2 Metadata Manager tool.

You can define the following data types to check:

- INTEGER (SMALL INT)
- NUMERIC (DECIMAL, FLOAT, REAL)
- DATE (with a date format such as D2, D, or D6)

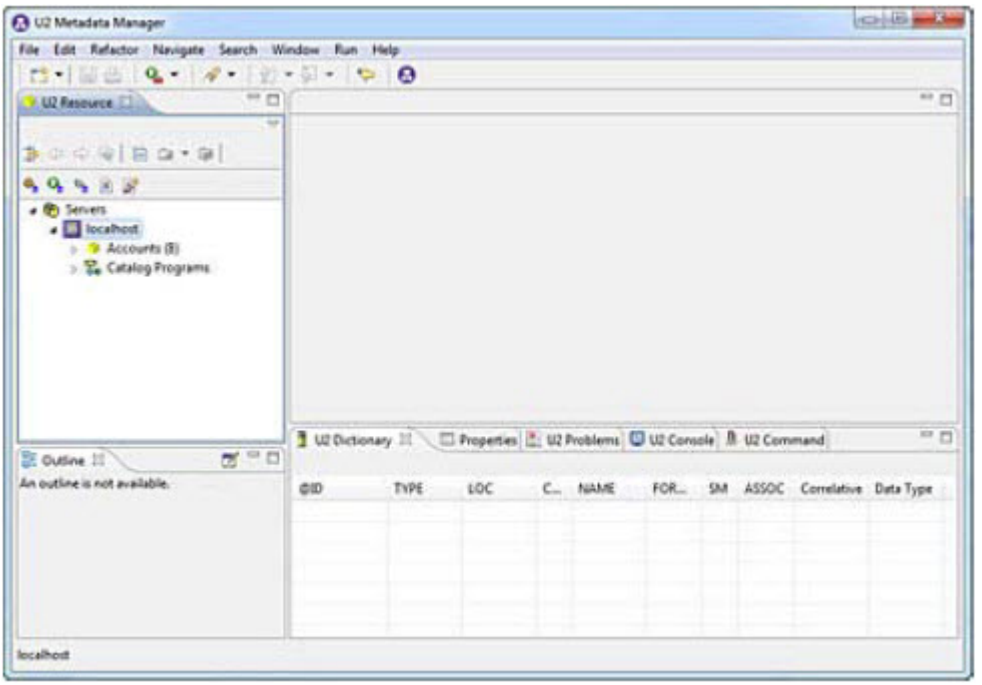
The data type is determined by the conversion code in the dictionary record of the attribute.

Conversion code	Data type
D*n*	DATE
MT	TIME
MD0	INTEGER (SMALL INT)
MD2, MR2	NUMERIC (DECIMAL, FLOAT, REAL)

Selecting an attribute for data type enforcement

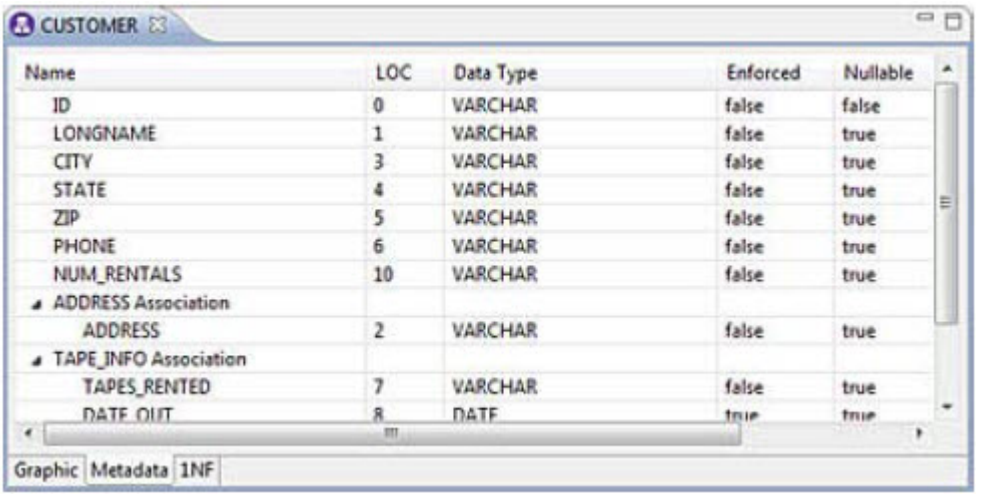
To select an attribute to apply data type enforcement, use the U2 Metadata Manager (U2 MDM) tool. For information about installing the Metadata Manager tool and connecting to the server, see the online help in the U2 Metadata Manager (U2MDM) tool.

From the **Start** menu, click **All Programs** > **Rocket U2** > **U2 Metadata Manager**. A window similar to the following example appears:



From the U2 Resource View, expand **Accounts**, then expand the account where the file for which you want to define attributes for data type enforcement resides. Drag the file name and place it in the Metadata pane, or right-click the file name, then select **U2 Metadata Manager** > **Open U2 MDM Editor**.

Click the **Metadata** tab. A CUSTOMER editor similar to the following example appears.



Select the attribute for which you want to enforce the data type. In the **Enforced** column, select **true**.

Creating and deleting metadata list

Use the commands discussed in this section to create, delete, and view metadata.

CREATE.METADATA

The CREATE.METADATA command creates a metadata list for a UniData file from the Metadata Repository File (_METADATA_REPOSITORY_) that contains all of the D-type attributes for which you want to enforce the data type. You can create only one definition for each location. U2 MDM physically saves the metadata list information to the file property group (FPG) of the file.

Syntax

```
CREATE.METADATA filename USING REPOSITORY
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	The UniData hashed file for which you want to define metadata to check.
USING REPOSITORY	The name of the core mapping file generated from the Metadata Manager. For information about the Metadata Manager, see the U2 Metadata Manager online help.

Example

The following example illustrates the CREATE.METADATA command:

```
:CREATE.METADATA INVENTORY USING REPOSITORY
Metadata is created.
```

DELETE.METADATA

Use the DELETE.METADATA command to delete the metadata list previously created for the file.

Syntax

```
DELETE.METADATA filename
```

where *filename* is the name of the file for which you created metadata.

LIST.METADATA

The LIST.METADATA command lists all the D-type attributes and associations previously defined for filename.

Syntax

```
LIST.METADATA filename
```

where *filename* is the name of the file for which you created metadata.

Example

The following example illustrates the output from the LIST.METADATA command:

```
:LIST.METADATA INVENTORY
  Data for Column @ID
       Position:       0
       Data Type:      VARCHAR
       Type Enf:       0
       dict info:       'INVENTORY' 10R S
  Data for Column INV_DATE
       Position:       1
       Data Type:      DATE
       Type Enf:       1
       dict info:      D4/ 'Inventory}Date' 10R S
  Data for Column INV_TIME
       Position:       2
       Data Type:      TIME
       Type Enf:       1
       dict info:      MTH 'Inventory}Time' 5R S
  Data for Column PROD_NAME
       Position:       3
       Data Type:      VARCHAR
       Type Enf:       0
       dict info:       'Product}Name' 10T S
  Data for Column FEATURES
       Position:       4
       Data Type:      VARCHAR
       Type Enf:       0
       dict info:       'Features' 30T S
  Data for Column COLOR
       Position:       5
       Data Type:      VARCHAR
       Type Enf:       0
       dict info:       'Color' 10T MV LINE_ITEMS
  Data for Column QTY
       Position:       6
       Data Type:      INT
       Type Enf:       0
       dict info:      MD0 'Quantity' 6R MV LINE_ITEMS
  Data for Column PRICE
       Position:       7
       Data Type:      FLOAT
       Type Enf:       1
       dict info:      MD2,$ 'Price' 10R MV LINE_ITEMS
  Data for Column REORDER
       Position:       8
       Data Type:      VARCHAR
       Type Enf:       0
       dict info:       'Reorder}Point' 6R MV LINE_ITEMS

  Data for Association_0



       Name:         SingleValuedFields
       SM Type:      S
       Field List:   @ID INV_DATE INV_TIME PROD_NAME FEATURES

  Data for Association_1
       Name:         LINE_ITEMS
       SM Type:      MV
       Field List:   COLOR QTY PRICE REORDER
:
```

Data type enforcement (DTENF) commands

This section describes the ECL commands UniData provides for data type enforcement. These commands maintain the data type enforcement (DTE) list in the file property group of the UniData hashed file.

ENABLE.DTENF command

The `ENABLE.DTENF` command enables one or more items in a metadata list. You can generate or add items to the DTENF list. For WRITE operations, data type enforcement goes through the LOCLIST and returns and error when the first location fails the data type check.

Syntax

```
ENABLE.DTENF filename [ALL | FIELDLIST name_list | LOCLIST loc_list]
[DTE.OPT {IGNORE | LOG | ON.ERROR}]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
filename	The name of the file for which you want to enable data type enforcement.
ALL	Enable all fields in the _METADATA_REPOSITORY_ record.
FIELDLIST *name_list*	Enables the fields defined by name. *name_list*. is defined as *name1 name2 name3* ...
LOCLIST *loc_list*	Enables the fields by location. loc_list is defined as *location1 location2 location3* ...
DTE.OPT	The error handling option to use when a data type enforcement check fails. Valid options are: ▪ IGNORE - UniData takes no action If the dte log exists, and error is written to the DTE_acct_pid record. ▪ LOG - Write the error to the DTE_acct_*filename* record in the dte log. ▪ ON.ERROR - Returns a WRITE error and sets an error status. The INMAT function returns the following values after a UniBasic WRITE is executed from a UniBasic program or subroutine: 0 – The WRITE completed successfully. 2 – The WRITE completed, but data type enforcement checking failed. Check the log file for errors.

Example

The following example illustrates the `ENABLE.DTENF` command:

```
:ENABLE.DTENF CUSTOMER ALL DTE.OPT LOG
DTE has been changed on file CUSTOMER,1 DTE item(s) enabled
```

DISABLE.DTENF command

Use the `DISABLE.DTENF` command to delete items from the DTENF list.

Syntax

```
DISABLE.DTENF filename [ALL | FIELDLIST name_list | LOCLIST loc_list]
```

Parameters

The following table describes each parameter of the syntax:

Parameter	Description
filename	The name of the file for which you want to disable data type enforcement.
ALL	Disable all defined fields in the DTE list.
FIELDLIST *name_list*	Disables the fields defined by name. *name_list* is defined as name1, name2, name3 ...
LOCLIST *loc_list*	Disables the fields by location. *loc_list* is defined as *location1*, *location2*, *location3...*

LIST.DTENF command

The `LIST.DTENF` command lists the items in the current DTE list.

Syntax

```
LIST.DTENF filename
```

where *filename* is the name of the file for which you want to list the DTE items.

Example

The following example illustrates the output from the `LIST.DTENF` command:

```
:LIST.DTENF INVENTORY
DTE enabled on INVENTORY.
   Option: 1
   Location: 1 Datatype: DATE
   Location: 2 Datatype: TIME
   Location: 7 Datatype: FLOATParameter
```

VERIFY.DTENF command

The `VERIFY.DTENF` command checks the locations defined in the DTE list and generates a list of @IDs that contain invalid data types.

Syntax

```
VERIFY.DTENF filename
```

where *filename* is the name of the file for which you want to verify data types.

Example

The following example illustrates the `VERIFY.DTENF` command. One record is selected because Robert is the value of the INV_DATE attribute, which is defined as a DATE data type.

```
VERIFY.DTENF INVENTORY
1 records selected to list 0.
>
```

```
LIST INVENTORY INV_DATE INV_TIME PROD_NAME FEATURES COLOR PRICE QTY REORDER DI
09:26:59 Feb 24 2012 1
INVENTORY 56060
Inventory Date Robert
Inventory Time 12:00PM
Product Name Trackball
Features Super Deluxe Model
Color Price Quantity Reorder Difference
Gray $98.99 494 70 424
1 record listed
```

SET.DTELOG command

The `SET.DTELOG` command enables or disables the DTE log. The name of the log file is DTE_acct_pid, and is located in the $UDTTMP directory.

Syntax

```
SET.DTELOG [ON | OFF]
```

Parameters

The following table describes each parameter of the syntax.

Parameters	Description
ON	Enables the DTE log file.
OFF	Disables the DTE log file.

The INMAT function

The `INMAT` function returns the following values after a UniBasic WRITE is executed from a UniBasic program or subroutine:

- 0 – The WRITE completed successfully
- 2 – The WRITE completed, but data type enforcement checking failed. If you specify the LOG DTE.OPT, the errors appear in the file_level log. If you specify the IGNORE option, use the `SET.DTELOG` command to set up your log file

Deploying data type enforcement

If you use a test account to set up data type enforcement, you can copy the changes to the live account in certain conditions. This works if the dictionary files in the live account are identical to those in the test account. To copy changes to the live account, use a paragraph similar the following example.

```
:AE VOC DEPLOYDTE
Top of New "DEPLOYDTE" in "VOC".
*--: I
001= PA
002= SETFILE<<Enter path to source account:>>
/_METADATA_REPOSITORY_<<Enter pointer name:>> OVERWRITING
003= COPY FROM <<Enter pointer name:>>
TO _METADATA_REPOSITORY_ <<Enter file name:>> OVERWRITING
004= *
```

```
005= DTE_SETUP <<Enter file name:>> <<Enter DTE mode [LOGGING/ON.ERROR]:>>
006= *
*--:Bottom
```

The next paragraph copies the item from the source repository to the local repository, creates the metadata, and enables data type enforcement:

```
:AE VOC DTE_SETUP
Top.
*--: P
001: PA
002: *
003: DISPLAY
004: DISPLAY -----Creating METADATA of I2,Filename:>>-----
005: CREATE.METADATAI2,Filename:>> USING REPOSITORY
006: *
007: DISPLAY
008: DISPLAY * ----------
009: LIST.METADATA I2,Filename:>>
010: *
011: DISPLAY
012: DISPLAY -----Enable DTE for I2,Filename:>>-----
013: ENABLE.DTENF I2,Filename:>> ALL DTE.OPT I3, Enter DTE mode
 [LOGGING/ON.ERROR]:>>
014: *
015: DISPLAY
016: DISPLAY *-------------
017: LIST.DTENFI2,Filename:>>
018: *
Bottom.
```

The UniData VOC file

The VOC (vocabulary) file serves as the central repository for information about a UniData account. It contains direct information, such as commands, paragraphs, and keywords you can use, as well as pointers to menus, data and dictionary files, and cataloged programs. Each time a user starts a UniData session, the first file UniData reads is the VOC file for that account. Each account's VOC file serves as the entry point for the command processor. Each UniData account must contain a VOC file and its dictionary, D_VOC.

For more information about the UniData VOC file, see The UniData VOC file, on page 56.

UDT.OPTIONS

A UDT.OPTION is a toggle that alters UniData's personality. UDT.OPTIONS can cause system administration commands, UniBasic, UniQuery, and UniData SQL to behave in two different ways, depending on whether a particular 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 OFF
20 U_IGNLGN_LGTO OFF
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_PASCAL 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 ECL command `UDT.OPTIONS`.

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 their default settings.

Tip: If you want certain options in effect for users when they enter a UniData account, set `UDT.OPTIONS` in the login paragraph to customize these settings for each user.
When a `UDT.OPTION` affects the behavior of a command, you see the `UDT.OPTIONS` icon following the description of the command in UniData manuals.

UniData parser types

In certain circumstances, some UniQuery and UniBasic 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 for UniQuery statements, or the BASICTYPE for UniBasic programs. To distinguish among the types, UniData displays the following icons and indicates the ECLTYPE or BASICTYPE.

Note: ECLTYPE can be 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.
BASICTYPE can be U, P, R, or M. When it is U, UniBasic executes commands and functions NSconsistent with the UniData parser. When BASICTYPE is P, UniBasic executes commands and functions consistent with the Pick® parser. BASICTYPE R makes UniBasic consistent with the Advanced Revelation® parser. BASICTYPE M is consistent with the McDonnell Douglas or Reality® parser.

Chapter 2: Getting started with UniData

This chapter explains how to get started with UniData, including:

- Logging in and out of UniData
- Understanding the UniData demo account
- Creating a file
- Creating a dictionary record
- Adding a record to a file
- Understanding UniQuery
- Understanding BASICTYPE and ECLTYPE

Logging in and out of UniData

Verifying environment variables on UniData for UNIX

To access UniData, you must set environment variables for UDTHOME and UDTBIN, and *udtbin* should appear in your PATH. To determine if these environment variables are already established, enter the UNIX `printenv` command:

```
% printenv
HOME=/users/claireg
PATH=/usr/ud73/bin:/bin/posix:/bin:/usr/bin:/usr/contrib/bin:/usr/local/bin
LOGNAME=claireg
TERM=xterm
SHELL=/bin/csh
MAIL=/usr/mail/claireg
MANPATH=/usr/man:/usr/contrib/man:/usr/local/man
UDTHOME=/usr/ud73
UDTBIN=/usr/ud73/bin
%
```

If UDTHOME and UDTBIN do not appear in the display, or if udtbin does not appear in your PATH, set these environment variables as follows:

If you are using the UNIX C shell, use these commands:

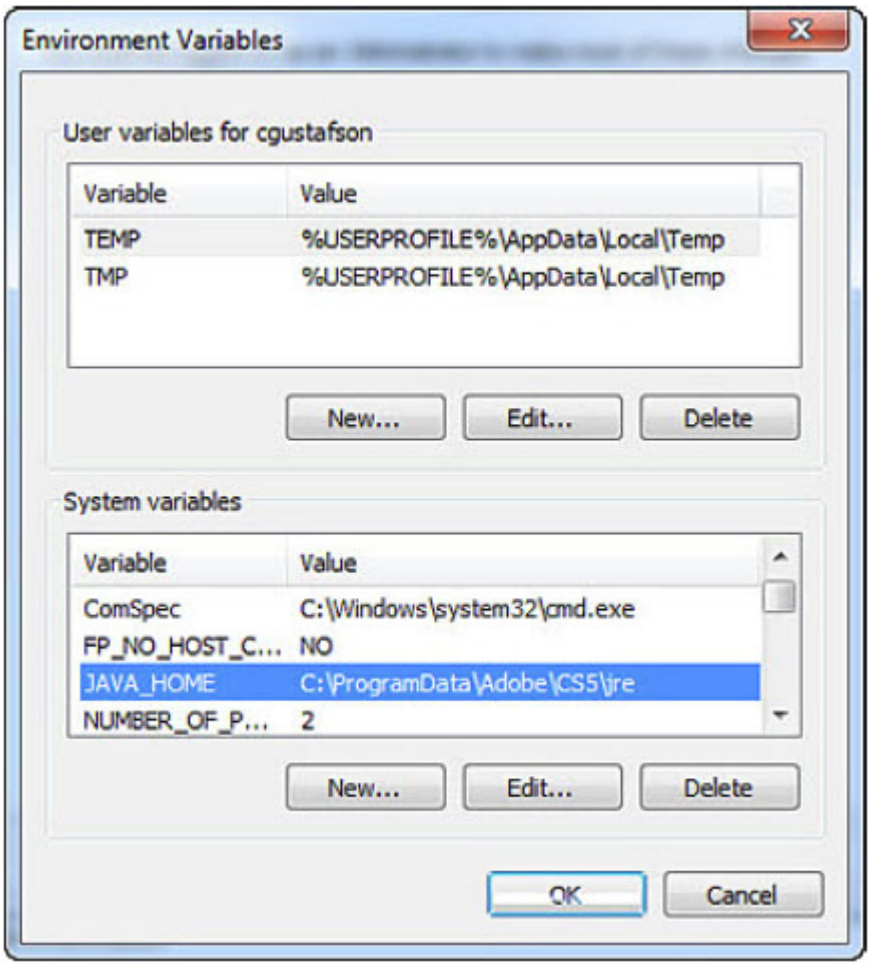
```
setenv UDTHOME /directory-name
setenv UDTBIN /directory-name
set path = ($path $UDTBIN)
```

If you are using the UNIX Bourne or Korn shell, use these commands:

```
UDTHOME=/directory-name;export UDTHOME
UDTBIN=/directory-name;export UDTBIN
PATH=$PATH:$UDTBIN;export PATH
```

Verifying environment variables on UniData for Windows platforms

UniData for Windows Platforms determines the values of UDTHOME and UDTBIN from the Registry, so it is not necessary to set them unless they differ from the Registry entry. If you want to set these variables, from the **Start** menu, click **Control Panel**, and then double-click **System**. From the **System** window, click **Advanced system settings** tab, then click **Environment Variables**. A window similar to the following example appears:



Click **New**. Enter the variable you want to change or add in the **Variable Name** box, then enter the value for that variable in the **Variable Value** box and click **OK**.

Accessing the UniData demo account

To access the UniData demo account, change to the `udthome/demo`directory on UniData for UNIX or the `udthome\demo`directory on UniData for Windows Platforms. If you are not using device licensing, at the operating system-level prompt, enter udt, as shown in the following example:

```
% cd demo
% udt
UniData Release 7.3 Build: (6059)
¬ Rocket Software, Inc. 1985-2011.
All rights reserved.
```

```
Current UniData home is C:\U2\ud73\.
Current working directory is C:\U2\ud73\Demo.
:
```

If you are using device licensing, enter udtts, rather than udt, to access your UniData account.

When you enter UniData, the screen displays the colon (:) prompt, called the ECL prompt. From the ECL prompt, you can enter UniData ECL and UniQuery commands.

Note: Depending on your application, you may not be placed at the ECL prompt when you enter UniData. See your system administrator for information about how to access the ECL prompt on your system.

Logging out of UniData

To exit a UniData session, you may enter the ECL `BYE`, `QUIT`, or `LO` command. Any one of these commands exits the UniData environment and returns to the host operating system prompt, as shown in the following example:

```
:QUIT
%
```

LOGIN and LOGOUT paragraphs

You can define LOGIN and LOGOUT paragraphs in the VOC files of your accounts to provide further control of users' environments. A typical LOGIN paragraph sets UDT.OPTIONS and then invokes an applications menu. A typical LOGOUT paragraph executes a routine that cleans up application files and exits the application in an orderly manner.

For more information about LOGIN and LOGOUT paragraphs, see UniData paragraphs, on page 102.

Contents of the UniData demo account

The UniData demo account consists of sample files and dictionaries intended for you to use to become familiar with UniData. This database uses a retail store as the foundation for the UniData dictionary and data files. 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.

To list all of the files in the demo account, enter the `LISTF` command from the ECL prompt, as shown in the following example:

```
:LISTF
LIST VOC F1 F2 F3 WITH F1 LIKE 'F...' OR F1 LIKE 'f...' OR F1 LIKE
'DIR...' OR F
1 LIKE 'dir...' OR F1 LIKE 'LF...' OR F1 LIKE 'lf...' OR F1 LIKE
'LD...' OR F1 L
IKE 'ld...' BY F1 BY @ID   16:03:38 Jun 17 2011 1
VOC..... F1..... F2.............. F3.............

AE_SCRATCH DIR        AE_SCRATCH      D_AE_SCRATCH
BP         DIR        BP              D_BP
BP_SOURCE  DIR        BP_SOURCE       D_BP_SOURCE
CTLG       DIR        CTLG            D_CTLG
SAVEDLISTS DIR        SAVEDLISTS      D_SAVEDLISTS
_HOLD_     DIR        _HOLD_          D__HOLD_
_PH_       DIR        _PH_            D__PH_
savedlists DIR        savedlists      D_savedlists
&MAP&      F          @UDTHOME/sys/_M @UDTHOME/sys/D_
                      AP_             _MAP_
&report&   F          _REPORT_        D__REPORT_
AE_DOC     F          @UDTHOME/sys/AE @UDTHOME/sys/D_
                      _DOC            AE_DOC
AE_XCOMS   F          @UDTHOME/sys/AE @UDTHOME/sys/D_
                      _XCOMS          AE_XCOMS
CATEGORIES F          CATEGORIES      D_CATEGORIES
CLIENTS    F          CLIENTS         D_CLIENTS
COURSES    F          COURSES         D_COURSES
CTLGTB     F          @UDTHOME/sys/CT @UDTHOME/sys/D_
                      LGTB            CTLGTB
CUSTOMER   F          CUSTOMER        D_CUSTOMER
ENGLISH.MS F          /disk1/ud61/sys /disk1/ud61/sys
G                     /ENGLISH.MSG    /D_ENGLISH.MSG
ERRMSG     F          @UDTHOME/sys/EN @UDTHOME/sys/D_
                      GLISH.MSG       ENGLISH.MSG
FCallBas   F          FCallBas        D_FCallBas
HELP.FILE  F          HELP.FILE       D_HELP.FILE
INVENTORY  F          INVENTORY       D_INVENTORY
MENUFILE   F          MENUFILE        D_MENUFILE
ORDERS     F          ORDERS          D_ORDERS
PARAGRAPHS F          PARAGRAPHS      D_PARAGRAPHS
STAFF      F          STAFF           D_STAFF
STATES     F          STATES          D_STATES
STUDENT    F          STUDENT         D_STUDENT
TAPES      F          TAPES           D_TAPES
VOC        F          VOC             D_VOC
VOC_SOURCE F          VOC_SOURCE      D_VOC_SOURCE
_DEBUG_    F          _DEBUG_         D__DEBUG_
_MAP_      F          @UDTHOME/sys/_M @UDTHOME/sys/D_
                      AP_             _MAP_
_REPORT_   F          _REPORT_        D__REPORT_
_SCREEN_   F          _SCREEN_        D__SCREEN_
__V__VIEW  F          __V__VIEW       D___V__VIEW

privilege  F          privilege       D_privilege
voc        F          VOC             D_VOC
38 records listed
```

The LISTF command is a predefined UniQuery sentence that lists all the VOC records of type F, LF, LD, or DIR.

Listing a dictionary file

Each UniData file must have a corresponding dictionary file. The name of the dictionary is listed in the last column of the LISTF display. Each record in a dictionary file defines the characteristics of an attribute in the data file. To list meaningful information from a UniData file, you need to know the records defined in the dictionary. For example, if you want to list names of the clients in the CLIENTS file, you need to know which dictionary records relate to names. In order to retrieve this information, use the LIST DICT *filename* command, as shown in the following example:

```
:LIST DICT CLIENTS
LIST DICT CLIENTS BY TYP BY @ID TYP LOC CONV NAME FORMAT SM ASSOC
12:24:01 Jun 02 2011 1
@ID............ TYP LOC.......... CONV NAME........... FORMAT SM
ASSOC.....

@ID             D               0       CLIENTS         10R     S
ADDRESS         D               4       Address         25T     MV
CITY            D               5       City            15T     S
COMPANY         D               3       Company Name    30T     S
COUNTRY         D               8       Country         15T     S
FNAME           D               1       First Name      15T     S
ID              D               0       Client #        10R     S
LNAME           D               2       Last Name       15T     S
PHONE_NUM       D               9       Phone Number    14R     MV
PHONE_ITEM
                                                                    S
PHONE_TYPE      D              10       Phone Category  10T     MV
PHONE_ITEM
                                                                    S
STATE           D               6       State/Territory 15T     S
ZIP_CODE        D               7       Postal Code     10R     S
@UQ             PH  NAME COMPANY
                    ADDRESS CITY
                    STATE ZIP COU
                    NTRY PHONE PH
                    ONE_TYPE
PHONE_ITEMS     PH  PHONE_NUM PHO
                    NE_TYPE
@ORIGINAL       SQ            @ID
@SYNONYM        SQ             ID
NAME            V   FNAME:" ":LNA       Name            30T     S
                    ME
PHONE           V   SUBR("PHONE_F       Phone Number    20R     MV
PHONE_ITEM
                    MT",PHONE_NUM                               S
                    ,COUNTRY)
ZIP             V   SUBR("PSTLCOD       Postal Code     10R     S
                    E_FMT",@RECOR
                    D)

NEXT_CLI_NO     X           10096       Next Client #
20 records listed
```

LIST DICT *filename* lists all records in the dictionary file in alphabetical order by type. LISTDICT *filename* (no space between LIST and DICT) lists the D-type records by attribute number, with all other dictionary records listed in alphabetical order by type, as follows:

```
:LISTDICT CLIENTS
SORT DICT CLIENTS TYP LOC CONV MNAME FORMAT SM ASSOC BY TYP BY LOC
BY @ID 12:40:11 Jun 02 2011 1
@ID............ TYP LOC.......... CONV MNAME.......... FORMAT SM
ASSOC.....

@ID             D               0       CLIENTS         10R     S
ID              D               0       Client #        10R     S
FNAME           D               1       First Name      15T     S
LNAME           D               2       Last Name       15T     S
COMPANY         D               3       Company Name    30T     S
ADDRESS         D               4       Address         25T     MV
CITY            D               5       City            15T     S
STATE           D               6       State/Territory 15T     S
ZIP_CODE        D               7       Postal Code     10R     S
COUNTRY         D               8       Country         15T     S
PHONE_NUM       D               9       Phone Number    14R     MV
PHONE_ITEM                                                      S
PHONE_TYPE      D              10       Phone Category  10T     MV
PHONE_ITEM                                                      S
@UQ             PH  NAME COMPANY
                    ADDRESS CITY
                    STATE ZIP COU
                    NTRY PHONE PH
                    ONE_TYPE
PHONE_ITEMS     PH  PHONE_NUM PHO
                    NE_TYPE
@ORIGINAL       SQ            @ID
@SYNONYM        SQ             ID
NAME            V   FNAME:" ":LNA       Name            30T     S
                    ME
PHONE           V   SUBR("PHONE_F       Phone Number    20R     MV
PHONE_ITEM                                                      S
                    MT",PHONE_NUM
                    ,COUNTRY)
ZIP             V   SUBR("PSTLCOD       Postal Code     10R     S
                    E_FMT",@RECOR
                    D)
NEXT_CLI_NO     X           10096       Next Client #
20 records listed
```

There are three items in the CLIENTS dictionary file that pertain to a name. Two of the dictionary items, FNAME and LNAME, are D-type records, while NAME is a virtual attribute. FNAME and LNAME point to attributes 1 and 2 of the data record, respectively. NAME concatenates FNAME and LNAME. The following example shows the display of these attributes:

```
:LIST CLIENTS FNAME LNAME NAME
LIST CLIENTS FNAME LNAME NAME 12:45:08 Jun 02 2011 1
CLIENTS... First Name..... Last Name......
Name.........................

      9999 Paul            Castiglione     Paul Castiglione
     10034 Fredrick        Anderson        Fredrick Anderson
      9980 Beverly         Ostrovich       Beverly Ostrovich
.
.
.
```

For information about virtual attributes, see Creating virtual attributes, on page 71.

Creating a UniData file

To create a UniData file, enter the CREATE.FILE command followed by the name of the file you want to create.

Syntax

```
CREATE.FILE [DICT | DATA] [DIR | MULTIFILE | MULTIDIR]
filename [,subfile] [modulo [,block.size.multiplier]][TYPE hashtype]
[DYNAMIC [KEYONLY | KEYDATA] [PARTTBL part_tbl]]
[RECOVERABLE] [OVERLFLOW]
```

Note: The PARTTBL options is not available on UniData for Windows Platforms.
For more information about the CREATE.FILE command, see the UniData Commands Reference.
For more information about file types, see UniData file types, on page 46.

In the following example, a static file called TEST_STATIC, and the dictionary file, D_TEST_STATIC, are created. TEST_STATIC is created with a modulo of 3:

```
:CREATE.FILE TEST_STATIC
modulos for file TEST_STATIC=3
Create file D_TEST_STATIC, modulo/1,blocksize/1024
Hash type = 0
Create file TEST_STATIC, modulo/3,blocksize/1024
Hash type = 0
Added "@ID", the default record for UniData to DICT TEST_STATIC
```

When a static file is created, an F-type record is written to the account's VOC file, as illustrated in the following example:

```
:AE VOC TEST_STATIC
Top of "TEST_STATIC" in "VOC", 3 lines, 27 characters.
*--: P
001: F
002: TEST_STATIC
003: D_TEST_STATIC
Bottom.
```

When you list a file or the dictionary of a file that you just created, the following screens display:

```
:LIST TEST_STATIC
LIST TEST_STATIC 15:26:41 Jun 02 2011 1
TEST_STATIC

No record listed.

:LIST DICT TEST_STATIC
LIST DICT TEST_STATIC BY TYP BY @ID TYP LOC CONV NAME FORMAT SM ASSOC
15:27:17 Jun 02 2011 1
@ID............ TYP LOC.......... CONV NAME........... FORMAT SM ASSOC.....

@ID     D   0   TEST_STATIC 10L         S
1 record listed
```

The data file does not contain any records immediately after creation. The @ID dictionary record is always created when the dictionary file is created, and is always attribute 0.

Conventions for naming files

UniData follows specific conventions for naming files and attributes:

- File name length
- Case sensitivity
- Special characters
- SQL and ODBC-related Restrictions

File name length

The length of a file name is determined by your operating system. The file name you choose must be, at most, 2 characters less than the longest name allowed by your host operating system. To determine the maximum file name length allowed by your system, enter the `ECL LIMIT` command. U_MAXFNAME displays the operating system file name limit.

Case sensitivity

All file names and attribute names are case sensitive within UniData. For example, “CUSTOMER,” “Customer,” and “customer” are all considered to be different names.

Special characters in file names

UniData allows some special characters in file names, but only if they occupy certain positions in the name.

The following special characters may appear in any position in the file name or attribute name:

```
& $ ! % @
```

The following special characters are allowed in a file or attribute name, but cannot be the first character:

```
+ - ^
```

Note: With UDT.OPTIONS 57 ON, UniData allows the number sign or pound sign (#) in an attribute name.
With UDT.OPTIONS 58 ON, UniData allows the colon (:) in an attribute name.

SQL- and ODBC-related restrictions

Use the following guidelines when naming files that must comply with ANSI SQL and ODBC standards:

- The length of the file name or attribute name must not be more than 30 characters.
- The first character in the file name or attribute name may be an alphabetic character, an underscore (_), or an “at” sign (@), except for those that must be accessed through ODBC, which does not allow the @ to be the first character.
- Except for the first character, all characters in the file name or attribute name can be alphanumeric characters, an “at” sign (@), an underscore (_), a dollar sign ($), a numeric character, or pound (#) sign.

- The dot (.) is not allowed in file names or attribute names.
- The file name or attribute name must not be a UniData SQL reserved word.
- The file name must not be an existing file, subtable, or view name.
- The value code specifier in the dictionary file cannot contain a null value or an empty string. Valid specifications are:
  - S — singlevalued
  - MV — multivalued
  - MS — multi-subvalued
- If an attribute name is part of an association in the dictionary file, that association name must exist in the dictionary as one of the phrase (PH) attributes.
- An association cannot contain a singlevalued (S) attribute in the dictionary file.

Adding records to a dictionary file

Once you have created a UniData file, you may add records to its dictionary file by using UniEntry or any valid UniData text editor.

Before you enter any dictionary records, you must determine the layout for the data file. Consider a file that stores information about individuals and their children. In the following examples, you will enter four dictionary records.

@ID	Type	Location	Conversion code	Display name	Format	Value code	Association
NAME	D	1	N/A	Name	25L	S	N/A
KIDS	D	2	N/A	Children	20L	MV	KID
AGES	D	3	MD0	Age	3R	MV	KID
KID	PH	KIDS AGES	N/A	N/A	N/A	N/A	N/A

In the following examples, the AE editor is used to enter the dictionary records for the preceding attributes:

```
:AE DICT TEST_STATIC
Record ids > NAME
Top of New "NAME" in "TEST_STATIC".
*--: I
001= D
002= 1
003=
004= Name
005= 25L
006= S
007=
*--:
*--: FI
Filed "NAME" in file "TEST_STATIC"
Record ids > KIDS
Top of New "KIDS" in "TEST_STATIC".
*--: I
001= D
002= 2
003=
004= Children
```

```
005= 20L
006= MV
007= KID
*--: FI
Filed "KIDS" in file "TEST_STATIC".
Record ids > AGES
Top of New "AGES" in "TEST_STATIC".
*--: I
001= D
002= 3
003= MD0
004= Age
005= 3R
006= MV
007= KID
*--: FI
Filed "AGES" in file "TEST_STATIC".
Record ids > KID
Top of New "KID" in "TEST_STATIC".
*--: I
001= PH
002= KIDS AGES
*--: FI
Filed "KID" in file "TEST_STATIC".
```

In the preceding example, NAME is a singlevalued D-type dictionary record. It is located in attribute 1 of the data file. The column heading for display is Name, and it is 25 characters in length, and is left justified.

KIDS and AGES are both defined as multivalued fields, and are located in attributes 2 and 3 of the data file, respectively. The column display name for KIDS is defined as Children, while the column display name for AGES is defined as Age. The attribute KIDS is 20 characters in length and is left justified. The attribute AGES is 3 characters in length, right justified, and has a conversion code of MD0. KIDS and AGES are included in the association defined as KID.

Note: For more information about singlevalued and multivalued attributes, see UniData relational database concepts, on page 10. For more information about conversion codes, see Dictionary conversion codes and format options, on page 163.

The following screen shows the contents of the dictionary file after adding these records:

```
:LIST DICT TEST_STATIC
LIST DICT TEST_STATIC BY TYP BY @ID TYP LOC CONV NAME FORMAT SM
ASSOC 18:18:03 Jun 22 2011 1
@ID............ TYP LOC.......... CONV NAME........... FORMAT SM
ASSOC.....

@ID             D               0      TEST_STATIC     10L    S
AGES            D               3 MD0  Ages            3R     M
KID
KIDS            D               2      Children        20L    M
KID
NAME            D               1      Name            25L    S
KID             PH      KIDS AGES
5 records listed
```

Adding records to a UniData file

Once you have defined the attributes of the records in a file, you are ready to populate the data file. Ordinarily, you may use an application program to add records. Records may be written to a file through UniBasic, any valid UniData text editor, UniEntry, or UniData SQL. This section illustrates entering data with the AE editor.

Note: For more information about UniBasic, see the*Developing UniBasic Applications* manual. For information about UniEntry, see Using UniEntry, on page 169. For information about UniData SQL, see *Using UniData SQL*.

In the following example, AE opens the CLIENTS demo database file and prompts for a record ID:

```
:AE CLIENTS
Record ids >
```

The next step is to enter the record ID, or @ID. It must be a unique identifier for the record, and can contain alpha and/or numeric characters; it cannot be more than 126 characters in length.

After you enter the record ID, enter I for input mode in the AE editor, and begin entering data. When entering multivalued attributes, separate the values with ^253, representing a value mark. When entering multi-subvalued attributes, separate the multi-subvalues with ^252 for a subvalue mark. If you want to leave an attribute empty, input one space followed by a carriage return.

Tip: On UniData for UNIX, the value mark may look like a right brace (}), and the subvalue mark may look like the UNIX pipe symbol (|) when you display a multivalued attribute with the AE editor. When entering data, do not enter a right curly bracket as a value mark or a pipe symbol for a subvalue mark. You must use ^253 for a value mark and ^252 for a subvalue mark.

```
:AE TEST_STATIC 1
Top of New "1" in "CLIENTS".
*--: I
001= Jane Robinson
002= Chad^253John^253Elizabeth
003= 7^2534^2531
*--: fi
Filed "1" in file "TEST_STATIC".
```

In the previous example, Jane Robinson is entered as the NAME, Chad, John and Elizabeth are the KIDS, and 7, 4, and 1 are the AGES. Since KIDS and AGES are associated, Chad is 7, John is 4, and Elizabeth is 1.

Tip: To view nonprinting characters in AE, enter the ^ key. To turn off viewing nonprintable characters, enter the ^ key again.

Overview of UniQuery

UniQuery is a language that enables you to perform queries against 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.

Every UniQuery statement must contain a verb (command) followed by the desired file name. All other elements of a UniQuery statement are optional, depending on the result you want to achieve. The most commonly used verbs in UniQuery are defined in the following table.

Command	Function
COUNT	Displays a count of the records meeting the defined selection criteria in a file. If no selection criteria is defined, counts the number of records in the file.
LIST	Lists the records in a file meeting the specified selection criteria, along with the display attributes requested. If no selection criteria or display attributes are defined, lists the @ID of the records in the file. If no sort criteria is specified, lists the records in the order they are stored.
SELECT	Creates a list of record IDs that meet the specified selection criteria. If no selection criteria is defined, creates a list of all record IDs in the file. The resulting list is referred to as a select list.
SORT	Lists the records in a file meeting the specified selection criteria sorted by @ID. If no selection criteria is specified, lists all the records in the file sorted by @ID.
SSELECT	Creates a list of record IDs meeting the specified selection criteria sorted by @ID.
SUM	Sums the values of selected numeric attributes meeting the specified selection criteria.
USHOW	Lists the records in a file meeting the specified selection criteria, along with the display attributes requested, in an interactive display. You may scroll forward and backward in the screen, and select specific records in the display to create a select list.

See Null value handling, on page 126 for special considerations when your data contains the null value.

Creating a UniQuery statement

The following is the syntax for a UniQuery statement:

Syntax

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

The following table describes the elements of a UniQuery statement.

Function	Description
command	Names the process to perform.
DICT	Stipulates the dictionary file rather than the data file. Must follow the command.
filename	Names the file on which to perform the operation(s). You can specify only one file name in a UniQuery statement. The file name must immediately follow the command, except in the case where the DICT file is specified.
display_attributes | ALL	Indicates the dictionary attributes of the specified file name to display. The ALL keyword displays all D-type attributes in a file.
record_IDs	Specifies the record IDs to use in the UniQuery statement. If you do not specify any record IDs, UniQuery performs the operations against all record IDs in the file.

Function	Description
selection_criteria	States one or more conditions that must be met for retrieving or displaying data.
sorting_criteria	Indicates how to sort the selected records.
format_options	Specifies how to format a report, including defining attribute names, page breaks, headers, and footers.
control_options	Includes keywords that control report output.

Note: The following examples were created after more data records were added to the TEST_STATIC file.

In the following example, the UniQuery statements lists the contents of the TEST_STATIC file using the dictionary records you previously created:

```
:LIST TEST_STATIC NAME KIDS AGES
LIST TEST_STATIC NAME KIDS AGES 15:38:28 Jun 25 2011 1
TEST_STATIC Name..................... Children..... Ages

3           Amy Wade                 Bryan              3
                                     Terri              1
1           Jane Robinson            Chad               7
                                     John               4
                                     Elizabeth          1
2           Bruce Simpson            Carly             10
                                     Stephen            4
3 records listed
```

The preceding example displays records in the TEST_STATIC file using the UniQuery LIST command, followed by the file name and the selected dictionary records. UniQuery displays the records in the order they are stored in the file because no sorting criteria was specified. UniQuery displays the @ID even though it was not one of the attribute names in the query. UniData displays the @ID in a query, unless you specifically suppress it. Notice that UniQuery displays each value in the multivalued attribute on a separate line.

For more information on UniQuery commands, see *Using UniQuery*.

The same information in the above example can be displayed by substituting the dictionary record KID for KIDS and AGES, since the KID association is defined in the dictionary of TEST_STATIC:

```
:LIST TEST_STATIC NAME KID
LIST TEST_STATIC NAME KIDS AGES 16:07:16 Jun 25 2011 1
TEST_STATIC Name..................... Children..... Ages
3           Amy Wade                 Bryan              3
                                     Terri              1
1           Jane Robinson            Chad               7
                                     John               4
                                     Elizabeth          1
2           Bruce Simpson            Carly             10
                                     Stephen            4
3 records listed
```

You can specify sorting and formatting options to produce a more readable report. In the following example, UniQuery sorts the report by NAME, suppresses the @ID, and double-spaces the report:

```
:LIST TEST_STATIC BY NAME NAME KIDS AGES ID.SUP DBL.SPC
LIST TEST_STATIC BY NAME NAME KIDS AGES ID.SUP DBL.SPC 11:15:13
Jun 26 2011 1
Name..................... Children............ Ages

Amy Wade                 Bryan                 3
                         Terri                 1

Bruce Simpson            Carly                10
                         Stephen               4

Jane Robinson            Chad                  7
                         John                  4
                         Elizabeth             1

3 records listed
```

If you want to total the ages of the children, you can do this by adding the TOTAL keyword, as shown in the following example:

```
:LIST TEST_STATIC BY NAME NAME KIDS TOTAL AGES ID.SUP DBL.SPC
LIST TEST_STATIC BY NAME NAME KIDS TOTAL AGES ID.SUP DBL.SPC
11:20:35 Jun 26 2011 1
Name..................... Children............ Ages

Amy Wade                 Bryan                 3
                         Terri                 1

Bruce Simpson            Carly                10
                         Stephen               4

Jane Robinson            Chad                  7
                         John                  4
                         Elizabeth             1

                                              ====
TOTAL                                           30
3 records listed
```

ECLTYPE and BASICTYPE

BASICTYPE is a UniBasic or ECL command that determines the type of parser UniBasic uses to compile programs. In certain circumstances, some UniBasic commands, keywords, and functions operate differently based on the kind of parser you use with UniData. UniData provides multiple parsers for backward compatibility.

Note: BASICTYPE is U, P, R, or M. When a program is compiled in BASICTYPE U, UniBasic executes commands and functions consistent with the UniData parser. When BASICTYPE is P, UniBasic executes commands and functions consistent with the Pick® parser. BASICTYPE R makes UniBasic consistent with the Advanced Revelation® parser. BASICTYPE M is consistent with the McDonnell Douglas or Reality® parser.

In the following example, the program segment searches the string EXAMPLE for the number of occurrences of the substring II, and returns the number 2. This program is compiled in BASICTYPE U:

```
:AE BP COUNT
```

```
Top of "COUNT" in "BP", 3 lines, 78 characters.
*--: P
001: EXAMPLE = "JAWSII,ROCKYIII,STARTREKIV"
002: IC = COUNT(EXAMPLE,"II")
003: PRINT "IC = ":IC
*--: FIBR
Filed "COUNT" in file "BP" unchanged.
Compiling Unibasic: BP/COUNT in mode 'u'.
compilation finished
IC = 2
```

In the next example, the program segment performs the same function as the preceding example, but returns a result of 3. The different result is due to this program having been compiled in BASICTYPE P. In BASICTYPE P, the COUNT function behaves differently and produces different results than if the program had been compiled in BASICTYPE U.

```
001: $BASICTYPE "P"
002: EXAMPLE = "JAWSII,ROCKYIII,STARTREKIV"
003: IC = COUNT(EXAMPLE,"II")
004: PRINT "IC = ":IC
*--: FIBR
Filed "COUNT" in file "BP" unchanged.

Compiling Unibasic: BP/COUNT in mode 'u'.
Basictype is changed, BP/COUNT is compiling in mode 'p'
compilation finished
IC = 3
```

BASICTYPE can be set in a UniBasic program or from ECL. If you set BASICTYPE in a UniBasic program, the `$BASICTYPE` command must be the first noncomment statement in the program or subroutine.

To set BASICTYPE from ECL, enter `BASICTYPE` followed by the desired type from the ECL command line. To determine the current BASICTYPE setting, enter the `BASICTYPE` command with no options, as shown in the following example:

```
:BASICTYPE
:BASICTYPE
BASICTYPE p
```

`BASICTYPE` can also be used to determine which parser was used to compile a program, as illustrated in the following example:

```
:BASICTYPE BP EXAMPLE
Basic program 'BP/_EXAMPLE' was compiled with mode 'u'.
:
```

Understanding ECLTYPE

`ECLTYPE` is an ECL command that determines which parser UniData uses when processing UniQuery statements. As with the `BASICTYPE` command, UniData provides different parsers for backward compatibility.

Note: `ECLTYPE` can be 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.

ECLTYPE may be set either at the ECL prompt or by setting UDT.OPTIONS 2 ON. The default setting is ECLTYPE U, which is the UniData parser. The following example illustrates how to display the current setting for ECLTYPE and how to set ECLTYPE to P from the ECL prompt:

```
:ECLTYPE
2 U_PSTYLEECL OFF
:
:ECLTYPE P
:ECLTYPE
2 U_PSTYLEECL ON
:
```

In the next example, ECLTYPE is set to P by setting UDT.OPTIONS 2 ON:

```
:ECLTYPE
2 U_PSTYLEECL OFF
:
:UDT.OPTIONS 2 ON
:ECLTYPE
2 U_PSTYLEECL ON
:
```

Chapter 3: UniData file types

This chapter provides an overview of the different file types used by UniData, including static, dynamic, and sequentially hashed files, directory files, multilevel files, alternate key indexes, and recoverable files.

UniData hashed files

Hashed files are binary files that cannot be viewed at the operating system level or read by text editors external to UniData. Each UniData hashed file consists of a file header and one or more groups of data.

Hashing allows efficient access to a record by translating its primary key into its location in a data file. UniData supports three proprietary hashing algorithms, which determine what data groups contain which records. These hashing algorithms are:

- Static hashing — produces binary data files with a fixed modulo (number of groups).
- Dynamic hashing — produces directory files whose modulo is not fixed, but changes as records are added or deleted from the data portion of the file.
- Sequential hashing — produces directory files whose modulos are fixed.

How hashing works

When a record is added to a file group, UniData stores the primary key separate from the data. This method enhances performance, because the record ID can always be found with a single logical disk read, unless the file is in level 2 overflow.

Once the record ID is found, UniData locates the record through a linked list. The primary key list grows from the middle of the group header block toward the pointer array. UniData stores the data portion (without the primary key) as a stream of data, called a dynamic array, that starts at the middle of the block and grows toward the end of the block.

An array of pointers at the top of the group header block gives the size and offset of the data portion of each record. This array grows toward the primary key list.

File overflow

When a group is not full, UniData can get the group number directly and access the record with one disk read. However, when a group is full, UniData locates an empty block and appends it to the file. UniData stores records that do not fit into the original group in this overflow block. When UniData reads records stored in the overflow blocks, it must read the original group and read the overflow block to search for data. If files are poorly sized, UniData reads several disk blocks to access data, which increases record retrieval time.

There are two different levels of overflow:

- **Level 1.** With level 1 overflow, data has outgrown the block. The block size may also be smaller than the record size. Level 1 overflow has little impact on system performance.
- **Level 2.** With level 2 overflow, the primary keys have outgrown the block. Level 2 overflow severely impacts performance. Although level 2 overflow does not cause file corruption, the risk for corruption increases when a file is in level 2 overflow should you experience a system failure.

The common causes of level 1 and level 2 overflows are:

- **Data Outgrows the Block.** If too many records are hashed to the same group, a level 1 overflow occurs. Data records reside in overflow blocks attached to the data file.
- **Block Size is Smaller than the Record Size.** If the record to be written is larger than the block size, a level 1 overflow is created, and the original block is left empty. The data spans multiple blocks, which causes UniData to perform multiple reads to locate each record.
- **Primary Keys Outgrow the Block.** If too many records are hashed to the same group, the primary key information eventually overflows the group, causing a level 2 overflow. Level 2 overflow causes UniData to create another block (a secondary group header block) for the overflowing keys and their data. This block is identical to the original block.

Static hashed files

When you create a static hashed file, you specify the file size parameters, including number of groups (modulo) and the block size. For files that do not change size significantly, (for instance, files used for query purposes only), the static file is efficient. If, however, your data overflows the groups (level 1 overflow), or if the primary keys overflow (level 2 overflow), UniData adds overflow blocks to the file to accommodate the data and keys. Accessing and updating records becomes resource-intensive and performance suffers. UniData provides utilities to resize static files by adding more groups (increasing the modulo) or by making the groups larger (changing the block size).

Static files should be used for files that are a fixed size and are not expected to grow substantially.

You create static files using the `CREATE.FILE` command.

Syntax

```
CREATE.FILE [DICT | DATA] [DIR | MULTIFILE | MULTIDIR]
filename [,subfile] [modulo [,block.size.multiplier]][TYPE hashtype]
[DYNAMIC [KEYONLY | KEYDATA] [PARTTBL part_tbl]]
[RECOVERABLE] [OVERFLOW]
```

Note: The PARTTBL option is not supported on UniData for Windows platforms.

To create the DICT and DATA portions of a static file, enter `CREATE.FILE` followed by the name of the file you are creating, as shown in the following example:

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

UniData creates the file TEST and the dictionary file, D_TEST, in the current directory. The VOC file is updated with a record for TEST, as shown in the following example:

```
:!ls -l *TEST*
-rw-rw-rw-   1 claireg   unisrc   2048 Nov 15 18:02 D_TEST
-rw-rw-rw-   1 claireg   unisrc   104448 Nov 15 18:02 TEST

:AE VOC TEST
Top of "TEST" in "VOC", 3 lines, 13 characters.
*--: P
001: F
002: TEST
```

```
003: D_TEST
Bottom.
*--:
```

For detailed information about `CREATE.FILE`, see the *UniData Commands Reference*.

Dynamic hashed files

Dynamic hashed files are system-level directory files, each containing a data and overflow file. If a dynamic file has an alternate index, the index file is also stored in this directory. On UniData for UNIX, if a dynamic file has a per-file part table, the part table is stored in this directory. UniData dynamic files can grow and shrink with respect to modulo (number of groups) as records are added and deleted. Dynamic files can also automatically expand beyond the limits of a single file system on UniData for UNIX.

Dynamic files should be used for files that are not fixed in size.

When you create a dynamic file, you specify the file size parameters, including number of groups (modulo) and the block size. The modulo you specify becomes the “minimum modulo” or “base modulo” of the file. The number of groups in the dat001 file is never less than the minimum modulo.

Tip: If you are creating a dynamic file, pick a minimum modulo that is appropriate for the number of records you expect the file to contain. If you start with a minimum modulo that is too small, records will not be hashed efficiently to the groups in the file.

The dat001 file

The dat001 file is also called the primary data file. As you add records to a dynamic file, UniData hashes the keys to groups in dat001. As the file fills up, UniData adds additional groups to the dat001 file. If the current file system fills up or if dat001 grows larger than a UniData limit, UniData creates a dat002 file. If dat002 resides on another file system, UniData creates a symbolic link to the dat002 file in the original dynamic file directory.

The over001 file

As you add records to a dynamic file, whenever the space reserved for data in a group in the primary file gets full, UniData writes the excess data into blocks in the over001 file. Registers within UniData track how blocks in over001 are linked to groups in dat001. If the current file system becomes full, or over001 grows larger than a UniData limit, UniData creates an over002 file. If the over002 file resides in a different file system than the over001 file, UniData creates a symbolic link to over002 in the original dynamic file.

You can specify the OVERFLOW option with the CREATE.FILE command to create a dynamic file with an overflow file for every dat file. For example, over001 corresponds to dat001, over002 corresponds to dat002, and so forth. When you clear the dynamic file, UniData maintains this overflow structure.

You create dynamic files by using the CREATE.FILE command.

Syntax

```
CREATE.FILE [DICT | DATA] [DIR | MULTIFILE | MULTIDIR]
filename [,subfile] [modulo [,block.size.multiplier]][TYPE hashtype]
[DYNAMIC [KEYONLY | KEYDATA] [PARTTBL
```

part_tbl]][RECOVERABLE] [OVERFLOW]

Note: The PARTTBL option is not available on UniData for Windows Platforms.

To create the DICT and DATA portions of a dynamic file, enter CREATE.FILE followed by the name of the file you are creating and the DYNAMIC option, as shown in the following example:

```
:CREATE.FILE DYNAMIC_TEST DYNAMIC
modulos for file DYNAMIC_TEST=101
Create file D_DYNAMIC_TEST, modulo/1,blocksize/1024
Hash type = 1
Create dynamic file DYNAMIC_TEST, modulo/101,blocksize/1024
Hash type = 1
Added "@ID", the default record for UniData to DICT DYNAMIC_TEST.
```

UniData creates the directory DYNAMIC.FILE and the dictionary file, D_DYNAMIC.FILE, in the current directory. The DYNAMIC.FILE directory contains the dat001 and over001 portions of the file, along with the index portion of the file (idx001) and the per-file part table (parttbl), if they exist. The VOC file is updated with a record for DYNAMIC.FILE, as shown in the following example:

```
:!ls -l *DYNAMIC*
-rw-rw-rw-      1 claireg   unisrc      2048 Nov 15 18:16
D_DYNAMIC_TEST

DYNAMIC_TEST:
total 226
-rw-rw-rw-      1 claireg   unisrc      104448 Nov 15 18:16 dat001
-rw-rw-rw-      1 claireg   unisrc        1024 Nov 15 18:16 over001
:AE VOC DYNAMIC_TEST
Top of "DYNAMIC_TEST" in “VOC”, 3 lines, 29 characters.
*--: P
001: F
002: DYNAMIC_TEST
003: D_DYNAMIC_TEST
Bottom.
*--:
```

For detailed information about CREATE.FILE, including the KEYONLY, KEYDATA, and PARTTBL keywords, see the *UniData Commands Reference*. For more information about managing dynamic files, see *Administering UniData on UNIX* or *Administering UniData on Windows Platforms*.

Sequentially hashed files

A sequentially hashed file has the same structure as a dynamic file, but all records are stored sequentially based on the primary key. The modulo (number of groups) for a sequentially hashed file is fixed, it does not grow and shrink as records are added or deleted.

Sequentially hashed files are created by converting from existing UniData static or dynamic files. You specify a percentage of the file that you want to remain empty to allow for growth. Although the structure for a sequentially hashed file is the same as a dynamic file, the modulo is fixed.

A sequentially hashed file is used for files where the majority of access is based on the primary key, and has a hash type of 2.

The dat001 file

The dat001 file is also called the primary data file. As you add records to a sequentially hashed file, UniData hashes the keys, based on information in the gmekey file, to groups in dat001. If your data overflows the group (level 1 overflow), UniData writes the overflow data to the over001 file.

The over001 file

As you add records to a sequentially hashed file, whenever the space reserved for data in a group in the primary file gets too full, UniData writes the excess data into blocks in over001. Registers within UniData track how blocks in over001 are linked to groups in dat001. If over001 gets too large, UniData adds additional blocks to it. If the current file system becomes full, or over001 grows larger than a UniData limit, UniData creates an over002 file. If the over002 file resides in a different file system than the over001 file, UniData creates a link to over002 in the original sequentially hashed file.

If the sequentially hashed file has level 2 overflow, the file should be rebuilt using the `shfbuild` command.

The gmekey file

Each sequentially hashed file contains a static, read-only file called the gmekey file. This file is read into memory when you open a sequentially hashed file. The gmekey file contains information about the type of keys in the file (alpha or numeric), and controls which group a record is hashed to when it is written.

You create a sequentially hashed file by converting an existing dynamic or static file with the shfbuild command:

Syntax

shfbuild [-a |-k] [-n | -t] [-f] [-e *empty percent*] [-m *modulo*] [-b *block size multiplier*] [-i *infile*] *outfile*

The following table describes each parameter of the syntax.

Parameter	Description
-a	Only rebuild the last group of the sequentially hashed file. UniData splits the last group into groups according to the records in the group. If you use this option, the *outfile* should be the name of the sequentially hashed file. Do not specify *infile*.
-k	Build the gmekey file only. If you use this option, *outfile* should be the name of the sequentially hashed file. Do not specify *infile*. UniData rebuilds the gmekey file according to the keys in each group of *outfile*.
-n/-t	Force the *outfile* to be in numeric or alphabetic order. By default, the order of *outfile* is determined by the *infile* primary key type. If *infile* is a sequentially hashed file, UniData uses the same order in *outfile*. If *infile* is not a sequentially hashed file, the order of *outfile* is determined by the justification of the @ID of the *infile* dictionary record. If it is right justified, it is numeric. Otherwise, it is alphabetic. If you use the -a or the -k option, these options have no effect.
-f	Force *outfile* to truncate before being built.
-m	Specifies the new modulo of *outfile*.

Parameter	Description
-e	Empty percent. This is a number between 0 and 99 which indicates how much space in the rebuilt groups to reserve. UniData calculates the new modulo of the file from *empty_percent* and the number of records in the rebuilt groups. If you do not specify -e or -m, UniData rebuilds the sequentially hashed file according to the default empty percent of 20.
-b	Specifies the block size of the sequentially hashed file in kilobytes.
-i *infile*	Load the contents from *infile* instead of *outfile*. *infile* can be any type of UniData file.
outfile	The name of the output file.

To convert an existing file, execute the shfbuild command from the system level prompt, as shown in the following example:

```
% shfbuild -m 59 SEQUENTIAL
175 keys found from SEQUENTIAL.
175 records appended to SEQUENTIAL; current modulo is 59.
```

After converting a file to a sequentially hashed file, you must manually enter a file pointer in the VOC file in order to access the sequentially hashed file, as shown in the following example:

```
:AE VOC SEQUENTIAL
Top of New "SEQUENTIAL" in "VOC".
*--: I
001= F
002= SEQUENTIAL
003= D_SEQUENTIAL
*--: FI
Filed "SEQUENTIAL" in file "VOC".
```

UniData directory files

A UniData directory file, also called a DIR file, is a directory that contains text or data files. Each text or data file is a UniData record.

The following kinds of text files are considered by UniData to be records in DIR files:

- UniBasic source code files — usually located in the BP file
- COMO files — log of a UniData session, located in the _PH_ file
- Saved select lists — located in the SAVEDLISTS file
- Print hold files — located in the _HOLD_ file
- Phantom log records — located in the _PH_ file

You create directory files using the `CREATE.FILE` command.

Syntax

```
CREATE.FILE [DICT | DATA] [DIR | MULTIFILE | MULTIDIR]
filename [,subfile] [modulo [,block.size.multiplier]][TYPE hashtype]
[DYNAMIC [KEYONLY | KEYDATA] [PARTTBL part_tbl]]
[RECOVERABLE] [OVERFLOW]
```

Note: The PARTTBL option is not available on UniData for Windows Platforms.

To create the DICT and DATA portions of a directory file, enter CREATE.FILE followed by the DIR option and the name of the file you are creating. After creating a DIR file, the first attribute of the VOC file will be DIR, as shown in the following example:

```
:CREATE.FILE DIR TEST.DIR
Create DIR type file TEST.DIR.
Create file D_TEST.DIR, modulo/1,blocksize/1024
Hash type = 0
Added "@ID", the default record for UniData to DICT TEST.DIR.
:AE VOC TEST.DIR
Top of "TEST.DIR" in "VOC", 3 lines, 23 characters.
*--: P
001: DIR
002: TEST.DIR
003: D_TEST.DIR
Bottom.
```

For detailed information about `CREATE.FILE`, see the *UniData Commands Reference*.

Multilevel files

A UniData multilevel (LF-type) file is a directory that contains one or more UniData hashed files. All the UniData hashed files share a common dictionary. To access a record, you must specify both the directory and the hashed file where the record resides.

You create multilevel files using the `CREATE.FILE` command. To create the DICT and DATA portions of a multilevel file, enter `CREATE.FILE` followed by the MULTIFILE option and the names of the files you are creating. After creating a multilevel file, the first attribute of the VOC file will be LF, as shown in the following example:

Syntax

`CREATE.FILE` [DICT | DATA] [DIR | **`MULTIFILE`** | MULTIDIR]
`filename` [*`,subfile`*] [*`modulo`* [*`,block.size.multiplier`*]][TYPE *`hashtype`*]
[DYNAMIC [KEYONLY | KEYDATA] [PARTTBL *`part_tbl`*]]
[RECOVERABLE]

Note: The PARTTBL option is not available on UniData for Windows Platforms.

The following example illustrates creating a multilevel file:

```
:CREATE.FILE MULTIFILE MULTI1,MULTI2
modulos for file MULTI1,MULTI2=59
Create DIR type file MULTI1.
Create file D_MULTI1, modulo/1,blocksize/1024
Hash type = 0
Added "@ID", the default record for UniData to DICT MULTI1.
Create file MULTI1/MULTI2, modulo/59,blocksize/1024
Hash type = 0
Added "@MULTI2" to DICT MULTI1.

:AE VOC MULTI1
Top of "MULTI1" in "VOC", 3 lines, 18 characters.
*--: P
001: LF
002: MULTI1
003: D_MULTI1
Bottom.
```

```
:!ls -l MULTI1
total 120
-rw-rw-rw-      1 claireg   unisrc      61440 Nov 18 13:09 MULTI2
```

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

UniData supports multilevel files to simplify conversion for legacy applications. The multilevel structure allows a number of data files to share a single dictionary. However, multilevel files are less efficient than ordinary static, dynamic, or sequentially hashed files. This structure requires more system resources to read and update these files. For this reason, we recommend using ordinary static, dynamic, or sequentially hashed files rather than multilevel files whenever possible. You can share a single dictionary among UniData files by modifying the VOC entries for each file to reference the same dictionary.

Multilevel directory files

A UniData multilevel directory (LD-type) file is a system-level directory. The directory contains one or more subdirectories (UniData DIR type files). All of the DIR files share the same dictionary. To access a record, you must specify both the multilevel directory file and the DIR file where the record resides. Each record in a multilevel directory is considered by UniData to be a file.

You create multilevel directory files using the CREATE.FILE command. To create the DICT and DATA portions of a multilevel directory file, enter CREATE.FILE followed by the MULTIDIR option and the names of the directory files you are creating.

Syntax

```
CREATE.FILE [DICT | DATA] [DIR | MULTIFILE | MULTIDIR]
filename [,subfile] [modulo [,block.size.multiplier]][TYPE hashtype]
[DYNAMIC [KEYONLY | KEYDATA] [PARTTBL part_tbl]]
[RECOVERABLE] [OVERFLOW]
```

Note: The PARTTBL option is not available on UniData for Windows Platforms.

The first attribute of the VOC record for multilevel directory files LD, as shown in the following example:

```
:CREATE.FILE MULTIDIR MULTIDIR1,MULTIDIR2
Create DIR type file MULTIDIR1.
Create file D_MULTIDIR1, modulo/1,blocksize/1024
Hash type = 0
Added "@ID", the default record for UniData to DICT MULTIDIR1.
Create DIR file MULTIDIR1,MULTIDIR2.
Added "@MULTIDIR2" to DICT MULTIDIR1.
:
:AE VOC MULTIDIR1
*--: P
001: LD
002: MULTIDIR1
003: D_MULTIDIR1
Bottom.
*--:
```

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

UniData supports multilevel directory files to simplify conversion for legacy applications. However, multilevel directory files are less efficient than ordinary DIR-type files. This structure requires more

system resources to read and update than static, dynamic, or sequentially hashed files. For this reason, we recommend using ordinary DIR files rather than multilevel directory files whenever possible. You can share a single dictionary among UniData DIR files by modifying the VOC records for each file to reference the same dictionary.

Index files and index log files

An alternate key index reduces processing time when UniData searches through files to find specific records. The alternate index consists of an alternate key, a list of values for the alternate key, and an associated list of primary keys for each record.

UniData creates an index file whenever a user creates the first alternate key index on a UniData hashed file. Index information is stored in B+ tree format. UniData index files are system-level data files.

Note: Regardless of how many alternate key indexes users create for a data file, UniData creates a single index file.

The `ECL CREATE.INDEX` command creates the index file. The `ECL BUILD.INDEX` command populates the index. The `ECL DELETE.INDEX` command (with the ALL option) removes the index file.

By default, each time a UniData data file is updated, its associated indexes are updated. You can turn off automatic indexing on one or more data files (using the ECL DISABLE.INDEX command) to speed performance during periods of heavy activity on your system. If you turn off automatic indexing for a file, UniData stores all its index updates in an index log file. The `ECL UPDATE.INDEX` command applies updates from index logs to indexes in batch mode, and the `ECL ENABLE.INDEX` command turns automatic updating back on. If an update is made to a file when the index is disabled, an index log file is created, if one is not already there. Either the `ECL CLEAR.FILE` or `DELETE.INDEX` command (with the ALL option) removes the index log file.

For a dynamic or sequentially hashed file, the index and the index log files are located in the file directory. The file idx001 is the index file, and xlog001 is the index log file. For a static file, the index is located in the account where the file resides. The file X_*filename* is the index file. On UniData for UNIX, x_*filename* is the index log file. On UniData for Windows Platforms, L_*filename* is the index log file.

For more information about alternate key indexes, see Alternate key indexes, on page 92.

Recoverable files

UniData provides the Recoverable File System (RFS) to protect your database in the event of hardware failures or system crashes. RFS includes before image logging, after image logging, archiving, and failure recovery.

The protection provided by the Recoverable File System works only on files you define as recoverable. You can create new files as recoverable with the `ECL CREATE.FILE` command, or convert existing nonrecoverable files to recoverable with the system-level `udfile` command.

Syntax

```
CREATE.FILE [DICT | DATA] [DIR | MULTIFILE | MULTIDIR]
filename [,subfile] [modulo [,block.size.multiplier]][TYPE hashtype]
[DYNAMIC [KEYONLY | KEYDATA] [PARTTBL part_tbl]]
[RECOVERABLE] [OVERFLOW]
```

The following example illustrates creating a recoverable file:

```
:CREATE.FILE TEST5 RECOVERABLE
modulos for file TEST5=59
Create file D_TEST5, modulo/1,blocksize/1024
Hash type = 0
Create file TEST5, modulo/59,blocksize/1024
Hash type = 0
```

To convert an existing static or dynamic hashed file from nonrecoverable to recoverable, use the system-level `udfile` command.

Note: A sequentially hashed file cannot be recoverable.

Syntax

udfile [-r |-s] *filename*

The UniData system-level `udfile` command converts a nonrecoverable file to a recoverable file, a recoverable file to a nonrecoverable file, or displays whether the file is recoverable or nonrecoverable. You must have root or Administrator privileges to change a file type using this command. If you do not specify an option, UniData returns the type of file (recoverable or nonrecoverable.) You do not need root or Administrator privileges if you only want to display the file type.

The following example illustrates converting a nonrecoverable file to a recoverable file.

```
# udfile -r INVENTORY
Non-recoverable file 'INVENTORY' is changed to recoverable file.
#
```

For more information about the `udfile` command, see *Administering the Recoverable File System*.

Chapter 4: The UniData VOC file

The VOC (vocabulary) file serves as the central repository for information about a UniData account. The VOC file contains direct information, such as commands, paragraphs, and keywords you can use, as well as pointers to menus, data and dictionary files, and cataloged programs.

Each time a user starts a UniData session, the first file UniData reads is the VOC file for that account. Each UniData account must contain a VOC file and its dictionary, D_VOC.

This chapter explains the elements of the VOC file, how to display VOC records, how to edit existing VOC records, and how to enter new VOC records.

VOC file record types

Each record in the VOC file has a record type. The record type is always located in attribute 1 of the VOC record. The following table lists valid record types. The record types are discussed in detail later in this chapter.

Type code	Description
C	**Locally or Directly Cataloged Program** - A UniBasic program that has been cataloged so that it can only be run in the account where it resides. UniData allows you to catalog programs to use as subroutines or commands by entering the name of the VOC record at the ECL prompt. Globally cataloged programs do not have a record in the VOC file; locally and directly cataloged programs do.
DIR	**Directory File Pointer** — Pointer to the path of a directory file. The VOC record may contain the full path to the file, or just the directory file name.
F	**File Pointer** — Pointer to the path of a UniData hashed file. The VOC record may contain the full path to the file, or just the file name.
FX	**External File Pointer** — A file pointer used by the Open File System (OFS) and Network File Access (NFA) to access external files (files on other machines).
K	**Keyword** — Specially defined words which are used to qualify UniData commands. Examples of keywords are BY, WITH, and so forth.
LD	**Multilevel Directory File** — Pointer to the path of a multilevel directory file. The VOC record may contain the full path to the file, or just the file name.
LF	**Multilevel File** — Pointer to the path of a multilevel file. The VOC record may contain the full path to the file, or just the file name.
M	**Menu** — Pointer to the path of a UniData menu, identifies the file and record in which the menu resides.
PA	**Paragraph** — Identifies the VOC record as a paragraph, and contains one or more UniQuery sentences that are executed as a group.
PQ	**Proc** — Identifies the VOC record as a PQ Proc, containing one or more UniQuery sentences that are executed as a group.
PQN	**Proc** — Identifies the VOC record as a PQN Proc, containing one or more UniQuery sentences that are executed as a group.
R	**Remote** — Pointer indicating the location of an item that exists in another file.
S	**Sentence** — A single UniQuery statement.
U	**Synonym** — Synonym for an existing VOC record.
V	**Verb** — A command that can be used to produce reports based on the dictionary records defined for a particular file.

Type code	Description
X	**User-defined** — Any information that you want to store in the VOC file for information purposes. X-type VOC records are ignored by UniData.

C-type — locally or directly cataloged programs

Locally cataloged programs

UniData creates a C-type VOC record for locally cataloged programs whenever you catalog a program with the LOCAL keyword. A locally cataloged program resides in the CTLG file of the account where you compile the program, and is available only in that account. If you recompile the program, you must also recatalog it. You execute the program by entering the VOC record name (typically the program name) at the UniData ECL prompt.

If a program is both locally and globally cataloged, the locally cataloged version takes precedence. Users typically catalog a program locally to test it before making it available to other users by globally cataloging it.

Record layout:

```
@ID:Item Name
Attribute 1:C [description]
Attribute 2:local catalog path
Attribute 3:program_file_name source_code_program_name
```

The following example shows a VOC record for a locally cataloged program:

```
:AE VOC EXAMPLE
Top of "EXAMPLE" in "VOC", 3 lines, 40 characters.
*--: P
001: C
002: /users/claireg/CTLG/EXAMPLE
003: BP EXAMPLE
Bottom.
*--:
```

Directly cataloged items

UniData creates a C-type VOC record for directly cataloged programs whenever you catalog a program with the DIRECT keyword. A directly cataloged program resides in the program file of the account where you compile the program and is available only in that account. Recataloging is not necessary when you recompile the program. You run the program by entering the VOC record name (typically the program name) at the UniData ECL prompt.

Record Layout:

```
@ID:Item Name
Attribute 1:C [description]
Attribute 2:program_file_name/compiled_program_name
```

The following example shows a VOC record for a directly cataloged program:

```
:AE VOC EXAMPLE1
Top of "EXAMPLE1" in "VOC", 2 lines, 14 characters.
001: C
```

```
002: BP/_EXAMPLE1
Bottom.
*--:
```

For more information about cataloging UniBasic programs, see *Administering UniData on UNIX* or *Administering UniData on Windows Platforms* and *Developing UniBasic Applications*.

DIR type — directory

A DIR type VOC record defines a UniData DIR type file. A DIR type file is a system-level directory that contains text or data files. Each text or data file is a UniData record. A common DIR- type file is the BP file, a UniData directory file that stores UniBasic source files and compiled programs.

Record Layout:

```
@ID:record_ID
Attribute 1:DIR [description]
Attribute 2:datafile_path
Attribute 3:dictfile_path
```

The following example shows the VOC record for the BP file:

```
:AE VOC BP
Top of "BP" in "VOC", 3 lines, 11 characters.
*--: P
001: DIR
002: BP
003: D_BP
Bottom.
*--:
```

F type — file pointer

Each file in a UniData account must have a file pointer record in the VOC file. This record points to the location of the file in the host operating system environment. This includes remote files as well as locally maintained files. You create local files by using the `CREATE.FILE` command, and you create remote file pointers (for files in other accounts) by using the `SETFILE` command.

If the file exists in the local account, attribute 2 contains the name of the data file and attribute 3 contains the name of the dictionary file. If the file exists in another account, attributes 2 and 3 contain the full path of the data file and dictionary file, respectively.

Recordlayout:

```
@ID:record_ID
Attribute 1:F [description]
Attribute 2:datafile_path
Attribute 3:dictfile_path
```

The following example illustrates a VOC record for a file existing in the local account:

```
:AE VOC CLIENTS
Top of "CLIENTS" in "VOC", 3 lines, 19 characters.
*--: P
001: F
002: CLIENTS
```

```
003: D_CLIENTS
Bottom.
*--:
```

The next example shows an F-type VOC pointer record for a file existing in another account:

```
:AE VOC TEST_FILE
Top of "TEST_FILE" in "VOC", 3 lines, 53 characters.
*--: P
001: F
002: /usr/ud72/demo/INVENTORY
003: /usr/ud72/demo/D_INVENTORY
Bottom.
*--:
```

Note: F-type VOC records for files in other accounts are created by the `SETFILE` command, or may be manually entered with a text editor. For more information about the `SETFILE` command, see the *UniData Commands Reference*.

Creating a synonym for a file

You can create a synonym for a data file by creating a new VOC record with a different record ID than the original VOC record. Attributes 2 and 3 contain the same information in attributes 2 and 3 as the original VOC record for the file. The file can then be accessed by both the original VOC record ID and the synonym, as shown in the following example:

```
:AE VOC INVENTORY
Top of "INVENTORY" in "VOC", 3 lines, 23 characters.
001: F
002: INVENTORY
003: D_INVENTORY
Bottom.
*--: Q
Quit "INVENTORY" in file "VOC" unchanged.
:AE VOC NEW_INVENTORY
Top of "NEW_INVENTORY" in "VOC".
*--: P
001= F
002= INVENTORY
003= D_INVENTORY
*--: Q
Quit "NEW_INVENTORY" in file "VOC" unchanged.
:
:LIST INVENTORY DESC
LIST INVENTORY DESC 17:34:15 Dec 03 2011 1
INVENTORY. .........
15001 Modem
35000 Speaker
.
.
.

:LIST NEW_INVENTORY DESC
LIST NEW_INVENTORY DESC 17:34:49 Dec 03 2011 1
INVENTORY. .........

15001 Modem
35000 Speaker
.
```

.
.

Sharing dictionary files

Data files can share a common dictionary file. You accomplish this by defining the same dictionary file in attribute 3 of the VOC records for the data files. The following example illustrates two files, NOV_INVENTORY and DEC_INVENTORY, which share the same dictionary:

```
:AE VOC NOV_INVENTORY
Top of "NOV_INVENTORY" in "VOC", 3 lines, 27 characters.
*--: P
001: F
002: NOV_INVENTORY
003: D_INVENTORY
Bottom.
*--: Q
Quit "NOV_INVENTORY" in file "VOC" unchanged.
:AE VOC DEC_INVENTORY
Top of "DEC_INVENTORY" in "VOC", 3 lines, 27 characters.
*--: P
001: F
002: DEC_INVENTORY
003: D_INVENTORY
Bottom.
*--: Q
Quit "DEC_INVENTORY" in file "VOC" unchanged.
:
```

Tip: Multilevel files and multilevel directory files have a structure that allows a number of files to share a single dictionary. However, these types of files are less efficient than ordinary static, dynamic, or sequentially hashed files because more system resources are needed to read and update them. An alternative way to have data files share the same dictionary is to modify the VOC records for files that share a common dictionary, as shown in the previous example.

FX type — external file pointer

External file pointers are used by the UniData Open File System (OFS) to recognize files that are on other machines, and to access the files as if they were local.

family is the OFS family of entry points. This is a set of routines that maintain a certain protocol. The functions in the family are preceded with the prefix in attribute 2 of the FX VOC record, which may be up to three characters long. These functions must be linked to UniData. *domain* is information particular to a group of files using a family of entry points.

Record layout:

```
@ID:record_ID
Attribute 1:FX [description]
Attribute 2:family
Attribute 3:domain
```

The following example shows an FX-type VOC record:

```
:AE VOC DATAFILE
Top of "DATAFILE" in "VOC", 3 lines, 46 characters.
```

```
*--: P
001: FX
002: BAR
003: MACHINE HOST2, FILE DATAFILE, PORT 3344
Bottom.
*--:
```

For more information about OFS, see *Developing OFS/NFA Applications*.

M type — menu item

M or MENU records are pointers to stored menus used with the UniData menu processor. A menu is a numbered list of items from which you can select an option that corresponds to a task you want to perform. A menu can invoke other menus, a sentence, a paragraph, a program, or any process that can be executed from the UniData command line.

Menus are stored in MENU files. When you create a new account, UniData creates a file called MENUFILE, which contains a dictionary record for each of the attribute names used by the menu processor. You may use this standard file, or another file of your own choosing. UniData creates records in the MENUFILE using the MENUS utility.

For more information about MENU files, see Using UniData MENUS, on page 143.

Record layout:

```
@ID:record_ID
Attribute 1:M [description]
Attribute 2:menu_name
Attribute 3:menu_ID
```

The following example shows an M-type VOC record:

```
:AE VOC REPORT.MENU
Top of "REPORT.MENU" in "VOC", 3 lines, 18 characters.
*--: P
001: M
002: MENUFILE
003: REPORTS
Bottom.
*--:
```

PA type — paragraphs

Paragraphs are a series of UniData commands or sentences that are stored in a VOC record. The first attribute of a paragraph is always PA. Commands in a paragraph are executed sequentially, and each command is evaluated by UniData as if the command was entered at the command line. A command in a paragraph can be the name of another paragraph, a sentence, a menu, or a complete UniQuery or UniData statement.

When a paragraph completes, you return to the point from which the paragraph was invoked.

Record layout:

```
@ID:record_ID
Attribute 1:PA [description]
Attribute 2:Command 1
```

```
Attribute 3:Command 2
.
Attribute n:Last command
```

The following paragraph example selects records from the CLIENTS file in the demo database, saves the records meeting the selection criteria to a saved list, retrieves the saved list, and finally lists the clients' name and state. The paragraph is executed by entering the name of the paragraph (the VOC record ID) at the command line:

```
:AE VOC TEST.PA
Top of "TEST.PA" in "VOC", 5 lines, 112 characters.
*--: P
001: PA
002: SELECT CLIENTS WITH STATE = 'CO'
003: SAVE.LIST COLORADO.CLIENTS
004: GET.LIST COLORADO.CLIENTS
005: LIST CLIENTS NAME STATE
```

Formatting paragraphs

If a command in a paragraph is longer than one line, you can continue the command to successive lines by entering the backslash (\) as the last character of the line to be continued.

You may enter spaces at the beginning of paragraph lines to indent text. This can add structure to a paragraph and make it easier to read.

You may comment out a line of a paragraph by entering an asterisk (*) at the beginning of the line. Commented lines are ignored when the paragraph is executed. Comments can be useful when you are debugging a paragraph, and enable you to store informative data about the paragraph in the VOC record.

UniData LOGIN and LOGOUT paragraphs

UniData enables you to create special LOGIN and LOGOUT paragraphs. The LOGIN paragraph is executed every time you enter a UniData session, and the LOGOUT paragraph is executed when you exit a UniData session. A LOGIN paragraph typically contains settings for UDT.OPTIONS, initiating a menu process, or providing additional security.

For detailed information about creating paragraphs, see UniData paragraphs, on page 102.

R type — remote Item

A remote record in the VOC file points to a record in another file. Remote items are used to help keep your VOC file small, to allow several accounts to share a common process, or to provide additional security by executing a security subroutine.

You can substitute remote items for sentences, paragraphs, verbs, locally cataloged programs, or menus.

Record layout:

```
@ID:remote_record_ID
Attribute 1:R [description]
Attribute 2:file_name
Attribute 3:record_name
Attribute 4:[subroutine_name]
```

Attribute 2 in the VOC record for a remote item is the file name where the remote item exists. The file name must be an existing F-type record in the VOC file. Attribute 3 of the remote item (*record_name*) is the record name of the record to be executed in the remote file.

One common use of remote VOC records is to segregate paragraphs in their own file. In the following example, a remote VOC record is created to point to a paragraph residing in the PA.MASTER file:

```
:AE VOC TEST_REMOTE
Top of "TEST_REMOTE" in "VOC", 3 lines, 16 characters.
*--: P
001: R
002: PA.MASTER
003: TEST
Bottom.
*--:
:AE PA.MASTER TEST
Top of "TEST" in "PA.MASTER", 2 lines, 35 characters.
*--: P
001: PA
002: DISPLAY THIS IS THE DEMO ACCOUNT
```

The format of a remote record is the same as the format of the same type of record in the VOC file. For instance, if a remote record is a paragraph, the format of the remote paragraph is identical to the format of a VOC paragraph.

How to execute remote items

You can execute a remote item either by entering the name of the item at the UniData command line, or by executing it by using an `EXECUTE` statement in a UniBasic program (for example, `EXECUTE` *remote_item*).

When you execute a remote item, UniData reads the VOC file to find the location of the remote item and identifies the type of record. If the remote item is a V or U type, UniData treats it as a UniData command. If the item is a C type, UniData treats it as a locally or directly cataloged UniBasic program. Otherwise, regardless of the type, UniData treats the item as a paragraph.

Using remote items for security

You can supply a security routine to be executed by the remote item. Remote items enable you to name a cataloged subroutine in attribute 4 which is executed when a user invokes the remote VOC record. The subroutine must have one argument, and return a value of 1 (true) or 0 (false). When a user invokes a remote item that executes 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:

```
:AE VOC LIST
Top of "LIST" in "VOC", 4 lines, 26 characters.
*--: P
001: R
002: OTHER_VOC
003: LIST
004: SECTEST2
Bottom.
*--:
```

With this VOC record in place, when you execute the LIST command, UniData executes a security subroutine called SECTEST2. If that subroutine returns a value of 1, UniData executes the record 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 the following 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 0 (false) and a user executes the ECL LIST command:

```
:CATALOG BP SECTEST2
: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' 17:55:22 Dec 06 2011 1
VOC.......

ECLTYPE
CP
CT
SP.OPEN
listdict
LISTDICT
```

S type — sentences

A sentence is a VOC record consisting of a complete UniQuery statement, containing a command, file name, and attribute names you want to display. Sentences are commonly used for commands that are used frequently, or when a command is long and complex and is used more than once. You execute a sentence by entering the record ID at the ECL prompt.

Record layout:

```
@ID:record_ID
Attribute 1:S [description]
Attribute 2:sentence
```

Attribute 1 of the VOC record for a sentence must be S. Attribute 2 can contain only one line. If the statement cannot be written on a single line, use a paragraph (PA type) instead of a sentence.

In the following example, a sentence is defined in the VOC file to list the name and company of records in the CLIENTS file:

```
:AE VOC TEST_SENTENCE
Top of "TEST_SENTENCE" in "VOC", 2 lines, 36 characters.
*--: P
001: S
002: LIST CLIENTS BY LNAME NAME COMPANY
Bottom.
*--:
```

Predefined UniQuery sentences

Every UniData system comes with a number of predefined sentences. These sentences provide convenient shorthand for reviewing the contents of an existing UniData account. The following table lists the predefined sentences in UniData accounts.

VOC record name	Description
CP	Sends a copy of a designated record to the printer.
CT	Displays the contents of a designated record to the screen.
LISTD LD	Lists all DIR-type files.
LISTDICTL	Sorts and lists dictionary record attributes by type, location, and @ID, and sends the list to the printer.
LISTF LF	Lists all the files defined in the VOC, including F-type, DIR-type, LD-type, and LF-type files by type, by file name.
LISTFL	Lists all the files that reside in the local account defined in the VOC, including F-type, DIR-type, LD-type and LF-type files by type, by name.
LISTFR	Lists all files that reside in other accounts defined in the VOC, including F-type, DIR-type, LD-type and LF-type files by type, by name.
LISTH LH	Lists all hashed files, sorted by file name.
LISTM	Lists all menu records defined in the VOC file.
LISTO	Lists records in the VOC file which do not contain V, LF, LD, M, R, PA or S in attribute 1 (i.e., all others).
LISTPA	Lists all paragraphs (PA type records) in the VOC file.
LISTR	Lists all remote items (R type records) in the VOC file.
LISTS	Lists all sentences (S type records) in the VOC file.
LISTV	Lists all verbs (V type records) in the VOC file.

U type — synonym

UniData allows you to create a synonym for any VOC record except another synonym. This feature allows you to assign alias names for files, commands, and other VOC records. For example, a name and address file might be called VENDOR in an accounts payable system but be called CUSTOMER in an accounts receivable system.

Record layout:

```
@ID:record_ID
Attribute 1:U [description]
```

```
Attribute 2:command_name
```

Synonyms also can be used to translate UniData commands from English into other languages. To do so, enter the name of the command in attribute 2 of the VOC record.

The following example shows a U-type VOC record called GET, which is a synonym for the `GET.LIST` command:

```
:AE VOC GET
Top of "GET" in "VOC", 2 lines, 9 characters.
*--: P
001: U
002: GET.LIST
```

V type — verbs

A V-type VOC record defines a UniQuery or UniData command. A UniQuery command extracts information from data files, and a UniData, or ECL, command manages files and records. When a V-type record in the VOC file is invoked, UniData checks to see if a VOC record exists for the requested command. If a record exists, UniData executes the program associated with the verb record. If the VOC item does not exist, the command cannot be executed.

Record layout:

```
@ID:record_ID
Attribute 1:V [description]
Attribute 2:command_name
```

Note: Users logged in as root on UniData for UNIX or Administrator on UniData for Windows Platforms can execute all UniData commands in the master VOC file, whether or not the commands exist in the VOC file for a particular UniData account. This allows a system administrator to log in as root on UniData for UNIX or Administrator on UniData for Windows Platforms and add or remove commands from a VOC file in a specific account, as necessary.

The following example shows the VOC record for the UniQuery SELECT verb:

```
:AE VOC SELECT
Top of "SELECT" in "VOC", 2 lines, 8 characters.
*--: P
001: V
002: SELECT
Bottom.
*--:
```

The next example shows the VOC record for the UniData `RESIZE` command:

```
:AE VOC RESIZE
Top of "RESIZE" in "VOC", 2 lines, 8 characters.
*--: P
001: V
002: RESIZE
Bottom.
*--:
```

For more information about UniData ECL commands, see the *UniData Commands Reference*. For more information about UniQuery commands, see *Using UniQuery* and the *UniQuery Commands Reference*.

X type — user-defined

An X-type VOC record contains user defined information, and is ignored by UniData. You can use an X-type VOC record to store any kind of information. In the following example, an X-type VOC record stores the current revision number of a software application.

Record layout:

```
@ID:record_ID
Attribute 1:X [description]
Attribute 2:user-defined information
.
.
.
Attribute n:user-defined information
```

```
:AE VOC REV_NUMBER
Top of "REV_NUMBER" in "VOC", 2 lines, 5 characters.
*--: P
001: X
002: 5.0
Bottom.
*--:
```

To view the contents of an X-type VOC record, you may either edit the record, or use the CT command to display the record to your screen.

Creating a VOC record

UniData contains a default VOC file containing commands, keywords, and predefined UniData paragraphs and sentences. UniData also creates VOC records in several instances:

- When you create a new directory, UniData creates a DIR-type VOC record.
- When you create a new file, UniData creates an F-type VOC record.
- When you save a statement or group of statements from the command stack, UniData creates a sentence (S-type) or paragraph (PA-type) record, as appropriate.
- When you locally or directly catalog a UniBasic program, UniData creates a C-type VOC record.

Editing the VOC file

You can edit the VOC file with UniEntry or a text editor such as vi or AE. UniData also provides three functions you can execute at the UniData ECL prompt that enable you to delete, list, or recall sentences and paragraphs. These three commands each begin with a period, and are sometimes referred to as “dot commands.”

Note: The following three dot commands operate only on sentences and paragraphs. If you attempt to use one of these functions on a VOC record other than a sentence or paragraph, UniData displays an error message.

These dot commands, shown in the following table, are similar to the command stack functions discussed in The UniData command stack, on page 116 but these commands operate on the VOC file, not the command stack.

Function	Description
.D *name*	Deletes the sentence or paragraph *name* from the VOC file.
.L *name*	Lists the contents of the sentence or paragraph *name* in the VOC file.
.R *name*	Recalls the sentence or paragraph name to the first line in the command stack.

Deleting a VOC record

Use the `.D` command to delete a sentence (S-type) or paragraph (PA-type) record from the VOC file.

Syntax

.D *record_ID*

Warning: You are not prompted for confirmation when you use the .D command. Therefore, be sure that the record you want to delete is not required by UniData or your application in any way.

Listing a sentence or paragraph

To list a VOC sentence record or paragraph record, use the `.L` command.

Syntax

.L *record_ID*

The following example displays the VOC record for the LISTDICT paragraph:

```
:.L LISTDICT
paragraph name=LISTDICT
type+description=PA
IF <<C3,DISPLAY>> = ''THEN GO L100
IF <<C3,DISPLAY>> = 'LPTR' THEN GO L200
IF <<C3,DISPLAY>> # 'lptr' THEN GO L100
L200: SORT DICT <<I2,FILE NAME>> TYP LOC CONV MNAME FORMAT
 SM ASSOC BY TYP BY LOC BY @ID LPTR
GO DONE
L100: SORT DICT <<I2,FILE NAME>> TYP LOC CONV MNAME FORMAT
SM ASSOC BY TYP BY LOC BY @ID
DONE: *
:
```

Recalling a sentence or paragraph

Use the `.R` command to recall a sentence or paragraph to the first position in the UniData command stack. You can recall a sentence or paragraph containing up to 2720 characters.

Syntax

.R *record_ID*

In the following example, the .R command loads the LISTDICT paragraph into the command stack:

```
3 SORT INVENTORY BY PRICE PRICE
2 AE VOC LISTPTR
1 LIST CLIENTS NAME
:.R LISTDICT
LISTDICT loaded from VOC file.
```

The .R command loads each line of the VOC paragraph into the command stack, so that the last line of the paragraph appears in the first position of the command stack, as shown in the following example:

```
10 SORT INVENTORY BY PRICE PRICE
9 AE VOC LISTPTR
8 LIST CLIENTS NAME
7 IF <<C3,DISPLAY>> = ''THEN GO L100
6 IF <<C3,DISPLAY>> = 'LPTR' THEN GO L200
5 IF <<C3,DISPLAY>> # 'lptr' THEN GO L100
4 L200: SORT DICT <<I2,FILE NAME>> TYP LOC CONV MNAME FORMAT SM
ASSOC BY TYP BY LOC BY @ID LPTR
3 GO DONE
2 L100: SORT DICT <<I2,FILE NAME>> TYP LOC CONV MNAME FORMAT SM
ASSOC BY TYP BY LOC BY @ID
1 DONE: *
:
```

Blocking execution of statements

You can prohibit UniData from executing a command within a sentence or paragraph by adding a question mark (?) to the end of the command you do not want to execute. For example, if you have four commands in a VOC paragraph and you do not want UniData to execute the third command, you can add a question mark to the end of the third command line. The question mark has the same effect as commenting out the line.

In the following example, UniData does not execute the third command in the paragraph since it contains a question mark at the end of the command:

```
:AE VOC LIST_VOC
Top of "LIST_VOC" in "VOC", 5 lines, 63 characters.
001: PA
002: SELECT VOC
003: LIST VOC SAMPLE 10
004: SELECT VOC BY @ID?
005: LIST VOC F1
Bottom.
*--:
```

User exits

User exits are special functions available in UniBasic, UniQuery, and Procs that enable you to customize your converted applications in UniData. By using user exits, you decide how UniData handles terminal output, return values, input strings, and other application factors.

UniData comes with several standard user exits, or you create your own.

For more information about user exits, see User exits, on page 129.

Chapter 5: Creating virtual attributes

The purpose of this chapter is to introduce you to virtual attributes and to provide examples of how to use virtual attributes. When you complete this chapter, you should be able to:

- Understand when to use a virtual attribute
- Know how to use the virtual attribute tools
- Create virtual attribute formulas and subroutines
- Check the virtual attribute syntax

Overview of virtual attributes

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, literal data, or other information in the database.

Through a formula or a UniBasic subroutine, UniData can access data stored in other attributes, perform calculations, and return results to a virtual attribute. UniData does not store data within the virtual attribute itself.

The primary use for virtual attributes is to display data using either UniQuery or UniData SQL. The value a virtual attribute displays is generated by formulas that operate on data that exists in one or more files.

Components of virtual attributes

Attribute 2 of a virtual attribute is made up of components. These components may be used alone or combined by virtual attribute operators with other components. You may use the following components to create virtual attribute formulas:

- Attribute names
- Constants and literal strings
- @variables
- UniBasic functions
- UniQuery functions

For information about the format of a virtual attribute dictionary record, see UniData relational database concepts, on page 10.

Attribute names

In a virtual attribute, an attribute name can reference another attribute which exists in the same dictionary file. The attribute name can be a D-type dictionary record or another virtual attribute.

In the following example, the virtual attribute concatenates FNAME, a space, and LNAME; FNAME and LNAME are two D-type dictionary records that exist in the same dictionary:

```
:AE DICT CLIENTS NAME
Top of "NAME" in "DICT CLIENTS", 7 lines, 30 characters.
*--: P
001: V
```

```
002: FNAME:" ":LNAME
003:
004: Name
005: 30T
006: S
```

In the preceding example, FNAME and LNAME are attribute names. The space (“ ”) is a literal string.

Constants or literal strings

A constant, or literal string, is a value that does not change during the execution of the virtual field. All nonnumeric constants and all literal expressions must appear within single or double quotation marks (‘’ or “ ”) in a virtual attribute formula. Constants may contain characters, numbers, symbols or a combination of the three.

In the following example, the virtual attribute COUNT contains the numeric constant 1 in attribute 2. When you execute a query using this virtual attribute, UniData keeps a tally of the number of items included in each breakpoint.

```
:AE DICT CLIENTS COUNT
Top of "COUNT" in "DICT CLIENTS", 6 lines, 12 characters.
*--: P
001: V
002: 1
003:
004:
005: 6R
006: S
Bottom.
*--:
:LIST CLIENTS WITH STATE = 'CA' OR WITH STATE = 'CO' BY STATE
BREAK.ON STATE TOTAL COUNT
LIST CLIENTS WITH STATE = 'CA' OR WITH STATE = 'CO' BY STATE
BREAK.ON STATE TOTAL COUNT 15:44:10 Dec 19 2011 1
CLIENTS... State/Territory  .....

      9982 CA                    1
      9986 CA                    1
           ********************* ------
           CA                    2

     10047 CO                    1
      9987 CO                    1
           ********************* ------
           CO                    2

                                 ======
TOTAL                            4
4 records listed
```

Variables

A variable is a value that can change during the execution of a virtual field. Many virtual attribute formulas operate on some form of a variable. You can use a UniData-provided @variable or a derived value in a virtual attribute formula.

@variables

@ variables (“at” variables) are special internal variables whose values are changed when the virtual attribute formula is evaluated. There are two types of @variables available in a virtual attribute formula: @variables that represent system information, such as the system date, and those that store intermediate results during the execution of compound virtual attribute formulas.

For information about @variables in compound virtual attribute statements, see Compound virtual attribute statements, on page 88.

The following table lists the @variables available in the UniData system.

@variable	Description
@DATE	System date in display format.
@DAY	Day of month (1-31).
@FILE.NAME	The file name used in the UniQuery statement.
@FM	Field mark character.
@ID	Record ID of the current record.
@IM @RM	Record mark character.
@ITEM.COUNT	The number of records displayed in a UniQuery LIST statement.
@MONTH	Month of the year (1-12).
@*n*	The result of the *n*th expression in the virtual attribute formula, where *n* is any number beginning with 1.
@NS or @SUBVALUE.COUNT	Assigns a number to each subvalue in a multi-subvalued attribute in the manner of a running pointer. Numbering begins at 1 for each subvalue. Line 2 of the virtual attribute must include the multi-subvalue field you are counting followed by ‘@NS.’
@NV or @VALUE.COUNT	Assigns a number for each value in a multivalued attribute in the manner of a running pointer. Numbering begins at 1 for each multivalue. Line 2 of the virtual attribute must include the multivalued field you are counting, followed by ‘;NV’ as shown in the following example: 001: V 002: TAPES_RENTED;@NV 003: . 004: Tapes Cnt 005: 5R 006: MV
@RECORD	Entire data record.
@SM	Subvalue mark character: CHAR(252). The subvalue mark character may vary depending on your language group setting. For further information on language group settings, see *UniData International*.
@TIME	Time in display format. Set when command is executed.
@TM	Text mark character: CHAR(251). The text mark character may vary depending on your language group setting. For further information on language group settings, see *UniData International*.

@variable	Description
@USER.NO	The user number on the host system. This number is always the same for a particular use, regardless of the terminal number. Its value depends on the version of the host operating system.
@USERNAME	The user name on the host system.
@VM	Value mark character: CHAR(253). The value mark character may vary depending on your language group setting. For further information on language group settings, see *UniData International*.
@SYSTEM.RETURN.CODE	A number that indicates whether an error occurred.
@YEAR	Two-digit number representing the current year.

In the following example, the virtual attribute TODAY prints the current system date, using @DATE:

```
:AE DICT CLIENTS TODAY
Top of "TODAY" in "DICT CLIENTS", 6 lines, 19 characters.
*--: P
001: V
002: @DATE
003: D2/
004:
005: 10R
006: S
Bottom.
:LIST CLIENTS TODAY
LIST CLIENTS TODAY 15:56:30 Nov 19 1998 1
CLIENTS... .........
        9999    12/19/96
       10034    12/19/96
.
.
.
```

Virtual attribute tools

In a UniData virtual attribute, you can use functions to manipulate data; and 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. Tools available for use in virtual attributes include:

- **Arithmetic Operators**: Anything needed for mathematical operations, such as plus, minus, and so forth.
- **String Extraction Operators**: Phrases to extract all or part of a string of data.
- **Relational Operators**: Operators needed for comparative logic, such as greater than, less than, equal to, and so forth.
- **Boolean Operators and Conditional Expression**: AND, OR, and IF/THEN/ELSE constructions.
- **UniBasic Data, String, Manipulation and Conversion Functions**: Functions that enable you to manipulate numeric data (ABS, RND, SQRT, and so forth) as well as manipulate text (DOWNCASE, SPACE, STR, and so forth).
- **System Variables**: Called “at” (@) variables.

- **Functions**: TRANS and TOTAL functions.
- **Subroutines**: Calls for UniBasic subroutines.

In a virtual attribute, you can apply the preceding tools to attribute names, numeric constants, and literal strings. You can make several function calls in one virtual attribute by separating each call with a semicolon (;).

For more information about UniBasic functions, see *Developing UniBasic Applications*.

UniBasic functions in virtual attributes

Many UniBasic functions are available for use in virtual attribute dictionary record definitions.

For detailed information about UniBasic functions, see the *UniBasic Commands Reference*.

The following table describes the functions that are available for use in virtual attributes:

Function	Multivalued equivalent	Description
ABS(*x*)	NA	Returns the absolute value of *x*.
ACOS(*expr*)	NA	Returns the trigonometric arc cosign of *expr*.
ASCII(*expr*)	NA	Converts *expr* from EBCDIC to ASCII.
ASIN(*expr*)	NA	Returns the trigonometric arc sine of *expr*.
ATAN(*expr*)	NA	Returns the trigonometric arc tangent of *expr*.
CHAR(*expr*)	SUBR('-CHARS',*expr*)	Converts *expr* to its ASCII value.
COL1()	NA	Returns column position before a substring found by FIELD.
COL2()	NA	Returns column position after a substring found by FIELD.
COS(*expr*)	NA	Returns the trigonometric cosine of *expr*.
DATE()	NA	Returns the current system date in internal format.
DCOUNT(*attr*,*delim*)	NA	Returns the number of substrings delimited by *delim* in a string in *attr*.
DOWNCASE(*attr*)	NA	Converts characters in *attr* to lowercase.
EBCDIC(*expr*)	NA	Converts ASCII value *expr* to EBCDIC.
EXP(*expr*)	NA	Raises *e* to the power of *expr*.
EXTRACT(*attr, attrib.expr, val.expr,subval.expr*)	NA	Returns an attribute, value, or subvalue of *attr*.
FIELD(*attr*,*delim*, *field*[,*occurrences*])	SUBR('-FIELDS',*attr*, *delim*,*field*[,*occurrences*])	Returns the number of substrings specified by *occurrences* of *delim* in *attr*, beginning at *field*.
FMT(*expr*,*code*)	SUBR('-FMTS',*expr*,*code*)	Formats *expr* based on *code*.
ICONV(*expr*,*conv.code*)	SUBR('-ICONVS',*expr*, *conv.code*)	Converts *expr* to internal format.
INDEX(*attr*,*str.expr*, *num.expr*)	SUBR('-INDEXS',*attr,str.exp, num.expr*)	Returns the starting position of *num.expr* occurrence of *str.expr* in *attr*.
INT(*num.expr*)	NA	Returns the integer value of *num.expr*.

Function	Multivalued equivalent	Description
ISNV(*attr_name*)	ISNVS(*attr_name*)	Returns 1 if *attr* is a null value; returns 0 if *attr* is not a null value.
LEN(*str.expr*)	SUBR('-LENS',*str.expr*)	Returns length of *str.expr*.
LN(*num.expr*)	NA	Returns natural log of *x*.
LOWER(*attr*)	NA	Lowers UniData file delimiters to the next lower level delimiter.
MOD(*num.expr1*, *num.expr2*)	NA	Returns the modulus (remainder) of dividing *num.expr2* into *num.expr1*.
NOT(*expr*)	SUBR('-NOTS',*expr*)	If *expr* is 0 or an empty string, returns 1; if *expr* is not 0 or an empty string, returns 0.
NUM(*expr*)	SUBR('NUMS',*expr*)	If *expr* is numeric or an empty string, returns 1; if *expr* is not numeric or an empty string, returns 0.
OCONV(*expr*, *conv.code*)	SUBR('-OCONVS',*expr*, *conv.code*)	Converts *expr* to external format based on *conv.code*.
PWR(*num.expr1*, *num.expr2*)	NA	Raises *num.expr1* to the *num.expr2* power.
RAISE(*attr*)	NA	Raises UniData file delimiters by one value in *attr*.
RND(*num.expr*)	NA	Returns a random number less than *num.expr*.
SEQ(*str.expr*)	SUBR('-SEQS',*str.expr*)	Converts ASCII value to number.
SIN(*num.expr*)	NA	Returns the SIN of *num.expr*.
SOUNDEX(*expr*)	NA	Converts *expr* into a phonetic code.
SPACE(*num.expr*)	SUBR('-SPACES', *num.expr*)	Returns *num.expr* number of spaces.
SQRT(*num.expr*)	NA	Returns square root of *num.expr*.
STR(*str.expr*,*num.expr*)	SUBR('-STRS',*str.expr*, *num.expr*)	Returns *num.expr* number of *str.expr*.
TAN(*num.expr*)	NA	Returns the tangent of *num.expr*.
TIME()	NA	Returns internal system time.
TIMEDATE()	NA	Returns external date and time.
TRIMB(*attr*)	NA	Trims trailing blanks from *attr*.
TRIMF(*attr*)	NA	Trims leading blanks from *attr*.
UPCASE(*attr*)	NA	Converts characters in *attr* to uppercase.

In the following example, a virtual attribute uses the LEN function to return the length of LNAME in the CLIENTS file:

```
:AE DICT CLIENTS LNAME_LENGTH
Top of "LNAME_LENGTH" in "DICT CLIENTS", 6 lines, 21 characters.
*--: P
001: V
002: LEN(LNAME)
003:
004:
005: 5R
006: S
Bottom.
*--:
```

```
:LIST CLIENTS LNAME LNAME_LENGTH
LIST CLIENTS LNAME LNAME_LENGTH 11:43:42 Jan 07 2011 1
CLIENTS... Last Name...... ....
        9999 Castiglione 11
      10034 Anderson 8
        9980 Ostrovich 9
.
.
.
```

Virtual attribute functions

UniData also provides functions that are available only in virtual attributes. Although some of the functions in the following table are similar to UniBasic functions, their syntax varies from that of the similar UniBasic functions. These functions are described in the following table.

Function	Multivalue equivalent	Description
ADD_MONTHS(*date*, *nbr*)	NA	Adds the number of months specified by *nbr* to a date attribute (*date*). If *nbr* is negative, subtracts the number of months.
CONVERT(“*char1*”, “*char2*”,*attr*)	NA	Converts character *char1* to *char2* in *attr*.
COUNT(*attr*,*expr*)	SUBR(’-COUNTS’,*attr*, *expr*)	Returns the number of *expr* in *attr*.
INITCAP(*x*)	NA	Capitalizes the first letter of *x*.
INSTR(*str,substr, s.pos,occurrence*)	NA	Returns the starting position of the specified *occurrence* of *substr* in *str*, starting at *s.pos*.
LAST_DAY(*date*)	NA	Returns the last day of the month of a date attribute.
LENGTH(*x*)	NA	Returns the length of *x*.
LPAD(*string,number,character*)	NA	Pads *number* of specified *character* to the left of *string*.
MATCHFIELD(*a*,*b*,*1*)	NA	Returns a portion of a string that matches substring *b* in *a*.
MONTHS_BETWEEN (*date1*,*date2*)	NA	Returns the number of months between *date1* and *date2*.
NEXT_DAY (*date*,*dayname*)	NA	Returns the date of the first *day of the week* following the specified *date*.
N(*expr*)	NA	Permits literals that are also keywords (for example, DATE, TIME, TRANS) to refer to dictionary items rather than be interpreted as keywords.
POWER(*nvalue*,*e*)	NA	Raises *nvalue* to the power *e*.
REUSE(*str.expr*)	NA	Reuses *str.expr* in successive operations.
ROUND(*nvalue*,*e*)	NA	Rounds *nvalue* to the number of decimal places specified by *e*.
RPAD(*string,number,character)*	NA	Pads *number* of specified *character* to the right of *string*.

Function	Multivalue equivalent	Description
SIGN(*nvalue*)	NA	Returns +1 if *nvalue* is greater than 0, 0 if *nvalue* is equal to 0, -1 if *nvalue* is less than 0.
SUBSTR or SUBSTRING(*attr*, *num.expr)*	NA	Returns substring of *attr* beginning at *num.expr* character
SUM(*attr*)	NA	Sums the values of *attr*.
TOTAL()	NA	Accumulates running totals on numeric fields.
TRIM(*attr*)	NA	Trims extraneous spaces from *attr*.
TRUNC(*n,* {d})	NA	Truncates *nvalue* to *d* spaces to the right of the decimal place. Default is 0.
UPPER(*attr*)	NA	Converts characters in *attr* to uppercase.
USER_ID	NA	Returns the user's uid.
USER_NAME	NA	Returns the user's user name.

Special virtual attribute functions

UniData provides three functions that are specifically designed to handle certain kinds of virtual attribute operations:

- TRANS — extracts attributes or records from other data files
- SUBR — calls UniBasic subroutines
- TOTAL — maintains intermediate totals of other expressions

TRANS function

UniQuery statements operate on a single file, sometimes referred to as the working file. A virtual attribute may be created in the dictionary of the working file to access data which resides in another file by using the TRANS function.

Syntax

TRANS([DICT] *targ_file*,*attr_ID*, *targ_attr*,*"ret_code"*)

The following table lists the parameters for the TRANS function.

Parameter	Description
targ_file	The name of the file that contains the data you want to access. This can be the current working file or another file, and it must have an entry in the VOC file. *targ_file* must be the first element in the TRANS function expression (with the exception of DICT). If a dictionary item exists in the current working directory with the same name as the *targ_file*, you must enclose *targ_file* in quotation marks.
attr_ID	An attribute or expression that evaluates to the record ID in the *targ_file*. If *attr_ID* is an expression, enclose it in quotation marks.
targ_attr	The name or number of the attribute in the *targ_file* you want to return. If you use the attribute name, it must be a D-type attribute. If the attribute name exists in both the working file and the *targ_file*, you must enclose it in quotation marks. You may also use -1 to return the entire record.

Parameter	Description
ret_code	Tells the TRANS function what to do if the operation is unsuccessful. If followed by a number, specifies which value in a multivalued attribute to return. The *ret_code* must be enclosed in quotation marks.

The following tables describes the available return codes.

ret_code	Description
C	If the record or the attribute does not exist, or the value is an empty string or null, return the *attr_ID* instead of the value of the function.
V	If the record or the attribute does not exist, or the value is an empty string or the null value, return an empty string and print an error message.
X	If the record or the attribute does not exist, or the value is an empty string, return an empty string.
[*n*]	If the *targ_attr* is multivalued, return the *n*th value. If *n* does not exist, return all values.

Note: If both*attr_ID* and *targ_attr* are multivalued, the TRANS function returns a separate set of subvalues from *targ_attr* for each value of the *attr_ID*. In this case, the value type for the virtual attribute in the working file should be MS.

The following example of a virtual attribute in the ORDERS file uses the TRANS function to return the COMPANY attribute from the CLIENTS file:

```
:AE DICT ORDERS COMPANY
Top of "COMPANY" in "DICT ORDERS", 7 lines, 58 characters.
*--: P
001: V
002: TRANS(CLIENTS,CLIENT_NO,COMPANY,'X')
003:
004: Company
005: 15T
006: S
007:
Bottom.
*--:
```

CLIENT_NO is an attribute in the ORDERS file and is also the @ID of the records in the CLIENTS file. This virtual attribute accesses the CLIENTS file, finds the record defined by CLIENT_NO, and returns the COMPANY, as shown in the following example:

```
:LIST ORDERS COMPANY
LIST ORDERS COMPANY 14:59:14 Jan 17 2011 1
ORDERS.... Company........

  912 Boulder
    Productions
  801 Meier Shipping
  941 York Software
.
.
.
```

Tip: If you need to execute multiple TRANS functions against the same file, consider using the -1 option to return the entire record, then the EXTRACT function to retrieve the desired attributes for greater efficiency. For more information, see Compound virtual attribute statements, on page 88.

SUBR function

The SUBR function to calls a UniBasic subroutine from a virtual attribute. The UniBasic subroutine can be one you have written, or one of the UniData-supplied subroutines.

The subroutine must be compiled and cataloged before being called from a virtual attribute. See *Developing UniBasic Applications* for information about compiling and cataloging UniBasic subroutines.

Attribute 2 of a virtual attribute calls a UniBasic subroutine using the following syntax:

```
SUBR("subroutine",argument1[,argumentN...])
```

where *subroutine* is the name of the UniBasic subroutine you are calling, and *argument* is the value you are passing to the subroutine. The SUBR function call must have the same number of arguments as the subroutine you are calling. The called subroutine returns one argument, which may be multivalued or multi-subvalued.

In the following example, the ZIP virtual attribute in the dictionary of the CLIENTS file calls the PSTLCODE_FMT subroutine:

```
:AE DICT CLIENTS ZIP
Top of "ZIP" in "DICT CLIENTS", 7 lines, 50 characters.
*--: P
001: V
002: SUBR("PSTLCODE_FMT",@RECORD)
003:
004: Postal Code
005: 10R
006: S
Bottom.
*--:
```

The following program segment shows a portion of the PSTLCODE_FMT subroutine:

```
:AE BP_SOURCE PSTLCODE_FMT
Top of "PSTLCODE_FMT" in "BP_SOURCE", 47 lines, 1,166 characters.
*--: p
001:
*****************************************************************
002: * Postal Code Format ||| DATE: 07/06/95 ||| Arden C. Harrell
*
003:
*****************************************************************
004:
005: SUBROUTINE PSTLCODE_FMT(RET_DATA,ENTIRE_RECORD)
006:
007: IF ENTIRE_RECORD = "" THEN
008:  PRINT "@RECORD is null; aborting"
009:  RETURN
010: END
011:
012:
013: POSTALCODE = ENTIRE_RECORD<7>
014: COUNTRY = ENTIRE_RECORD<8>
015:
016: COUNTRY = UPCASE(COUNTRY)
017:
018: BEGIN CASE
019:   CASE COUNTRY = 'USA'
020:     IF LEN(POSTALCODE) = 5 THEN
021:       RET_DATA = FMT(POSTALCODE,"10R     #####")
022:     END ELSE
023:       RET_DATA = FMT(POSTALCODE,"10R#####-####")
024:     END
025:   CASE COUNTRY = 'CANADA'
026:     RET_DATA = FMT(POSTALCODE,"7R### ###")
027:   CASE COUNTRY = 'FRANCE'
028:     BEGIN CASE
029:       CASE LEN(POSTALCODE) = 4
030:         RET_DATA = FMT(POSTALCODE,"6R  ####")
031:       CASE LEN(POSTALCODE) = 5
032:         IF UPCASE(TRIM(POSTALCODE[1,1])) = "F" THEN
033:           RET_DATA = FMT(POSTALCODE,"6R#-#####")
034:         END ELSE
035:           RET_DATA = FMT(POSTALCODE,"5R#####")
036:         END
037:       CASE 1
038:         RET_DATA = POSTALCODE
039:     END CASE
040:   CASE COUNTRY = 'AUSTRALIA'
041:     RET_DATA = FMT(POSTALCODE,"4R####")
042:   CASE 1
043:     RET_DATA = POSTALCODE
044: END CASE
045:
046: RETURN
047: END
Bottom.
```

In the previous example, RET_DATA is the value that the subroutine returns to the virtual attribute, and ENTIRE_RECORD defines the argument that is passed from the virtual attribute to the subroutine.

For more information about writing UniBasic subroutines, see *Developing UniBasic Applications*.

Nested subroutines for multivalued attributes

UniData provides subroutines for use with multivalued attributes.

The subroutines are defined in UniBasic functions in virtual attributes, on page 75 and Virtual attribute functions, on page 77. In some cases, you must nest functions in a virtual attribute to achieve the results you desire.

You must delimit the separate formulas of nested functions with parentheses. When UniData encounters a virtual attribute formula containing nested functions, UniData evaluates from the innermost formula to the outermost.

When performing comparisons of a single value to a multivalued or multi-subvalued attribute, you need to use the REUSE function so that the comparison can be made for all the values in the multivalued attribute. If you do not use the REUSE function, the comparison is made only against the first value.

In the following example, the virtual attribute COLOR_TYPE created in the ORDERS file nests three functions to display the attributes whose COLOR is equal to Black. If the COLOR matches Black, UniData displays Black. If the COLOR is not Black, UniData displays Not Black.

```
:AE DICT ORDERS COLOR_TYPE
Top of "COLOR_TYPE" in "DICT ORDERS", 6 lines, 79 characters.
*--: P
001: V
002: SUBR("-IFS",SUBR("-EQS",COLOR,REUSE("Black")),"Black","Not Black")
003:
004:
005: 15L
006: MV
Bottom.
*--:
```

This virtual attribute uses the IFS subroutine to determine if COLOR is equal to Black. Since COLOR is a multi-subvalued attribute, you must use this subroutine rather than the IF/THEN/ELSE statement used with singlevalued attributes.

The EQS subroutine is nested within the IFS subroutine, telling UniData to evaluate the COLOR attribute to test for Black. Since you want to evaluate each value, you need to use the REUSE function to tell UniData to test for Black against each value.

The final two arguments of the IFS subroutine are Black and Not Black. If the result of the EQS subroutine is true, then Black becomes the value of the virtual attribute, otherwise the value is Not Black.

TOTAL function

The TOTAL function is used in a virtual attribute to accumulate subtotals for numeric attributes. These subtotals can then be used to calculate totals and averages, and to determine percentages using the CALC (or CALCULATE) keyword. The calculated values are printed on breakpoint lines and the final line of the report.

Note: The TOTAL function is not a UniBasic function, and differs from the UniQuery TOTAL keyword.

Use of the TOTAL function is a two-step process. First, create a virtual attribute using the TOTAL function for each attribute for which you want to accumulate subtotals. In the following example, the INV_COST virtual attribute created in the dictionary of the INVENTORY file multiplies PRICE by QTY when PRICE is greater than 0:

```
:AE DICT INVENTORY INV_COST
Top of "INV_COST" in "DICT INVENTORY", 6 lines, 78 characters.
*--: P
```

```
001: V
002: IF TOTAL(PRICE) > 0 THEN TOTAL(PRICE)*TOTAL(QTY) ELSE TOTAL("")
003: MD2,
004:
005: 13R
006: S
Bottom.
*--:
```

The next step in using the TOTAL function is creating a query statement using the CALC keyword to display the totals. In the following example, records from the INVENTORY file are displayed with a breakpoint on PROD_NAME, totals are displayed for the QTY and PRICE attributes, and totals for INV_COST are calculated both at the breakpoints and as a grand total:

```
:LIST INVENTORY BY PROD_NAME BREAK.ON PROD_NAME TOTAL QTY TOTAL
PRICE CALC INV_COST
LIST INVENTORY BY PROD_NAME BREAK.ON PROD_NAME TOTAL QTY TOTAL
PRICE CALC INV_COST 11:06:54 Jan 23 2011 1
            Product
INVENTORY. Name..... Quantity Price.....  .............

14001      Memory         6131     $49.95    306,243.45
           ********** -------- ---------- -------------
           Memory         6131     $49.95    306,243.45

15001      Modem          7486    $119.00    890,834.00
15002      Modem          3988    $199.99    797,560.12
15003      Modem          4913    $259.99  1,277,330.87
15004      Modem           146    $219.99     32,118.54
           ********** -------- ---------- -------------
           Modem        16533    $798.97 13,209,371.01

                       ======== ========== =============
                        22664    $848.92 19,239,922.88
5 records listed
```

If you use the TOTAL keyword with INV_COST rather than CALC, the total of the breakpoint for the INV_COST column is displayed at the breakpoint, rather than the calculated total of QTY * PRICE:

```
>LIST INVENTORY BY PROD_NAME BREAK.ON PROD_NAME TOTAL QTY TOTAL
PRICE TOTAL INV_COST
LIST INVENTORY BY PROD_NAME BREAK.ON PROD_NAME TOTAL QTY TOTAL
PRICE TOTAL INV_COST 11:18:00 Jan 23 2011 1
            Product
INVENTORY. Name..... Quantity Price.....  .............

14001      Memory         6131     $49.95    306,243.45
           ********** -------- ---------- -------------
           Memory         6131     $49.95    306,243.45

15001      Modem          7486    $119.00    890,834.00
15002      Modem          3988    $199.99    797,560.12
15003      Modem          4913    $259.99  1,277,330.87
15004      Modem           146    $219.99     32,118.54
           ********** -------- ---------- -------------
           Modem        16533    $798.97  2,997,843.53

                       ======== ========== =============
TOTAL                   22664    $848.92  3,304,086.98
5 records listed
```

Arithmetic operators

Arithmetic operators perform mathematical operations against elements in a virtual attribute formula to derive a value for the virtual attribute.

The following table lists the available arithmetic operators.

Operator	Description
+ and -	Unary plus and minus
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
^	Exponentiation

The dictionary of the ORDERS file in the demo database contains a virtual attribute called EPRICE. This virtual attribute multiplies PRICE by QTY; multiplication is indicated by the asterisk (*) on line 2.

```
:AE DICT ORDERS EPRICE
Top of "EPRICE" in "DICT ORDERS", 7 lines, 50 characters.
*--: P
001: V
002: PRICE*QTY
003: MD2,$
004: Extended Price
005: 12R
006: MS
007: LINE_ITEMS
Bottom.
*--:
```

Priority of arithmetic operators

UniData evaluates arithmetic expressions in a virtual attribute formula in the following priority:

1. Exponentiation
2. Multiplication and division
3. Addition and subtraction

To change the priority, use parentheses to group operations in the virtual attribute formula.

String operators

UniData provides two types of string operators: concatenation operators and extraction operators. The concatenation operators join (concatenate) two strings, while the string extraction operators return a string based on two or more variables.

Concatenation operators

The following table describes the UniData concatenation operators for singlevalued and multivalued attributes.

Operator	Multivalue equivalent	Description
string1:*string2*	SUBR('-CATS',*string1*,*string2)*	Concatenates (joins) *string2* to the end of *string1*.
string1 CAT *string2*	SUBR('-CATS',*string1*,*string2*)	Concatenates (joins) *string2* to the end of *string1*.
x:*string*:*y*	SUBR('-SPLICE',*x*,*string*,*y*)	Splices (inserts) *string* between values *x* and *y* in *string*.

The dictionary of the CLIENTS file in the demo database contains a virtual attribute called NAME which concatenates FNAME with a space, then LNAME:

```
:AE DICT CLIENTS NAME
Top of "NAME" in "DICT CLIENTS", 7 lines, 30 characters.
*--: P
001: V
002: FNAME:" ":LNAME
003:
004: Name
005: 30T
006: S
007:
Bottom.
*--:
```

String extraction operators

String extraction operators return a substring within a specified string based on the starting position and the number of characters to extract provided in the operator.

The following table describes the string extraction operators for singlevalued and multivalued attributes.

Operator	Multivalued equivalent	Description
string[*x*,*y*]	SUBR('-SUBSTRINGS,*string*,*x*,*y*)	Starting from the *x*th position in string, extract *y* characters.
string[*x*,*y*,*z*]	SUBR('-CATS',*string1*,*string2*	Marks the number of *x* characters (for example, spaces or commas, and so forth), and starting with *y*th occurrence of that character, extracts *z* number of subsequent attributes of characters.

The following virtual attribute can be created in the CLIENTS file to return the first four characters of LNAME:

```
:AE DICT CLIENTS PARTIAL_NAME
Top of "PARTIAL_NAME" in "DICT CLIENTS", 6 lines, 21 characters.
*--: P
001: V
002: LNAME[1,4]
```

```
003:
004:
005: 4L
006: S
Bottom.
*--:
```

The virtual attribute PARTIAL_NAME returns the following results:

```
:LIST CLIENTS LNAME PARTIAL_NAME
LIST CLIENTS LNAME PARTIAL_NAME 16:09:58 Jan 23 2011 1
CLIENTS... Last Name..... ...

9999 Castiglione Cast
10034 Anderson Ande
9980 Ostrovich Ostr
10015 di Grigorio di G
10072 Sears Sear
10053 Lee Lee
.
.
.
```

Relational operators

Relational operators compare values with variables or constants. Numeric variables or constants are compared arithmetically. String variables 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. Since lowercase letters appear later in the sequence of ASCII codes than uppercase letters, UniData evaluates lowercase letters as greater than uppercase letters.

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

- The null value — the null value is always the smallest value when values are sorted. Arithmetic operations that involve the null value always result in the null value. For more information, see Null value handling, on page 126.
- ECLTYPE — controls whether UniData interprets a query based on the UniData parser or the Pick® parser.
- SORT.TYPE — governs the sorting algorithm that UniData uses.
- COLLATIONS file — defines the default sorting order for UniQuery reports.
- The setting of UDT.OPTION 69 — determines how alphanumeric data is sorted when the dictionary item specifies a right-justified sort.

> **Tip:** For more information about ECLTYPE and SORT.TYPE, see*Using UniQuery*. For more information about UDT.OPTIONS, see the *UDT.OPTIONS Commands Reference*.

The following table describes valid relational operators.

Operator	Multivalued Equivalent	Description
EQ or =	SUBR('-EQS',*x,y*)	Equal to

Operator	Multivalued Equivalent	Description
NE or # or <> or ><	SUBR('-NES',*x,y*)	Not equal to
GT or >	SUBR('-GTS',*x,y*)	Greater than
LT or <	SUBR('-LTS',*x,y*)	Less than
GE or >= or =>	SUBR('-GES',*x,y*)	Greater than or equal to
LE or <= or =<	SUBR('-LES',*x,y*)	Less than or equal to
#>	NA	Not greater than
#<	NA	Not less than
MATCH or MATCHES	NA	Value matches pattern — 0N, 0A, or literal. See the *UniBasic Commands Reference* for more information.

Boolean operators

Boolean operators combine or compare sets of relational data, and produce a result of true or false. Typically, Boolean operators are used in conditional expressions (IF/THEN/ELSE statements).

The following table describes the Boolean operators.

Operator	Multivalued equivalent	Description
expr1 AND *expr2*	SUBR('-ANDS',*expr1,expr2*)	Expressions linked by AND must both be true. There must be two expressions.
expr1 OR *expr2*	SUBR('-ORS',*x,y*)	Either expression linked by OR can be true. There must be two expressions.
NOT(*expr1 OPERATOR expr2*)	SUBR('-NOTS',*expr1 OPERATOR expr2*)	If *expr* is 0 or an empty string, returns 1; if *expr* is not 0 or is an empty string, returns 0. There must be only one operand, enclosed by parentheses.

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. SeeNull value handling, on page 126 for more information on selecting and comparing the null value.

Conditional expressions

A conditional expression causes something to happen if a statement is true, and something else to happen if a statement is false. A conditional expression must contain an IF clause, a THEN clause and an ELSE clause.

The following table shows the syntax for a conditional expression.

Operator	Multivalued equivalent	Description
IF *expr1* THEN*expr2* ELSE *expr3*	SUBR('-IFS',*expr1,expr2,expr3*)	If *expr1* is true (returns the value 1), then perform the statements in *expr2*, otherwise perform those in *expr3*.

In the following example, the virtual attribute SALE, created in the INVENTORY file, evaluates PROD_NAME. If the PROD_NAME equals Modem, On Sale is returned. If the PROD_NAME is not equal to Modem, Regular Price is returned.

```
:AE DICT INVENTORY SALE
Top of "SALE" in "DICT INVENTORY", 7 lines, 69 characters.
001: V
002: IF PROD_NAME = 'Modem' THEN 'On Sale' ELSE 'Regular Price'
003:
004:
005: 25L
006: S
Bottom.
*--:
```

The following example shows the output from a UniQuery statement using the SALE virtual attribute:

```
:LIST INVENTORY PROD_NAME SALE
15003       Modem       On Sale
15004       Modem       On Sale
14001       Memory      Regular Price
...
```

Compound virtual attribute statements

Compound statements consist of multiple expressions separated by semicolons. A compound statement is evaluated from left to right. The value for the entire compound statement is the value of the last expression in the statement.

Syntax

expression1;*expression2*[;*expression3*]...

Expressions in a compound statement are numbered sequentially from left to right, beginning with @1 (there is no @0). You can reference an expression in a compound statement with the following @variables:

- @ — Value of the expression immediately preceding the current expression
- @*n* — Value of a specific expression in a compound statement

The following virtual attribute containing a compound statement can be created in the ORDERS file to return the FNAME, LNAME, COMPANY, CITY, STATE, and ZIP from the CLIENTS file:

```
:AE DICT ORDERS CLIENT_INFO
Top of "CLIENT_INFO" in "DICT ORDERS", 6 lines, 191 characters.
*--: P
001: V
002:TRANS(CLIENTS,CLIENT_NO,-1,'X');EXTRACT(@1,1,1,1);EXTRACT(@1,2,1,1);
EXTRACT(@1,3,1,1);EXTRACT(@1,5,1,1);EXTRACT(@1,6,1,1);EXTRACT(@1,7,1,1);@2:
" ":@3:@VM:@4:@VM:@5:", ":@6:" ":@7
003:
004:
005: 25L
006: M
Bottom.
*--:
```

This virtual field may be broken down as follows:

- *Expression 1*
  (@1) TRANS(CLIENTS,CLIENT_NO,-1,'X')
  Reads the CLIENTS file, locates the record with @ID equal to CLIENT_NO, and returns the entire record, indicated by -1 in the TRANS function.
- *Expression 2*
  (@2) EXTRACT(@1,1,1,1)
  Extracts the first attribute, first value, first subvalue from the CLIENTS record. The entire CLIENTS record is the result of the first expression, indicated by @1 in the EXTRACT function. In this expression you could use @ instead of @1, since the entire CLIENTS record is the result of the expression immediately preceding the current expression. The first attribute in the CLIENTS file is FNAME.
- *Expression 3*
  (@3) EXTRACT(@1,2,1,1)
  Extracts the second attribute, first value, first subvalue from the CLIENTS record. You must use @1, not @, in this expression to reference the entire CLIENTS record, since the result of the immediately preceding expression is FNAME, not the CLIENTS record. The second attribute in the CLIENTS file is LNAME.
- *Expression 4*
  (@4) EXTRACT(@1,3,1,1)
  Extracts the third attribute, first value, first subvalue from the CLIENTS record. The third attribute in the CLIENTS file is COMPANY.
- *Expression 5*
  (@5) EXTRACT(@1,5,1,1)
  Extracts the fifth attribute, first value, first subvalue from the CLIENTS record. The fifth attribute in the CLIENTS file is CITY.
- *Expression 6*
  (@6) EXTRACT(@1,6,1,1)
  Extracts the sixth attribute, first value, first subvalue from the CLIENTS record. The sixth attribute in the CLIENTS file is STATE.
- *Expression 7*
  (@7) EXTRACT(@1,7,1,1)
  Extracts the seventh attribute, first value, first subvalue from the CLIENTS record. The seventh attribute in the CLIENTS file is ZIP.
- *Expression 8*
  (@8) @2:" ":@3:@VM:@4:@VM:@5:", ":@6:" ":@7
  Displays @2 (FNAME), concatenates a space, displays @3 (LNAME), line feed (indicated by @VM), displays @4 (COMPANY), line feed, displays @5 (CITY), concatenates a comma and a space, displays @6 (CITY), concatenates a space, displays @7 (ZIP).

The following example illustrates the output from the virtual field described above:

```
:LIST ORDERS CLIENT_INFO
LIST ORDERS CLIENT_INFO 16:32:25 Jan 27 2011 1
ORDERS.... ......................

    912 Jana Taylor
        Boulder Productions
        Bozeman, MT 83587
    801 Mary Johnson
        Meier Shipping
        Denver, CO 80203
    941 Lonnie Wilbanks
        York Software
        Abbotsford, VIC 3067
.
```

.
.

Tip: If you need to execute multiple TRANS functions against the same file in a virtual field, consider using the -1 option to return the entire record, then the EXTRACT function to retrieve the desired attributes for greater efficiency.

Hierarchy of operators

When multiple operators appear within a single statement or parenthetical expression, UniData uses the following order of precedence.

Order of pecedence	Operator	Description
First	^ or **	Exponentiation
Second	*	Multiplication
	/	Division
Third	+	Unary plus
	-	Unary minus
	+	Addition
	-	Subtraction
Fourth	: or CAT	Concatenation
Fifth	> or GT	Greater than
	< or LT	Less than
	= or EQ	Equal
	>= or => or GE	Greater than or equal to
	#>	Not greater than
	<= or =< or LE	Less than or equal to
	#<	Not less than
	<> or >< or # or NE	Not equal to
	MATCH or MATCHES	Matching
Sixth	& or AND	And
	! or OR	Or

Checking syntax

You can verify the syntax of virtual attribute formulas by using the `ECL COMPILE.DICT`, or `CD`, command. Although this command checks for valid syntax, it does not check logic. You do not have to use the `COMPILE.DICT` command on a virtual attribute for it to work.

Syntax

COMPILE.DICT *file* [*attribute_name*]
CD *file* [*attribute_name*]

If the syntax of the virtual attribute is correct, UniData displays a message similar to the following:

```
:COMPILE.DICT ORDERS EPRICE
EPRICE=PRICE*QTY
Virtual field EPRICE is syntactically correct.
:
```

If the syntax for the virtual attribute is incorrect, UniData displays an error message. In the following example, the subroutine expression in the virtual attribute is not preceded by SUBR:

```
:COMPILE.DICT ORDERS COLOR_TYPE
COLOR_TYPE=("-IFS",SUBR("-EQS",COLOR,REUSE("Black")),"Black","Not Black")
("-IFS",SUBR("-EQS",COLOR,REUSE("Black")),"Black","Not Black")
-------^
Virtual Attribute Error: COLOR_TYPE syntax error
Virtual field COLOR_TYPE has syntax error.
:
```

After you execute the `COMPILE.DICT`, or `CD`, command against a virtual attribute, UniData adds two additional attributes to the virtual attribute dictionary record:

- Attribute 8 — Stores the dictionary record location of the attributes that constitute the virtual attribute formula and displays the formula itself.
- Attribute 9 — Stores the name of the file where the virtual attribute resides. If the file name is a synonym in the VOC file, stores the full path of the file.

Note: Although you do not have to compile a virtual attribute for it to work in UniQuery or UniData SQL, you must compile it if you are executing it with the UniBasic ITYPE function. For more information about the UniBasic ITYPE function, see the *UniBasic Commands Reference*.

Chapter 6: Alternate key indexes

This chapter introduces the concept of alternate key indexing and the processes of using alternate keys and creating index files. This chapter covers the following topics:

- Creating the index file
- Creating one or more alternate key indexes
- Using indexing commands
- Updating alternate key indexes

Introduction to alternate key indexing

Alternate key indexing is a method to speed the process of looking for one or more records within a database. Alternate keys enable access to records without the need to use the record ID or to read through the entire file. An alternate key index consists of values for the alternate key and the associated @IDs of each record.

The primary key

UniData requires that each record in the database have a unique identifier, which is called the record ID, @ID, or primary key. The UniData hashing algorithm uses the @ID to locate the record in the database. When UniData looks for a record, it uses the @ID as the “key” to finding the record quickly.

The alternate key

UniData allows only one primary key for a record. If you are searching for a record based on the primary key, searches are fast. Most of the time, however, searches are based on other information contained in a record, such as a name, city, phone number, or the result of a virtual attribute calculation. These searches are slower than a search based on the primary key.

To optimize searches based on data other than the primary key, UniData provides a method to specify another attribute as the key, called the alternate key. You can build an index based on the values in an alternate key. The index consists of alternate key values and, for each value, the corresponding list of @IDs of each record. When you execute a search based on the alternate key, UniData searches the index and, when it identifies the record, uses the @ID to retrieve the complete record from the database.

For example, in the demo database the CLIENTS file contains records whose @ID is an assigned number. Other fields in the CLIENTS file include NAME, CITY, and STATE. If you create an alternate key index on STATE, you can select records for clients in a certain state without having to search the entire data file.

Alternate key file structure

UniData uses the B+ tree data structure to build the alternate key index which provides a quick way to access, retrieve, sort, and manipulate data. Each alternate key index has its own associated B+ tree.

All alternate key indexes for a data file are stored in an index file. For a static hashed file, the index is always named X_*filename*, where *filename* is the data file associated with the index file. For a dynamic or sequentially hashed file, the index file is named idx001, idx002, and so forth. The idx001 resides in the dynamic file directory with the dat001 and over001 files.

Corrupt index files

As with other UniData file types, an index file could become corrupt due to hardware failures, the interruption of a write to the index file, or an incomplete write. If an index file is corrupt, UniData displays a run time error when a UniData process tries to access the index. If the index file is associated with a recoverable file, a message is written to the sm.log.

UniData provides the guide_ndx tool to check the integrity of an index.

Syntax

```
guide_ndx{-x | -X} {1 | 2 | 3}, {index_names, ... | ALL} [-p n | -P n]
[-t template | -T template] filename
```

The following table describes each parameter of the syntax.

Parameter	Description
-x{1 | 2 | 3}	Determines the type of checking guide_ndx performs: ▪ 1 – Perform physical checking ▪ 2 – Perform logical checking ▪ 3 – Perform physical and logical checking
index_names	The index names you want guide_ndx to check. Separate each index name with a comma, or enter ALL to check all indexes for the file.
-p *n*	Determines the number of concurrent processes to use when checking the index.
-t *template*	The template to use for output files. The default is GUIDE.
filename	The name of the data file containing the index.

guide_ndx produces 2 files, GUIDE_XERRORS.LIS and GUIDE_XSTATS.LIS. GUIDE_XERRORS.LIS contains errors detected in the index, and GUIDE_XSTATS.LIS contains statistics about the index.

How UniData handles alternate keys

You can create alternate key indexes any time after the data file is created. There is no limit on the number of alternate keys you can create for a single file.

Alternate key attributes

UniData supports both D-type (data) and V-type (virtual) attributes as alternate keys. Attributes can be singlevalued, multivalued, or multi-subvalued. For static hashed files and dynamic hashed files, you can build an index on the @ID, providing a fast way to access records whose @IDs fall into a certain value range.

Types of keys

UniData sorts data by using specific algorithms based on whether the data is numeric or text. UniData sorts alternate keys differently according to the format of the attributes they index. There are two types of keys:

- Text keys — If the format of an attribute being indexed is left justified or centered text, UniData sorts it as a text key.
- Numeric keys — If the format of an attribute being indexed is right justified, UniData sorts it as a numeric key.

The difference between text and numeric keys is important in an index, because the items are sorted differently. If the justification in the dictionary record for an attribute is incorrect, the index is sorted incorrectly, and query statements against the index may produce unexpected results. For example, if a date field is defined as left-justified in the dictionary record, SELECT statements for a certain date range may not return all records meeting the selection criteria, since all numeric fields should be right justified.

Alternate key size

When you create the first index for a data file, UniData prompts for an alternate key length. The alternate key length is used to determine the node size of the B+ tree for all indexes created for that data file. Make sure that the alternate key length is at least as large as the longest attribute being indexed. If you do not specify the alternate key length, UniData uses a length of 20 as the default.

Creating and deleting an alternate key index

Creating an alternate key index requires that you follow these steps:

- Create the index file and the alternate key index with the CREATE.INDEX command.
- Populate the alternate key index with the BUILD.INDEX command.

Refer to Alternate key index commands, on page 97 for a list of commands to create and manage your index files. For more information on alternate key commands, see the *UniData Commands Reference*.

Creating the index file

When you execute the CREATE.INDEX command for the first time for a specific file, UniData does two things:

- Creates the index file for the data file.
- Creates the alternate key index for each attribute you designate. UniData does not populate the index at this time.

> **Note:** You cannot create a UniData index on a file you have converted to an external database using External Database Access (EDA).

Syntax

CREATE.INDEX *filename attribute1* [*attribute2...*] [NO.DUPS] [NO.NULLS]

The following table describes the parameters of the CREATE.INDEX command.

Parameter	Description
filename	Name of the UniData data file.
attribute	Data attribute on which UniData creates an alternate key index. You can name multiple data attributes on the command line to create multiple alternate key indexes simultaneously.
NO.DUPS	Prevents duplicate data values from being written to an alternate key index through automatic index updating or the BUILD.INDEX command. If duplicate values exist in the data file, BUILD.INDEX fails, the alternate key index containing duplicate values is not built, and an error message is displayed. If the alternate key index already exists and you try to add a duplicate value to that attribute in the data file, the write to the index and the data file fails and an error message is displayed.
NO.NULLS	Specifies that no empty strings are to be included in the index. The NO.NULLS option does not prevent the null value from being included in the index.

Note: The NO.DUPS option is not supported for recoverable files.

Warning: If you are using the NO.DUPS option within a transaction, and the Recoverable File System is disabled, you risk partially committed transactions if one of the updates introduces a duplicate value. This is because the transaction cannot be completely rolled back without RFS. For example, if you have a transaction containing 5 record updates, with the third update introducing a duplicate value, UniData writes the first two updated records to disk, but the third update fails and UniData aborts the transaction.

Enter the CREATE.INDEX command at the ECL prompt. When you execute CREATE.INDEX for the first time, the index file itself is created, as well as an alternate key index for each attribute you specify. Subsequently, UniData creates an alternate key index for the designated attributes, but does not create another index file. You can create more than one alternate key index at a time.

In the following example, an alternate key index is created for the FNAME and LNAME attributes of the CLIENTS demo database file, with an alternate key length of 25:

```
:CREATE.INDEX CLIENTS FNAME LNAME
Alternate key length (default 20): 25
"FNAME" created
"LNAME" created
```

The following example illustrates creating an alternate key index on the CITY attribute of the CLIENTS demo database with the NO.DUPS option:

```
:CREATE.INDEX CLIENTS CITY NO.DUPS
"CITY" created.
```

Note: Since the FNAME and LNAME alternate key indexes were previously created, UniData does not reprompt for alternate key length.

Building the index file

The BUILD.INDEX command causes UniData to populate the alternate key index with values from the data file.

Syntax

BUILD.INDEX *filename attribute1* [*attribute2...*]
BUILD.INDEX *filename* **ALL**

where *filename* is the name of the data file and *attribute* is the name of the alternate key index you are populating. You can populate a specific alternate key index, or you can use the ALL keyword to populate all alternate key indexes at once.

In the following example, BUILD.INDEX populates the FNAME and LNAME alternate key indexes:

```
:BUILD.INDEX CLIENTS FNAME LNAME
One "*" represents 1000 records

Building "FNAME" ...
Building "LNAME" ...

130 record(s) processed.
```

In the next example, BUILD.INDEX is executed to populate the CITY alternate key index. BUILD.INDEX fails because the NO.DUPS option was specified when the alternate key index was created in the previous example, and the data file contains duplicate values:

```
:BUILD.INDEX CLIENTS CITY
One "*" represents 1000 records

Building "CITY" ...
There are duplicates while building CITY (not built).
:
```

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

Note: You cannot create a UniData index on a file you have converted to an external database using External Database Access (EDA).

Displaying an index

You can display the information about an index file and the alternate key indexes using the LIST.INDEX command.

Syntax

LIST.INDEX *filename* [*attribute1* [*attribute2...*] | ALL] [STATISTICS | DETAIL] [NO.PAGE] [LPTR *n*]

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

Parameter	Description
filename	Name of the UniData data file.

Parameter	Description
attribute | ALL	One or more alternate key indexes to be examined. If you do not specify an attribute, UniData displays information about all alternate key indexes created for the file.
STATISTICS	Provides statistical information about the alternate key indexes, including number of keys, number of overflowed keys, average number of records per alternate key, minimum records per alternate key, and maximum records per alternate key.
DETAIL	Produces a detailed listing of each alternate key value in an alternate key index, the number of records containing the alternate key value, and the number of overflowed keys. Also displays statistical information about the alternate key indexes, including number of keys, number of overflowed keys, average number of records per alternate key, minimum records per alternate key, and maximum records per alternate key.
NO.PAGE	Prevents the report from pausing at the end of a full screen.
LPTR *n*	Directs the report to a logical printer represented by *n*.

The following example displays information about the alternate key indexes for the CLIENTS file:

```
:LIST.INDEX CLIENTS
Alternate Key Index Details for File CLIENTS        Page    1
File.................  CLIENTS
Alternate key length..  25
Node/Block size.......  4K
OV blocks............  1 (0 in use, 0 overflowed)
Indices..............  3 (3 D-type)
Index updates.........  Enabled, No updates pending
Index-Name...... F-type K-type Built Empties Dups In-DICT S/M F-no/VF-expr....
FNAME    D    Txt      Yes Yes      Yes      Yes      S    1
LNAME    D    Txt      Yes Yes      Yes      Yes      S    2
CITY     D    Txt      No Yes  No   Yes      S    5
```

For more information about `LIST.INDEX`, see the *UniData Commands Reference*.

Deleting an alternate key index

The `DELETE.INDEX` command deletes the specified alternate key index from an index file. If you use the ALL option, all alternate keys and the index file itself are deleted.

Syntax

`DELETE.INDEX` *`filename {attribute1 [attribute2...] | ALL}`*

where *filename* is the name of the data file and *attribute* is the name of each alternate key index you want to delete.

> **Tip:** Occasionally index files can become corrupt due to hardware or software failures. In these cases, we recommend that you use the ALL option with `DELETE.INDEX` to delete the index file and all alternate key indexes, then rebuild the index file and the alternate key indexes.

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

Alternate key index commands

The following table describes commands to create and manage alternate key indexes.

Command	Description
CREATE.INDEX	Creates the index file for the specified data file the first time it is executed, and creates an alternate key index for each specified attribute.
BUILD.INDEX	Populates an alternate key index with values.
DELETE.INDEX	Deletes an alternate key index. If the ALL option is specified, also deletes the index file.
DISABLE.INDEX	Disables automatic index updates.
ENABLE.INDEX	Enables automatic index updates.
LIST.INDEX	Displays summary, detail, and statistical information about an index.
UPDATE.INDEX	Updates an alternate key index previously disabled by the DISABLE.INDEX command.
guide_ndx	Checks the integrity of the index.

See the *UniData Commands Reference* for command syntax and detailed information about the preceding commands.

Alternate keys in UniBasic

UniBasic programs can access alternate key indexes if by using the `SELECT` or `SETINDEX` command. The following table lists the commands that use indexes.

Command	Description
INDICES	Returns the names of alternate key indexes or information about a particular alternate key index.
SELECT	Creates a select list based on an alternate key index.
SETINDEX	Sets a pointer to a key in an alternate key index.
READFWD	The first `READFWD` retrieves the alternate key set by `SETINDEX`, then each subsequent `READFWD` retrieves the next alternate key value in the index.
READFWDL	The first `READFWDL` retrieves the alternate key set by `SETINDEX`, then each subsequent `READFWDL` retrieves the next alternate key value in the index. `READFWDL` sets a shared (L) lock before reading the record.
READFWDU	The first `READFWDU` retrieves the alternate key set by `SETINDEX`, then each subsequent `READFWDU` retrieves the next alternate key value in the index. `READFWDU` sets an exclusive (U) lock before reading the record.
READBCK	The first `READBCK` retrieves the alternate key set by `SETINDEX`, then each subsequent `READBCK` retrieves the previous alternate key value in the index.
READBCKL	The first `READBCKL` retrieves the alternate key set by `SETINDEX`, then each subsequent `READBCKL` retrieves the previous alternate key value in the index. `READBCKL` checks for locks; if the record is available, it sets a shared (L) lock and reads the record.
READBCKU	The first `READBCKU` retrieves the alternate key set by `SETINDEX`, then each subsequent `READBCKU` retrieves the previous alternate key value in the index. `READBCKU` checks for locks; if the record is available, it sets an exclusive (U) lock.

For a complete description of the commands shown in the preceding table, see the *UniBasic Commands Reference*.

Alternate key indexes in UniQuery

When alternate key indexes exist for a data file, UniQuery statements may execute faster. You can force UniData to use an existing alternate key index by including the REQUIRE.INDEX keyword in the query. Conversely, you can bypass using an existing alternate key index by including the NO.INDEX keyword in the query.

Alternate indexes in UniQuery statement

Prior to UniData 5.2, only one index was used in the selection criteria of a UniQuery statement, even if the selection criteria contained more than one indexed attribute. UniQuery now uses all available indexes when processing a statement containing selection criteria.

If a D-type index exists for a specific attribute, any synonym D-type dictionary record for the same attribute also uses the index.

Note: You cannot use an alternate index with a LIKE clause when the comparative attribute is numeric.

Virtual attribute alternate key indexes

When you create and build an alternate key index for a virtual attribute, UniData stores the values of the virtual attribute in the index file. This is unlike the way UniData treats a data file; UniData does not store values in a data file for a virtual attribute. When UniData builds the alternate key index for a virtual attribute, it evaluates the virtual attribute for each record and adds the values to the B+ tree structure.

When you modify a record in a data file that has an index, UniData modifies the alternate key index for the virtual attribute also, through automatic updating. UniData reevaluates the virtual attribute, deletes the current value in the index file, then adds the new value to the index. However, UniData does not automatically update an alternate key index for a virtual attribute when the formula in the virtual attribute dictionary record changes.

Although UniData supports virtual attribute alternate key indexes, care should be taken when determining which virtual attributes are indexed. A virtual attribute that contains a TRANS statement to another file, or the same file, is not a recommended use of an alternate key index, unless the file being translated to does not change. If the file being translated to does change, the alternate key index is not updated. Therefore, to obtain accurate results from a UniQuery statement, the alternate key index would need to be rebuilt each time the associated file is updated.

Alternate key index automatic updating

By default, each time you modify an attribute in a data record that has an alternate key index, UniData automatically updates the associated alternate key indexes. Updating alternate key indexes can slow the performance of your application, since the alternate key update requires at least one additional write or delete operation.

For instance, if a file has 10 alternate key indexes defined, the system may require at least 20 extra I/O operations and another 20 search operations to write only one data record. If there are no alternate key indexes defined, the system requires only one write to disk.

Disabling automatic updating

You can temporarily turn off automatic updating to an alternate key index on a file-by-file basis by using the DISABLE.INDEX command.

If you disable automatic updating, UniData saves the changes to the data file in an index log file. For static hashed files, this file is called x_*filename* on UniData for UNIX and L_*filename* on UniData for Windows Platforms, where *filename* is the name of the data file. For dynamic hashed files, this file is called xlog001. This method of saving changes saves system resources, since UniData writes the changes only once. Since UniData stores the updates, you can update your alternate key indexes at a more convenient time, such as overnight or over a weekend.

You cannot disable an index for a data file if any of the alternate key indexes do not allow duplicate values. In the following example, DISABLE.INDEX for the CLIENTS file fails, since CITY does not allow duplicate entries:

```
:LIST.INDEX CLIENTS
Alternate Key Index Details for File CLIENTS                Page
1

File.................  CLIENTS
Alternate key length..  25
Node/Block size.......  4K
OV blocks.............  1 (0 in use, 0 overflowed)
Indices...............  3 (3 D-type)
Index updates.........  Enabled, No updates pending

Index-Name......  F-type K-type Built Empties Dups In-DICT S/M F-
no/VF-expr....
FNAME          D     Txt   Yes   Yes    Yes  Yes    S   1
LNAME          D     Txt   Yes   Yes    Yes  Yes    S   2
CITY           D     Txt   No    Yes    No   Yes    S   5

:DISABLE.INDEX CLIENTS
Index can't be disabled due to NO.DUPS
:
```

Tip: If an index file is disabled, you cannot use the REQUIRE.INDEX keyword in a UniQuery statement.

Applying deferred updates

To update the index file with saved updates, use the UPDATE.INDEX command. When you issue this command, UniData updates the index file by applying the stored updates from the log file (x_*filename* on UniData for UNIX or L_*filename* on UniData for Windows Platforms, or xlog001) to the index file. Automatic updating can be enabled or disabled when you execute UPDATE.INDEX.

Note: When you enable automatic indexing for a recoverable file with the ENABLE.INDEX command, deferred updates are automatically applied to the index file. You do not need to execute UPDATE.INDEX after enabling automatic updating.

Enabling automatic updating

If you have turned automatic indexing off with the DISABLE.INDEX command, you can turn it back on with the ENABLE.INDEX command. When you turn on automatic indexing, UniData handles the deferred updates for recoverable and nonrecoverable files in the following ways:

- Recoverable Files — resumes updating the index file with subsequent modifications and applies the deferred updates.
- Nonrecoverable Files — resumes updating the index file with subsequent modifications, but does not apply the deferred updates. To apply deferred updates, you must use the UPDATE.INDEX command.

Tip: If you are using nonrecoverable files, you should execute UPDATE.INDEX immediately following ENABLE.INDEX to avoid data inconsistency.

Chapter 7: UniData paragraphs

This chapter describes the use of the Process Control Language (PCL) in paragraphs. PCL broadens the tasks a paragraph can perform without requiring you to modify the paragraph each time you want to execute it.

Note: You cannot use PCL with UniData SQL.

PCL provides the following elements to create paragraphs:

- Comments — By entering an asterisk (*) as the first character of line, you can enter a comment. The asterisk prevents execution of a command line.
- Inline prompting — Inline prompting prompts for user input during the execution of a paragraph.
- DATA statements — DATA statements pass data entered as literals in the paragraph, or entered through inline prompting, to other processes or UniBasic programs.
- IF/THEN conditional statements — These statements control branching within a paragraph depending upon the results of a specific process, user input, or the value in a system variable.
- Branching — By using the GO keyword, you can branch to a statement label within the paragraph.
- LOOP/REPEAT command flow — LOOP/REPEAT statements execute a portion of the paragraph repetitively until the loop exit condition is reached.
- LOGIN/LOGOUT paragraphs — You can create LOGIN and LOGOUT paragraphs which are executed each time you log on and log off of UniData, respectively.

Comments in a paragraph

When you begin a line of a paragraph with an asterisk (*) or a hash mark (“#), UniData interprets the line as a comment rather than a command. This allows you to add explanatory text to the paragraph, or prevent the execution of a particular line in a paragraph.

Syntax

```
* text
or
# text
```

In the following example, two lines of a paragraph contain comments. Line 2 contains informational text, while line 4 of the paragraph contains a UniQuery statement. Because the asterisk is the first character of the line, neither of these statements is executed, as shown in the following example:

```
:AE VOC EXAMPLE
Top of "EXAMPLE" in "VOC", 5 lines, 157 characters.
*--: P
001: PA
002: *Paragraph to select all NY clients
003: SELECT CLIENTS WITH STATE = 'NY'
004: *LIST CLIENTS BY CITY NAME CITY STATE PHONE_NUM ID.SUP
005: LIST CLIENTS BY NAME NAME CITY
Bottom.

:EXAMPLE
LIST CLIENTS BY NAME NAME CITY 11:41:48 Feb 07 1998 1
CLIENTS... Name.......................... City...........

10055      Gross, Cathy                  Lowell
9965       Phillips, Gary                New York
2 records listed
```

Tip: You can also indicate a comment line by specifying the question mark (?) as the last character of a line.

Inline prompting

In a UniData paragraph, you can leave some items unspecified until the execution of the paragraph by using inline prompting. The user input in response to an inline prompt is used as the values necessary to complete the statement.

Warning: Do not use an inline prompt after a HUSH ON statement in a paragraph.

You can use conversion codes and pattern matching to check the input from an inline prompt. To differentiate a conversion code from a pattern match, you must enclose the conversion code in parentheses.

Syntax

```
<<[options,]prompt_text[,check | (conv)]>>
```

The following table lists the inline prompting parameters.

Parameter	Description
options	Controls how the response to inline prompt is handled. See “Valid Options for Prompting” in this chapter for more information.
prompt_text	Text UniData displays to prompt the user for input. Cannot contain commas, except for the comma that precedes a conversion code.
check	Pattern that UniData uses to verify user input. See “Prompting Checks” in this chapter for more information.
conv	Conversion code UniData uses to verify user input. You can use any OCONV conversion code.

In the following example, a paragraph contains an inline prompt, which prompts the user for the state code each time the paragraph is executed. The inline prompt, as well as user input, appears in boldface:

```
:AE VOC EXAMPLE
Top of "EXAMPLE" in "VOC", 4 lines, 141 characters.
```

```
*--: P
001: PA
002: *Paragraph to select clients in a specified state
003: SELECT CLIENTS WITH STATE = <<Enter State Code>>
004: LIST CLIENTS BY CITY NAME CITY STATE PHONE_NUM ID.SUP
Bottom.
```

When you execute the preceding paragraph UniData prompts for the state code, as shown in the following example.

```
:EXAMPLE
Enter State Code NY
LIST CLIENTS BY CITY NAME CITY STATE PHONE_NUM ID.SUP 12:46:20 Feb 07 1998 1
Name..................... City.......... State/Territory Phone Number..
Gross, Cathy                              Lowell
          NY                      6096460030
6096469570
Phillips, Gary                          New York
      NY                     2125556161
2125556162
2 records listed
```

Note: If you enter quit or `QUIT` in response to an inline prompt with UDT.OPTIONS 77 ON, control is returned to the calling process. If UDT.OPTIONS 77 is off, UniData aborts the paragraph process and returns to the ECL prompt.

Prompt options

UniData can do more than simply accept user input at an inline prompt. For example, you can sound the terminal bell if input is incorrect, or UniData can redisplay the same prompt or automatically use prior input. The following table lists inline prompt options.

Option	Description
@(CLR)	Clears the screen.
@(BELL)	Rings the terminal bell.
@(COL,ROW)	Displays the prompt at the position specified by COL (column) and ROW.
@(TOF)	Positions the prompt at the top left of the screen.
A,*prompt_text*	Always prompts for input. Without the A option, if identical *prompt_text* appears more than once in a paragraph, UniData stores the input to the first *prompt_text* and automatically substitutes this input to further occurrences of *prompt_text* without reprompting. If the A option is used, UniData reprompts when another occurrence of *prompt_text* is encountered.
C*n*	Uses the *n*th word on the command line as an argument in an inline prompt. This option allows the user to enter responses to inline prompts at ECL on the same line following the paragraph name.

Option	Description
F(*filename,record,*[*.att.num, value.num,subvalue.num,*]	Retrieves input from *record* in *filename,* and optionally from *att.num, value.num,* and *subvalue.num.* Prompt text is optional.
I*n*	Uses the *n*th word on the command line as an argument in an inline prompt, just as the C*n* option, but prompts for *n* if *n* is not specified.
R,*prompt_text*	Repetitively prompts until you press ENTER without entering a value.
R(*string*),*prompt_text*	Repetitively prompts until you press ENTER without entering a value, inserting *string* between each entry.
S*n*	This option is used for paragraphs called by other paragraphs. Uses the *n*th word from the original command line (just as C*n* or I*n*), but passes the value to the called paragraph as well as using the value in the current paragraph.

Validating user input with pattern matching

You can validate user input by inserting a pattern or a text string in an inline prompt. With pattern matching, user input must have a matching alphanumeric format. A text string indicates that the input must be identical to the text enclosed in quotation marks.

The following table describes valid patterns.

Pattern	Description
0X or ...	Any number of characters, including no characters.
*n*X	*n* number of any character.
[~]0A	Any number of alphabetic characters, including none.
[~]*n*A	*n* number of alphabetic characters.
*n-m*A	*n* to *m* number of alphabetic characters.
*n-m*X	*n* to *m* number of any characters.
[~]0N	Any number of numeric characters, including none.
[~]*n*N	*n* number of numeric characters.
*n-m*N	*n* to *m* number of numeric characters.
[~]*text*	Exact text. Text must be enclosed in quotation marks.

In the above pattern matches, ~ indicates NOT.

Many special characters, such as the period (.), hyphen (-), and forward slash (/) are supported in pattern matching check strings without being enclosed in quotation marks. Thus, 2N/2N/2N checks for 2 numerics, a slash, 2 numerics, a slash, 2 numerics.

Converting user input to internal format

Inline prompts can also contain conversion codes to check the validity of the input string. Common uses for conversion codes are (D) to ensure a date is input, and (MD) to ensure a valid number is input. The conversion code must be enclosed in parentheses.

In the following example, the (D) conversion code is used in the inline prompt. When a value is entered that is not a date, UniData returns an error message, as shown in the following example:

```
:AE VOC TEST
Top of "TEST" in "VOC", 2 lines, 75 characters.
*--: P
001: PA
002: LIST ORDERS WITH ORD_DATE LT <<ENTER DATE,(D2/)>> ORD_DATE PROD_NAME QTY
Bottom.
*--: q
:TEST
ENTER DATE COLORADO
ICONV D2/ error.
```

Always prompting with an inline prompt

UniData automatically uses the input from an initial inline prompt as values for subsequent identical inline prompts. For example, the following paragraph prompts for the desired state code in line 2, and in line 3. UniData automatically uses the response to the first inline prompt as input to the second inline prompt without reprompting:

```
:AE VOC PARA
Top of "PARA" in "VOC", 3 lines, 156 characters.
*--: P
001: PA
002: LIST CLIENTS WITH STATE = <<Enter State Code>> NAME STATE
003: LIST CLIENTS WITH STATE = <<Enter State Code>> AND WITH CITY
= "<<Enter City>>" NAME CITY STATE
Bottom.
*--:
:PARA
Enter State Code CA
LIST CLIENTS WITH STATE = CA NAME STATE 11:24:34 Feb 11 1998 1
CLIENTS... Name.......................... State/Territory

9982       Willette, Marc                CA
9986       Gunter, Sam                   CA
2 records listed

Enter City Los Angeles
LIST CLIENTS WITH STATE = CA AND WITH CITY = "Los Angeles" NAME
CITY STATE 11:25:14 Feb 11 1998 1
CLIENTS... Name.......................... City...........
State/Territory

9982       Willette, Marc                Los Angeles     CA
1 record listed
```

To force UniData to reprompt when it encounters a subsequent identical inline prompt, use the A (always prompt) option in subsequent identical inline prompts:

```
:AE VOC PARA
Top of "PARA" in "VOC", 3 lines, 158 characters.
*--: P
001: PA
002: LIST CLIENTS WITH STATE = <<Enter State Code>> NAME STATE
003: LIST CLIENTS WITH STATE = <<A,Enter State Code>> AND WITH
CITY = "<<Enter City>>" NAME CITY STATE
Bottom.
*--:
:PARA
Enter State Code CA
LIST CLIENTS WITH STATE = CA NAME STATE 11:48:06 Feb 11 1998 1
CLIENTS... Name..................... State/Territory

9982       Willette, Marc                CA
9986       Gunter, Sam                   CA
2 records listed

Enter State Code CA
Enter City Los Angeles
LIST CLIENTS WITH STATE = CA AND WITH CITY = "Los Angeles" NAME
CITY STATE 11:48:33 Feb 11 1997 1
CLIENTS... Name..................... City.....
State/Territory

9982       Willette, Marc                Los Angeles     CA
1 record listed
```

You can also use the CLEARPROMPTS command to clear all previous responses to inline prompts and force UniData to reprompt when it encounters an identical inline prompt. The following example illustrates a paragraph containing the CLEARPROMPTS command before the second occurrence of an identical inline prompt. Because all previous responses are cleared, UniData reprompts for the state code:

```
:AE VOC PARA
Top of "PARA" in "VOC", 4 lines, 169 characters.
*--: P
001: PA
002: LIST CLIENTS WITH STATE = <<Enter State Code>> NAME STATE
003: CLEARPROMPTS
004: LIST CLIENTS WITH STATE = <<Enter State Code>> AND WITH CITY
= "<<Enter City>>" NAME CITY STATE
Bottom.
*--:
```

Stacking inline responses

The C option in an inline prompt takes the *n*th entry from the ECL command line as the response to an inline prompt. This option allows the user to execute the paragraph and respond to all inline prompts in one line.

```
:AE VOC PARA
Top of "PARA" in "VOC", 2 lines, 99 characters.
001: PA
002: LIST CLIENTS WITH STATE = <<C2, State Code>>
 AND WITH CITY = "<<C3,Enter City>>" NAME CITY STATE
Bottom.
*--:
```

```
:PARA CO Denver
LIST CLIENTS WITH STATE = CO AND WITH CITY = "Denver"
 NAME CITY STATE 14:32:06 Feb 11 1998 1
CLIENTS... Name..................... City.......... State/Territory
10018      Johnson, Mary           Denver     CO
1 record listed
```

In the previous example, CO is the second entry on the command line, and Denver is the third entry on the command line. The C2 option in the State Code inline prompt therefore substitutes CO for the value, while the C3 option in the Enter City inline prompt substitutes Denver for the value.

The I and S options are similar to the C option. The I*n* option prompts for text if *n* is not specified, and the S*n* option passes the value of *n* to called paragraphs.

Retrieving data from file through inline prompting

The F option in an inline prompt retrieves data from a file and uses that value as the response to an inline prompt.

In the following example, the inline prompt requests a client number. When the client number is entered, UniData reads the CLIENTS file and retrieves attribute 8 of the requested record, then includes the attribute value in a UniQuery statement:

```
:AE VOC PARA3
Top of "PARA3" in "VOC", 2 lines, 85 characters.
001: PA
002: LIST ORDERS WITH COUNTRY = '<<F(CLIENTS,<<Enter Client
Number>>,8)>>' CITY COUNTRY
Bottom.

:PARA3
Enter Client Number 10054
LIST ORDERS WITH COUNTRY = 'France' CITY COUNTRY 12:18:30 Feb 11 1998 1
ORDERS.... City.......... Country...
      830  Paris       France
      832  Marseille       France
      913  Lyon         France
.
.
.
```

Creating repetitive inline prompts

The R inline prompt option repetitively prompts for input until you press ENTER without entering a response. The values entered are concatenated to each other with spaces between them, creating OR conditions in the resulting UniQuery statement:

```
:AE VOC PARA1
Top of "PARA1" in "VOC", 2 lines, 53 characters.
*--: P
001: PA
002: LIST ORDERS WITH CITY = <<R,Enter City>> NAME CITY
Bottom.
*--:
:PARA1
Enter City Philadelphia
```

```
Enter City Paris
Enter City Denver
Enter City
LIST ORDERS WITH CITY = Philadelphia Paris Denver NAME CITY
15:43:02 Feb 11 1998 1

ORDERS.... Name..................... City...........
801              Mary Johnson                    Denver
830              Fernando Ducrot                 Paris
859              Omar Saulnier               Paris
888              Alicia Rodriguez            Philadelphia
.
.
.
```

Commands available in paragraphs

UniData supplies the following commands for use in paragraphs. These commands control the execution of paragraphs:

- DATA statement
- IF/THEN command flow
- LOOP/REPEAT command flow
- GO command andlLabels

DATA statement

You can use a DATA statement in a paragraph to pass data entered as a literal, entered through inline prompting, to other processes or UniBasic programs.

Syntax

DATA *input_value*

The *input_value* must appear immediately following the DATA statement.

When a UniBasic program contains an INPUT statement that would normally require input from the terminal, the *input_value* of the first DATA statement in the paragraph is used. The second INPUT statement in a UniBasic program uses the *input_value* of the second DATA statement in the paragraph, and so forth. When no more DATA statements are present in the paragraph, UniBasic prompts for subsequent INPUT statements from the terminal.

You can also use responses to inline prompts as DATA statements, but the DATA statement cannot be used to provide responses to inline prompts.

The following example illustrates a paragraph which runs a UniBasic program and passes NY as the response to the INPUT statement in the UniBasic program:

```
:AE VOC CUST_INFO
Top of "CUST_INFO" in "VOC", 3 lines, 27 characters.
*--: P
001: PA
002: RUN BP CUST_INFO
003: DATA NY
Bottom.
*--:
```

```
:CUST_INFO
Enter State Code :?NY

2 records selected to list 0.

LIST CLIENTS NAME CITY STATE 12:11:03 Feb 13 1998 1
CLIENTS... Name..................... City..........
State/Territory

10055        Gross, Cathy                              Lowell          NY
9965         Phillips, Gary                            New York        NY
2 records listed

Enter State Code :?
```

In the previous example, NY is entered as response to the inline prompt and the UniQuery statements contained in the UniBasic program are executed. The user is prompted to “Enter State Code” after the initial report is displayed because there are no further DATA statements in the paragraph, and control is returned to the terminal for input. To terminate the program, press ENTER.

The UniBasic program called from the preceding paragraph appears below:

```
:AE BP CUST_INFO
Top of "CUST_INFO" in "BP", 8 lines, 164 characters.
*--: P
001: LOOP
002: PRINT "Enter State Code :":
003: INPUT STATE
004: UNTIL STATE = ''
005: EXECUTE "SELECT CLIENTS WITH STATE = ":STATE
006: EXECUTE "LIST CLIENTS NAME CITY STATE"
007: REPEAT
008: END
Bottom.
*--:
```

IF/THEN command flow

IF/THEN command flow allows you to alter the sequence in which statements are executed in a paragraph, depending upon the result of a conditional test. When IF/THEN statements are not present in a paragraph, each statement is executed sequentially. When IF/THEN statements are present, and the condition is true, UniData executes whatever follows THEN in the paragraph. If the condition is not true, UniData executes the next sequential statement.

Syntax

IF *condition* **THEN** *true_result*

where *condition* is the test to be performed and *true_result* is any valid command. The *condition* has the following format:

parameter1relational_operator parameter2

parameter can be a constant, an inline prompt, or an @variable. For a list of valid relational operators, see Creating virtual attributes, on page 71.

The following table describes valid @ variables for use with IF/THEN command flow.

@variable	Description
@ACCOUNT	Path from which you first entered UniData. The value of @ACCOUNT does not change when a LOGTO statement is executed.
@DATE	System date represented in internal format.
@DAY	Two-digit day of the current month.
@GID	Group ID assigned to the user.
@LOGNAME	User's login name.
@MONTH	Two-digit representation of the current month.
@PATH	Current path of the directory where the user resides. The value of @PATH changes with the LOGTO command.
@SYTEM.RETURN .CODE	Value indicating a condition or error after the execution of an ECL command. For most commands, 0 indicates that the command completed correctly, and -1 indicates the command was not completed correctly.
@TIME	Current time represented in internal format.
@TTY	Terminal port name, such as tty01, tty1a, console, etc.
@UID	UNIX user ID number for the user.
@USERNAME	UNIX user name.
@USERNO	Current udt process ID number.
@USER.RETURN .CODE	User-defined message.
@WHO	Retrieves the current directory (for example, if you are under /usr/ud72/demo, the value of @WHO is demo).
@YEAR	Two-digit representation of the year.

The following example illustrates using an inline prompt with IF/THEN command flow:

```
:AE VOC TEST1
Top of "TEST1" in "VOC", 4 lines, 157 characters.
*--: P
001: PA
002: DISPLAY Do You Want to Process the Report Now?
003: IF <<A,Enter YES/NO>> = 'YES' THEN LIST CLIENTS BY STATE NAME
STATE PHONE_NUM
004: DISPLAY Not Processing Report
Bottom.
*--:
```

In the next example, the IF/THEN command flow tests for the current month using the @MONTH @variable. If the month is February, UniData processes the report, otherwise the report is not processed:

```
:AE VOC TEST2
Top of "TEST2" in "VOC", 3 lines, 139 characters.
*--: P
001: PA
002: IF @MONTH = '02' THEN LIST INVENTORY WITH INV_DATE <
'01/01/96' PROD_NAME INV_DATE
003: DISPLAY Current month is not February, Not Processing
Bottom.
```

```
*--:
```

The GO command and labels

The GO command redirects the execution sequence of statements in a paragraph unconditionally.

Syntax

GO *label*

where *label* uniquely identifies the label for a specific command or set of commands within a paragraph. The label and commands referenced by a GO command must appear in the same paragraph.

You may identify any line of a paragraph as a label. Labels:

- Must be unique within the paragraph.
- Must end with a colon (:) and a space.
- May be strictly numeric, strictly alphabetic, or may begin with an alpha character followed by any combination of alphanumeric characters.

Syntax

label: [*command*]

If *command* is not specified, UniData executes the first command encounters after the label.

The GO command is commonly used in LOOP/REPEAT and IF/THEN command flow. The following example shows the UniData-supplied LISTDICT paragraph, which uses IF/THEN command flow, inline prompts, and GO commands:

```
:AE VOC LISTDICT
Top of "LISTDICT" in "VOC", 8 lines, 317 characters.
*--: P
001: PA
002: IF <<C3,DISPLAY>> = '' THEN GO L100
003: IF <<C3,DISPLAY>> = 'LPTR' THEN GO L200
004: IF <<C3,DISPLAY>> # 'lptr' THEN GO L100
005: L200: SORT DICT <<I2,FILE NAME>> TYP LOC CONV MNAME FORMAT SM
ASSOC BY TYP BY LOC BY @ID LPTR
006: GO DONE
007: L100: SORT DICT <<I2,FILE NAME>> TYP LOC CONV MNAME FORMAT SM
ASSOC BY TYP BY LOC BY @ID
008: DONE: *
Bottom.
*--:

:LISTDICT INVENTORY
LIST DICT INVENTORY BY TYP BY @ID TYP LOC CONV NAME FORMAT SM
ASSOC 11:36:39 Feb 14 1998 1
@ID............ TYP LOC.......... CONV NAME........... FORMAT SM
ASSOC.....

@ID             D               0       INVENTORY       10L     S
COLOR           D               5       Color           10T     MV
LINE_ITEMS
FEATURES        D               4       Features        30T     S
.
.
.
```

The preceding paragraph performs the following:

- Line 2 — If the third word on the command line (identified by C3 in the inline prompt) is an empty string, proceed to line 7, which is identified with the label L100.
- Line 3 — If the third word on the command line (identified by C3 in the inline prompt) equals LPTR, proceed to line 5, which is identified with the label L200.
- Line 4 — If the third word on the command line (identified by C3 in the inline prompt) is not equal to lptr, proceed to line 7, which is identified with the label L100.
- Line 5 — The command identified by the label L200. L200 is followed by a colon and space, distinguishing it as a label. I2 in the inline prompt retrieves the second word on the command line, and if none was entered, prompts for FILE NAME.
- Line 6 — Proceed to line 8, identified by the label DONE.
- Line 7 — The command identified by the label L100. L100 is followed by a colon and space, distinguishing it as a label. I2 in the inline prompt retrieves the second word on the command line, and if none was entered, prompts for FILE NAME.
- Line 8 — The command identified by the label DONE. In this case, the asterisk (*) represents a comment line. Since no additional commands follow the DONE label, the paragraph terminates.

LOOP/REPEAT command flow

The LOOP/REPEAT command flow structure repetitively executes a block of commands until UniData encounters a GO command to a label outside the LOOP/REPEAT command.

Syntax

LOOP

```
command 1
command 2
command 3
.
.
.
loop.exit
REPEAT
```

The *loop.exit* condition terminates a loop and occurs within a LOOP/REPEAT or IF/THEN command flow. The *loop.exit* may appear anywhere within the loop, but generally appears immediately following the LOOP statement or immediately preceding the REPEAT statement.

Warning: If you use inline prompting within a loop, make sure you use the A (always prompt) option. If you do not use the A option, the inline prompt is executed only when UniData encounters it the first time, and there is no opportunity to exit the loop. If you fail to include a method for UniData to exit a loop, a command could loop indefinitely.

The following example uses the LOOP/REPEAT command flow to prompt for executing a report. UniData prompts repetitively until the user answers “N” to the prompt when processing proceeds to the command identified by the DONE label:

```
:AE VOC TEST_LOOP
Top of "TEST_LOOP" in "VOC", 6 lines, 140 characters.
*--: P
001: PA
002: LOOP
003: IF <<A,Do you want to process the report?>> = "N" THEN GO DONE
004: LIST <<A,FILENAME>> ALL
005: REPEAT
006: DONE: DISPLAY End of Report Processing
Bottom.
*--:
```

LOGIN and LOGOUT paragraphs

UniData provides the ability to create a LOGIN paragraph which it executes each time you enter a UniData account. Conversely, you can create a LOGOUT paragraph, which UniData executes each time the user exits the UniData account.

The LOGIN paragraph

The LOGIN paragraph is typically used to set UDT.OPTIONS, set ECLTYPE, set printer characteristics, invoke a menu, and display messages. If a LOGIN paragraph exists, UniData executes it each time the user enters a UniData account.

Note: UDT.OPTIONS 20 determines if UniData executes a LOGIN paragraph when the LOGTO command is executed. If UDT.OPTIONS 20 is off, the LOGIN paragraph is always executed. If UDT.OPTIONS 20 is on, UniData does not execute the LOGIN paragraph when a superuser executes the LOGTO command.

The following sample LOGIN paragraph sets ECLTYPE to U, sets UDT.OPTIONS 19 and 20 ON, and displays “Good Morning” when a user invokes a UniData session.

```
:AE VOC LOGIN
Top of New "LOGIN" in "VOC".
*--: I
001= PA
002= ECLTYPE U
003= UDT.OPTIONS 19 ON
004= UDT.OPTIONS 20 ON
005= DISPLAY Good Morning
*--: FI
Filed "LOGIN" in file "VOC".
:
```

The LOGOUT paragraph

If a LOGOUT paragraph exists in the VOC file, it is executed each time you exit UniData. You can use LOGOUT paragraphs to execute a routine that cleans up application files, exits the application in an orderly manner, and displays messages.

Note: UDT.OPTIONS 74 causes phantom processes to execute the LOGOUT paragraph. If UDT.OPTIONS 74 is on, the phantom process executes the LOGOUT paragraph, otherwise the phantom process does not execute the LOGOUT paragraph.

Chapter 8: The UniData command stack

The UniData command stack stores statements entered from the ECL command line and allows you to recall, edit, reexecute, or save them. If you make a typographical error in a command, the command stack feature allows you to correct the error without having to retype the entire statement.

By default, UniData stores up to 49 commands in the command stack. To change the default, you can set the CSTACKSZ environment variable to save a specified number of command lines. Each command line can contain up to 2720 characters. See Administering UniData on UNIX or Administering UniData on Windows Platforms for information about setting UniData environment variables.

Note: If you change the value of the CSTACKSZ environment variable on Windows platforms, you must restart UniData for the change to take effect.

When you enter a statement on the ECL command line, UniData stores it in position 1 of the stack. As you enter more commands, UniData pushes prior commands in the stack up one position and inserts the current command in position 1 of the stack. When a command moves beyond the stack's limit, it is discarded.

The command stack for each user is saved in a file called .ustk_logname in the directory of the current UniData account. Each time you enter UniData, the stack is recalled.

Command stack functions

Command stack functions recall, change, insert, append, and save commands in the command stack. Command stack functions are always preceded by a single period, and are sometimes referred to as "dot" commands. These commands may be entered in uppercase or lowercase letters.

The following table describes the command stack functions.

Function	Description
.?	Displays online help for the command stack.
.L*n*	Lists the command stack from line 1 through line *n*.
.A*n*	Appends text to the command on line *n*.
.C*n/string1/string2*	Changes *string1* to *string2* in the command on line *n*.
.D*n*	Deletes the command on line *n*.
.I*n*	Inserts a command into the stack on line *n*.
.R*n*	Recalls the command from line *n* to line 1.
.S *name nm*	Saves the command(s) from line *n* through line *m* in the VOC file *name*.
.U*n*	Converts the command at line *n* to uppercase.
.X*n*	Executes the command at line *n*.
.CLEARSTACK	Clears the command stack.

UniData also supplies three command stack functions that operate on items in the VOC file: .D *name*, .L *name*, and .R *name*, where *name* is the name of the VOC record. See Chapter 4, The UniData VOC file, on page 56 for more information about these functions.

Command stack help

You can enter ".?" at the ECL command line to access command stack help, as shown in the following example:

```
:.?
.a# text                Append 'text' to end of sentence "#"
.c#/s1/s2/          Change 's1' to 's2' in sentence '#'
.d#                         Delete sentence number '#' from stack
.d name                 Delete sentence or pargraph 'name' from VOC
.i# any                 Insert 'any' in stack before sentence "#"
.l#                         List '#' lines of stack
.l name                 List paragraph or sentence 'name' from VOC
.r#                         Recalls sentence number '#' to number 1
.r name                 Loads 'name' from VOC into stack
.s name s# e#       Save lines S# thru e# in VOc as name
.x#                         Execute sentence number '#'
:
```

Listing the command stack

To list the commands in your command stack, use the .L function.

Syntax

.L*n*

where *n* is the number of lines of the command stack you want to display. If you do not specify *n*, UniData displays 20 lines of the command stack. The command stack is always displayed with the most recent commands listed from 1 through *n*, as shown in the following example:

```
:.L6
6 LIST INVENTORY PROD_DESC SAMPLE
5 LIST CLIENTS NAME SAMPLE
4 LISTDICT INVENTORY
3 LIST INVENTORY BY COLOR
2 LISTDICT CLIENTS
1 LIST CLIENTS BY CITY CITY SAMPLE
:
```

Appending to a stack command line

To append a character or string to a command line in the stack, use the .A function.

Syntax

.A*n text*

where *n* is the line number in the stack of the command to which you want to append *text*. If *n* is omitted, *text* is appended to the command in position 1 in the stack.

The .A function does not automatically append a space to the last character of the original command. If you want to separate the original command and *text*, you must enter two spaces after .A*n*.

In the following example, .L5 displays 5 lines of the command stack, and the .A function appends LPTR to the first command line.

```
:.L5
5 LIST INVENTORY DESC FEATURES
4 LIST INVENTORY BY DESC BREAK.ON DESC FEATURES
3 LIST INVENTORY FEATURES
2 LIST INVENTORY DESC
1 LIST INVENTORY BY DESC BREAK.ON DESC FEATURES
:
:.A LPTR
LIST INVENTORY BY DESC BREAK.ON DESC FEATURES LPTR
```

Changing a command line

To change the contents of a command line in the stack, use the .C function.

Syntax

.C[*n*]/*old_string*/*new_string*[/G]

where *n* is the position of the stack command you want to change; *old_string* is the text in the original command and *new_string* is the new text. The G (global) option changes every occurrence of *old_string* in the original command. If you omit the stack line position, UniData automatically changes the command line in position 1 in the stack.

Normally, .C*n*, *old_string*, *new_string,* and the G option are separated by a forward slash (/). However, when changing text in the original command line that already contains a forward slash (/), such as a date, you cannot use the forward slash as the delimiter. Any of the following delimiters may be used in the syntax:

! " $ % & ( ) * , - . : = @ [ ] \ _ | { } + ’ ^

In the following example, the date in the original command line is changed, and an asterisk (*) is used as the delimiter:

```
:.L5
5 LIST INVENTORY BY DESC BREAK.ON DESC FEATURES
4 LIST INVENTORY FEATURES
3 LIST INVENTORY DESC
2 LIST INVENTORY BY DESC BREAK.ON DESC FEATURES LPTR
1 LIST INVENTORY WITH INV_DATE < '01/01/95'
:
:.C*01/01/95*08/01/96
LIST INVENTORY WITH INV_DATE < '08/01/96'
```

Deleting a command line

You can delete a command from the stack by using the .D function.

Syntax

.D[*n*]

where *n* is the position in the command stack of the command you want to delete. If you do not enter *n*, the first line of the command stack is deleted. When a command is deleted from the command stack, the remaining commands are moved down one position.

In the following example, the command in position 2 of the command stack is deleted, and the remaining commands move down one position:

```
:.L5
5 LIST INVENTORY BY DESC BREAK.ON DESC FEATURES
4 LIST INVENTORY FEATURES
3 LIST INVENTORY DESC
2 LIST INVENTORY BY DESC BREAK.ON DESC FEATURES LPTR
1 LIST INVENTORY WITH INV_DATE < '08/01/96'
:.D2
:.L5
5 LIST INVENTORY DESC FEATURES
4 LIST INVENTORY BY DESC BREAK.ON DESC FEATURES
3 LIST INVENTORY FEATURES
2 LIST INVENTORY DESC
1 LIST INVENTORY WITH INV_DATE < '08/01/96'
:
```

Note: You can also use the .D function to delete a sentence or paragraph from the VOC file. For more information about deleting these types of VOC records, see The UniData VOC file, on page 56.

Inserting a command line

You can insert a new command to a position in the command stack by using the .I function.

Syntax

.I[*n*] *new_command*

where *n* is the position in the command stack where you want to insert *new_command*. If you do not enter *n*, UniData inserts *new_command* in position 1 of the command stack.

Commands that exist in the command stack in positions greater than or equal to the position where you insert the new command are moved up one position in the command stack. The commands that are below the position where the new command is inserted retain their position.

You can also insert a new command in position 1 of the command stack by entering “?” at the end of the command when you enter it on the ECL command line. Entering “?” prohibits the command from being executed, but adds it to the command stack.

In the following example, a new command is inserted at position 3 in the command stack:

```
:.L5
5 LIST INVENTORY DESC FEATURES
4 LIST INVENTORY BY DESC BREAK.ON DESC FEATURES
3 LIST INVENTORY FEATURES
2 LIST INVENTORY DESC
1 LIST INVENTORY WITH INV_DATE < '08/01/96'
:.I3 COUNT CLIENTS
COUNT CLIENTS
:.L5
5 LIST INVENTORY BY DESC BREAK.ON DESC FEATURES
4 LIST INVENTORY FEATURES
3 COUNT CLIENTS
```

```
 2 LIST INVENTORY DESC
 1 LIST INVENTORY WITH INV_DATE < '08/01/96'
 :
```

The .I function is often used in conjunction with the .S function, which allows you to save a command or series of commands to a record in the VOC file. The .I function inserts commands in a particular order before saving them.

Recalling a command line

To place an existing command line in the first position in the stack, use the .R function. When UniData recalls a command line to position 1, all existing stack entries move up one position. The original command line retains its relative position in the stack. UniData does not execute the recalled command.

Syntax

.R[*n*]

where *n* is the stack position of the command you want to recall to position 1.

In the following example, the command line in position 4 of the command stack is recalled to position 1:

```
:.L5
5 LIST INVENTORY BY DESC BREAK.ON DESC FEATURES
4 LIST INVENTORY FEATURES
3 COUNT CLIENTS
2 LIST INVENTORY DESC
1 LIST INVENTORY WITH INV_DATE < '08/01/96'
:.R4
LIST INVENTORY FEATURES
:.L5
5 LIST INVENTORY FEATURES
4 COUNT CLIENTS
3 LIST INVENTORY DESC
2 LIST INVENTORY WITH INV_DATE < '08/01/96'
1 LIST INVENTORY FEATURES
:
```

Note: You can also use the .R function to recall a sentence or paragraph from the VOC file. For more information about recalling these types of VOC records, see The UniData VOC file, on page 56.

Storing stack commands

To store all or part of the command stack as a record in the VOC file, use the .S function.

Syntax

.S *record_name n m*

where *record_name* is the name of the VOC record where you want to store the command line(s); *n* is the first position in the command stack you want to save; and *m* is the last position in the command stack you want to save. *n* must be greater than or equal to *m*. UniData inclusively saves all commands

between *n* and *m*. If you do not enter command stack positions, UniData saves the command line in position 1of the command stack.

If you save only one command line, UniData creates an S-type (sentence) VOC record. If you save more than one command line, UniData creates a PA-type (paragraph) VOC record.

In the following example, stack positions 5 through 2 are saved as a PA-type VOC record:

```
:.L5
5 LIST INVENTORY FEATURES
4 COUNT CLIENTS
3 LIST INVENTORY DESC
2 LIST INVENTORY WITH INV_DATE < '08/01/96'
1 LIST INVENTORY FEATURES
:.S INV_RPT 5 2
save INV_RPT to VOC.
:
:CT VOC INV_RPT
VOC:
INV_RPT:
PA
LIST INVENTORY FEATURES
COUNT CLIENTS
LIST INVENTORY DESC
LIST INVENTORY WITH INV_DATE < '08/01/96'
:
```

Note: When UDT.OPTIONS 17 is on, UniData disables the .S function.

Changing command lines to uppercase

To convert a command line in the stack to uppercase, use the .U function.

Syntax

.U[*n*]

where *n* is the position of the command line in the stack you want to convert. If you do not enter *n*, UniData converts the command line in position 1.

In the following example, the command line in position 1 of the stack is converted to uppercase:

```
:.L5
5 LIST INVENTORY DESC
4 LIST INVENTORY WITH INV_DATE < '08/01/96'
3 LIST INVENTORY FEATURES
2 CT VOC INV_RPT
1 list inventory desc
:.U
1 LIST INVENTORY DESC
:.L5
5 LIST INVENTORY DESC
4 LIST INVENTORY WITH INV_DATE < '08/01/96'
3 LIST INVENTORY FEATURES
2 CT VOC INV_RPT
1 LIST INVENTORY DESC
:
```

Executing command lines

You can execute a command line in the stack by using the .X function.

Syntax:

.X[*n*]

where *n* is the position of the command line in the stack you want to execute. If you do not enter *n*, UniData executes the command line in position 1 of the stack.

.X*n* recalls the command line to position 1 in the stack, and other command lines in the stack move up one position.

In the following example, UniData executes the command line in position 6 of the stack:

```
:.L8
8 LIST INVENTORY BY DESC BREAK.ON DESC FEATURES
7 LIST INVENTORY FEATURES
6 COUNT CLIENTS
5 LIST INVENTORY DESC
4 LIST INVENTORY WITH INV_DATE < '08/01/96'
3 LIST INVENTORY FEATURES
2 CT VOC INV_RPT
1 LIST INVENTORY DESC
:.X6
COUNT CLIENTS
COUNT CLIENTS
130 record(s) counted.
:
:.L5
5 LIST INVENTORY WITH INV_DATE < '08/01/96'
4 LIST INVENTORY FEATURES
3 CT VOC INV_RPT
2 LIST INVENTORY DESC
1 COUNT CLIENTS
:
```

Clearing the command stack

To clear the command stack, enter the `.CLEARSTACK` command from the ECL prompt. All entries in the stack are cleared, including commands from previous UniData sessions.

Chapter 9: UniData triggers

UniData provides a file management tool, called a trigger, that enables you to set up checks to ensure database integrity. You can use UniData triggers during file update operations and file delete operations. With UniData triggers, you can check data input integrity and impose record-level security.

A UniData trigger is a UniBasic subroutine that UniData calls upon the occurrence of an event. When a UniData file header contains a trigger name, UniData executes the subroutine whenever the file is changed. The events that can trigger a call to the specified subroutine are record updates and deletions. Therefore, you can control any change at the record level by defining an appropriate trigger.

If an operation violates the constraints of a UniData trigger, UniData rejects the operation and may display an error message.

Tip: Because triggers are invoked with every attempt to update the database, we recommend that you minimize the use of triggers that perform time-consuming operations.

Trigger commands

Triggers are composed of the trigger definition, which is stored in the file header, and a globally cataloged UniBasic subroutine that is called before an update or delete is attempted.

Use the `ECL CREATE.TRIGGER` command to define a trigger. If the file header already contains a trigger of the same name, UniData does not overwrite the existing trigger. Instead, you must delete the existing trigger before you can create the new one.

Three ECL commands allow you to create, delete, and list triggers, but do not affect the UniBasic subroutine. The next sections show syntax for these commands. For detailed information about trigger commands, see the *UniData Commands Reference*.

Command	Action
`CREATE.TRIGGER`	Places a trigger name in the file header.
`DELETE.TRIGGER`	Deletes a trigger name from a file header.
`LIST.TRIGGER`	Lists the trigger names in a file header.

Creating a trigger

The `CREATE.TRIGGER` command places a trigger name that calls a UniBasic subroutine. When you attempt to update or delete a file, and UniData encounters an update trigger in the file header, the UniBasic trigger subroutine executes before executing the command.

Syntax

`CREATE.TRIGGER` `[DATA | DICT] filename trigger [BEFORE |AFTER] {UPDATE | DELETE}`

Note: If the AFTER trigger is not defined, then the BEFORE option is assumed.

Deleting a trigger

The DELETE.TRIGGER command deletes a trigger name from a file header. Once the trigger name is deleted, UniData does not call the related trigger subroutine.

Syntax

DELETE.TRIGGER *filename* [DATA | DICT] [BEFORE |AFTER] {UPDATE | DELETE}

Note: To create or delete a trigger, you must be the owner of the data file on which the trigger is being set at the operating system level, or you must have root privileges on UniData for UNIX or Administrator privileges on UniData for Windows Platforms.

Note: If the AFTER trigger is not defined, then the BEFORE option is assumed.

Listing triggers

The LIST.TRIGGER command displays a list of trigger names in a file header.

Syntax

LIST.TRIGGER [DATA | DICT] *filename*

Other ECL commands affected

UniData triggers have an impact on how some ECL commands behave. The following table lists the commands and the behavior when a trigger exists.

Command	Behavior when trigger is present
CLEAR.FILE	You cannot clear a file if a delete trigger is present. UniData displays an informational message if unable to execute CLEAR.FILE due to the presence of trigger.
COPY	UniData displays informational messages if unable to delete a record during execution of a COPY...DELETING statement due to a trigger.
DELETE	UniData displays error and informational messages if unable to delete a record in a file due to a delete trigger. The DELETE command with the ALL keyword executes a CLEAR.FILE. Therefore, when a trigger exists, DELETE...ALL is under the same constraints as CLEAR.FILE.
ED	If UniData cannot write an edited record to a data file due to an update trigger, UniData displays an informational message indicating that the updates were not saved.
MODIFY	If UniData cannot write a modified record to a data file due to an update trigger, UniData displays an informational message indicating that the updates were not saved.

Trigger subroutines

The UniBasic trigger subroutine can contain whatever operation you need to perform. For example, the subroutine could stipulate that certain kinds of record modifications are not allowed for a file, or it might place restrictions on whether users can delete records from a particular file. You can change the underlying trigger subroutine without having to delete and recreate the trigger.

Tip: A trigger subroutine has a specific format according to whether it is an update trigger subroutine or a delete trigger subroutine. For details about creating trigger subroutines, see *Developing UniBasic Applications* or the *UniBasic Command Reference*.

Guidelines for triggers

UniData triggers are governed by the following rules:

- Types — UniData triggers are activated either before or after the update or delete.
- UniBasic subroutines — A UniData trigger is a globally cataloged UniBasic subroutine.
- Writes to hashed files — Write operations are allowed to hashed files within a trigger subroutine.
- Writes to DIR files — You can write to a directory file from within a trigger subroutine. You might want to do so to create a log of trigger operations.
- Nested triggers — A trigger can invoke another trigger.
- Hashed files — You can use triggers with static or dynamic hashed files only. The files can be recoverable or nonrecoverable. You cannot set a trigger on a DIR file or a multilevel directory. You can use triggers for the static or dynamic subfiles of a multilevel file, but not for the multilevel file itself.
- Network File Access (NFA) — You must create the trigger subroutine and the trigger itself on the host, where the hashed file actually resides, and not on the client.

Chapter 10: Null value handling

This chapter introduces UniData's null value handling, which you turn on by setting the UniData configuration parameters NULL_FLAG to 1.

See *Administering UniData on UNIX* or *Administering UniData for Windows Platforms* for information on turning on UniData configuration parameters. Null value handling makes the UniData RDBMS more compliant with the standards defined by ANSI SQL '92. Compliance improves compatibility with client and desktop tools.

For detailed discussions and examples of null value handling for each UniData product, see the manual for that product.

Introduction to the null value

If you turn null value handling on, the null value represents an unknown value, and an empty string ("") represents missing data. If you accept the default language group when installing UniData, the null value is represented by the ASCII character 129. However, if you select an alternative language group, this character can change.

Restrictions — In all UniData products, a single ASCII character (determined by language group) represents the null value. Combined with any other character, including itself, it no longer represents the null value and is not recognized as such by UniData. Also, the null value is not valid for @ID.

Keywords — We recommends that you use keywords to select, insert, and update records with the null value.

Product	Null value keyword
UniData, UniQuery, UniData SQL	IS [NOT] NULLVAL
UniBasic	@NULL You cannot use NOT @NULL to select data values that are not the null value.

Null value handling is turned on with the UniData configuration parameter NULL_FLAG. UniData must be stopped and restarted (with startud) for a udtconfig parameter to take effect. See *Administering UniData on UNIX* or *Administering UniData on Windows Platforms* for instructions on setting the UniData configuration parameters NULL_FLAG and NVLMARK.

Tip: 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.

Accessing UniData through desktop products

Third-party products, such as Microsoft Access and PowerBuilder, access UniData through the Open Database Connectivity (ODBC) standard.

The way these products handle the null value differs from UniData's null value handling. Also, the UniData version of ODBC may, in some cases, change missing values to null characters. See *Developing UniData ODBC Applications* for more information.

Effects on UniData operations

The following table summarizes the effects of the null value on UniData operations in any UniData product with the utdconfig variable NULL_FLAG set to 1.

Operation	Effect
Aggregation	UniData ignores the null value.
Numeric and Date	UniData interprets the null value as 0; arithmetic operations produce a result of the null value.
Conditional tests	Comparisons with the null value produce a negative result; the operation returns 0.
Sorting and indexing	The null value is the smallest value, lower than all negative numbers. It is sorted first in descending sort in the absence of selection criteria. Therefore, null values are sorted first in the alternate key index.
Printing and displaying	The ASCII character that represents the null value is nonprinting. Depending on the product you are using, you can make the null value print or display as another character or sort in the following ways: UniData/UniQuery — The UniData configuration parameter NVLMARK sets a character to print in place of the null value. UniBasic— NVLMARK does not affect UniBasic. You must convert the null value to another character before printing. UniData SQL — NVLMARK does not affect UniData SQL.
Converting	UniBasic — Use a string conversion function (such as SWAP or CHANGE) in a program or virtual attribute to convert the null value to another string. UniData SQL — Use the NVL function to convert the null value to another string.
Selecting	UniQuery/UniData paragraphs — Use IS [NOT] NULLVAL in a WITH or WHEN clause to select records that contain null values. UniData SQL — Use IS [NOT] NULL in a WHERE clause. Because aggregate functions ignore the null value, you cannot use COUNT to count null values. However, you can use COUNT(*) with a WHERE IS NULL clause to count rows containing null values. UniBasic — Use the function ISNV or ISNVS to determine if an entire string or array element is the null value. Because the test [NOT] @NULL always returns a negative result, you cannot directly test for the null value imbedded in a string or array element. To set a pointer in a sorted alternate key index to null value keys, use the UniBasic SETINDEX command with the NULLVAL_ALT_KEY keyword. The character that represents the null value must stand alone to be recognized as the null value by any UniData product. However, you can use the UniBasic functions COUNT, COUNTS, or DCOUNT to count the number of any character, including the null value.

Inserting and removing the null value

With null value handling on, you can insert or remove the null value using the following tools:

- A text editor such as UNIX vi — Enter or remove the ASCII value that represents the null value in your language group.
- The UniData Alternative Editor (AE) — Enter Shift-6 to display nonprinting characters. Then enter or remove the ASCII value that represents the null value in your language group.
- UniEntry — You cannot use UniEntry to insert or remove the null value.
- UniData SQL — You cannot use the INSERT or UPDATE commands to add the null value, along with other values, into a multivalued or multi-subvalued attribute. This is because UniData SQL cannot recognize the keyword NULL embedded in a quoted string such as "`D|NULL|I`". To insert the null value into a multivalued or multi-subvalued attribute that also contains other values, you must first create the record, then use the UPDATE command to change existing values to null, or to append null values. (With null value handling turned off, the keyword NULL inserts an empty string.)
- UniBasic — You can execute a UniBasic program from the colon prompt or call it from a virtual attribute, paragraph, or trigger. Update records or variables with the keyword @NULL.
- UniQuery — You cannot insert or remove the null value with the UniQuery MODIFY command.

When null value handling is turned off

When the UniData configuration parameter NULL_FLAG is set to 0, null value handling is off. The the ASCII character that would represent the null value has the following effects on UniData processing:

- Aggregation Operations — The null value is a non-numeric ASCII character. UniData generates a runtime error indicating "nonnumeric encountered when numeric expected".
- Numeric and Date Calculations — UniData generates a runtime error indicating "nonnumeric encountered when numeric expected."
- Conditional Tests — All characters are tested according to their ASCII value. Programs and paragraphs containing the keywords NULL, IS NULLVAL, or @NULL produce a runtime error. In UniData SQL, with null value handing off, the keyword NULL refers to missing values; for example, SELECT...WHERE f1 IS NULL retrieves missing values.
- Sorting and Indexing — All characters are sorted according to their ASCII value.
- UniBasic Programs — UniBasic programs containing @NULL will not compile. If you attempt to execute object code containing @NULL, the program terminates with a fatal error.

Appendix A: User exits

User exits are special UniData functions that you can use to customize your converted applications in UniData. With user exits, you decide how UniData handles terminal output, return values, and input strings, among other application factors. You can implement user exits through any of the following:

- UniBasic programs
- UniQuery statements that include a virtual attribute that references a user exit
- Procs

This appendix explains how to implement user exits.

Introduction to user exits

UniData provides many user exits, and you can create your own, as well. In addition, you may find that some operations provided by user exits are performed more efficiently by using UniData @variables, UniBasic functions or expressions, or Proc commands.

Tip: For more information about @variables, see For information about UniBasic functions and expressions, see *Developing UniBasic Applications*.

You can implement a user exit in any of the following ways:

- In a UniBasic ICONV or OCONV function
- In a UniData virtual attribute within an ICONV or OCONV function
- In a Proc with parameters

The user exits that UniData provides fall into two categories. User exits that you use with UniBasic subroutines and UniQuery statements are in one category. User exits that you use with Procs are in the other. This chapter handles the categories separately.

Developing UniBasic/UniQuery user exits

You can embed UniData UniBasic/UniQuery user exits in your applications in the following ways:

- Put them in a UniBasic subroutine
- Incorporate them in a virtual attribute definition

You can choose from the following user exits.

User exit	Description
0196	Replace embedded system delimiters in an input string with tildes (~).
01BE	Accept terminal input of a specified number of characters, then output Carriage Return / Line Feed.
11BE	Accept a specified number of characters into a variable and leave the cursor at the current position after the last character.
20E0	Return the current command line.
2193	Cause a page break after the line or attribute is printed.
307A	Sleep until the time specified (in display format).
30E0	Return status of default select list.

User exit	Description
407A	Sleep until the time specified (in internal or display format).
508E	List the contents of a record.
50BB	Return the port and account name.
60BB	Return the account name.
60E0	Return the current page width.
70E0	Turn on display terminal echo.
7201	Return the number of keystrokes stored in the type-ahead buffer.
80E0	Turn off display terminal echo.
81F5	Return the user number.

Inserting user exits in UniBasic programs

In a UniBasic subroutine, the user exit appears in the first line of the subroutine as a return value.

The first line of the UniBasic subroutine must have the following format.

Syntax

SUBROUTINE *subroutine*(*return_val, status, input_val, type*)

The *subroutine* arguments are:

Argument	Description
subroutine	Name of the subroutine.
return_val	String or value the subroutine returns. The value UniData loads into the variable during conversion. The syntax of the user exit statement (*return_val*) within the subroutine looks like this: *return_val* = {ICONV | OCONV}(*input_val*,"U*user_exit*") where *user_exit* is the number of the user exit you are implementing.
status	UniBasic STATUS return value, set to 0 for a successful conversion.
input_val	Input string or value that UniBasic processes.
type	Direction of the conversion: 0 for ICONV and 1 for OCONV. UniBasic automatically returns the correct type.

The following subroutine template contains the elements of a UniBasic subroutine that implements a UniBasic/UniQuery user exit. Subroutines that you write to implement user exits must contain these elements.

```
SUBROUTINE name( return_val, STATUS, input_val, TYPE )
************************************************************
*
* Brief functional description
*
* (c) Copyright notice
************************************************************
*
* Detailed description
*
* Usage, such as return_val = OCONV(input_val,"U0001")
*
```

```
************************************************************
* processing code area:
* Arguments:
* return_val = final data to return.
* STATUS = 0 for success, other for failure.
* input_val= data to be converted or processed.
* TYPE: 0 for ICONV, 1 for OCONV.
*
RETURN
END
```

Tip: For more information about calling a user exit from UniBasic, see *Developing UniBasic Applications*.

Building UniQuery statements with user exits

To use UniQuery to implement user exits, embed the user exit in a virtual attribute definition. You can use user exits in single or multivalued virtual attributes.

Syntax for a singlevalued attribute:

```
{ICONV | OCONV}(input_val,"Uuser_exit")
```

Syntax for a multivalued virtual attribute:

```
SUBR('-{ICONVS | OCONVS}',input_val,"Uuser_exit")
```

You must use the UniData **SUBR** function to execute a user exit against a multivalued virtual attribute. SUBR tells UniData to expect a subroutine that operates on a multivalued attribute. The subroutine is OCONVS or ICONVS.

See Creating virtual attributes, on page 71 for more information about SUBR.

The following example shows a virtual attribute formula that uses U0196:

```
             INVENTORY RECORD ID==>DELIMITER

0 @ID=DELIMITER
1 TYPE=V Replaces delimiters with a tilde.
2 LOC=OCONV(COLOR, "U0196")
3 CONV=
4 NAME=Value Mark
5 FORMAT=25L
6 SM=S
7 ASSOC=
```

The output from a query that uses this attribute follows. Notice that a tilde (~) separates the multivalues, which is what user exit U0196 does:

```
:LIST INVENTORY DELIMITER
LIST INVENTORY DELIMITER 17:28:45 Apr 24 2011 1
INVENTORY. Value Mark..............
    53050 Beige
    56060 Gray
    57030 Gray
    31000 Black
    10140 Black~Silver
    11001 Gray
    10150 Black~Silver
    53040 Beige
```

```
    56070 Red~Blue~Gray~Rose
.
.
.
```

UniBasic/UniQuery user exits reference

0196

0196 replaces embedded system delimiters in the input string with tildes (‘~’).

Syntax

```
X = OCONV(" ", "U0196")
```

01BE

01BE accepts terminal input of a specified number of characters, then issues Return/Line Feed, and returns the input string to the calling program.

Syntax

```
X = OCONV(n, "U01BE")
```

where *n* is the number of characters to accept.

11BE

11ABE accepts a specified number of characters into a variable and leaves the cursor in its current position.

Syntax

```
X = OCONV(n, "U11BE")
```

where *n* is the number of characters to accept.

20E0

20E0 returns the current command line.

Syntax

```
X = OCONV(" ", "U20E0")
```

2193

2193 causes a page break after printing the line, or, with vertical output, after printing the attribute.

Syntax

```
X = OCONV(" ", "U2193")
```

307A

307A causes the program or UniQuery statement to sleep until the specified time (in external form).

Syntax

```
X = OCONV(hh:mm:ss, "U307A")
```

where *hh:mm:ss* represents the hour, minute, second for a process to resume.

30E0

30E0 returns a one if there is an active select list, or a zero if no active select list.

Syntax

```
X = OCONV(" ", "U30E0")
```

407A

407A causes the program or UniQuery statement to sleep until the specific time or for a specified number of seconds.

Syntax

```
X = OCONV(hh:mm:ss, "U407A")
X = OCONV(ss, "U407A")
```

where *hh:mm:ss* represents the hour, minute, and second for a process to resume.

508E

508E lists the contents of a record.

Syntax

```
X = OCONV(data, "U508E")
```

where *data* is @RECORD.

50BB

50BB returns the port and account name.

Syntax

```
X = OCONV(" ", "U50BB")
```

60BB

60BB returns the account name.

Syntax

```
X = OCONV(" ", "U60BB")
```

60E0

60E0 returns the current page width.

Syntax

```
X = OCONV(" ", "U60E0")
```

70E0

70E0 turns on display terminal echo.

Syntax

```
X = OCONV(" ", "U70E0")
```

7201

7201 returns a count of the number of keystrokes stored in the type-ahead buffer.

Syntax

```
X = OCONV(" ", "U7201")
```

80E0

80E0 turns off display terminal echo.

Syntax

```
X = OCONV(" ", "U80E0")
```

81F5

81F5 returns the user number.

Syntax

```
X=OCONV(" ", "U81F5")
```

Developing PROC user exits

You can access one of the UniData PROC user exits in two ways:

- Issue a user exit statement
- Create a UniBasic subroutine

Syntax

```
Uuser_exit
param1 [,param2]...[,]
...
```

Syntax of the first line of a UniBasic subroutine:

```
SUBROUTINE subroutine(proc,cib,pib,sib,ibp,cob,pob,sob)
```

The following table lists subroutine arguments.The *proc* option and all of the input and output buffer arguments are optional.

Argument	Description
subroutine	Name of the subroutine.
proc	Source code of the PROC itself. UniData reads the PROC user exit statement, such as U31AD, then passes it and all subsequent lines as *proc*.
cib	Current input buffer: 0 for primary and 1 for secondary.
pib	Primary input buffer.
sib	Secondary input buffer.
ibp	Input buffer pointer.
cob	Current output buffer: 0 for primary and 1 for secondary.
pob	Primary output buffer.
sob	Secondary output buffer.

You can select from the following PROC user exits.

User exit	Description
0190	Perform arithmetic on reverse Polish string in current output buffer.
0192	Provide output formatting to control spacing of output data.
01A2	Perform an *N*-way branch to a label within the PROC.
01A6	Reposition the cursor to the specified column and row.

User exit	Description
01AD	Extracts a specified attribute from an item in a file.
01B0	Return system date and, time, or both.
11A2	Fill the string in the current input buffer with zeros from the left.
11B0	Return system date and, time, or both.
2196	Return the user's terminal number.
21A2	Delete values from the current buffer.
31AD	Return user's terminal number in file name form.
31B0	Return system date and/or time.
41AD	Force string into current parameter in primary input buffer.
61A2	Toggle output between printer and terminal display.
A1A2	Move the pointer in the primary input buffer back one parameter.

The following template contains the required elements of a subroutine that contains a PROC user exit. Any subroutine that you create to implement a PROC user exit must conform to this template.

```
SUBROUTINE name( PROC, CIB, PIB, SIB, IBP, COB, POB, SOB ) These arguments are optional
************************************************************
*
* Brief functional description
*
* (c) Copyright notice
************************************************************
*
* Detailed description
*
* Usage, such as U0001
*                           target.location
************************************************************
* processing code area:
* Arguments:
*           PROC = the source code of the proc itself
*           CIB  = the current input buffer switch (0 = primary;
*                1 = secondary)
*           PIB  = the primary input buffer
*           SIB  = the secondary input buffer
*           IBP  = the input buffer pointer
*           COB  = the current output buffer switch (0 = primary;
*                1 = secondary)
*           POB  = the primary output buffer
*           SOB  = the secondary output buffer
*
************************************************************
        RETURN
END
```

PROC user exits reference

0190

0190 performs arithmetic functions on a string in the reverse Polish stack current output buffer. UniData computes the arithmetic function on the string, removes the string from the stack, and sends the result to the specified target.

Syntax

U0190
delimiter {T | S | A | P}

The following table lists the 0190 delimiters.

Delimiter	Description
T	Terminal
S	Current output buffer
A	Alternate output buffer
P	Primary input buffer

0192

0192 formats data to control spacing. The data can be a literal or an expression.

Syntax

U0192
... options ...

The following table lists the 0192 options.

Option	Description
X*nn*	Places *nn* blanks in the output.
S*nn*	Places *nn* line feeds in the output.
P	Places a form feed in the output, resets the current page and line counters, and blanks out the heading.
L	Turns the printer on.
H*nn*	Accepts a starting column number and evaluates trailing PQN indirect references.
V*nn file-ref item-ref field-ref*	Pads line to *nn* spaces, derives and prints a value.
**nn file-ref item-ref field-ref*	Pad line to *nn* spaces, apply any conversions present in the dictionary, derive and print a value. *field-ref* is a field description in the dictionary that locates the data and provides a conversion factor.
+	Trailing + suppresses Carriage Return/Line Feed when the line is finally output.

01A2

01A2 performs an *n*-way branch, the equivalent of the UniBasic statement:

ON X GOTO label1[:] [,label2[:] ...]

Syntax

U01A2
```
output_buffer_param value_1 value_2 ... value_n
command 1
command 2
...
```

01A2 controls the flow within the PROC. If the value matches *value_1*, the user exit branches to *label1*, etc.

01A6

01A6 repositions the cursor to a specified column and row.

Syntax

U01A6
```
{(column, row) | (col) | B | C | Xnn | text | + }
```

The following table lists the 01A6 parameters.

Option	Description
column, row	Column and row number where you want to position the cursor.
col	Specifies the column on which to begin printing.
B	Sounds the terminal bell.
C	Clears the screen.
X*nn*	Indicates hexadecimal equivalent of ASCII character *nn*.
text	Specifies the text to be printed.
+	Suppress the carriage return/line feed when the line is output.

Note: This user exit is not allowed if you are attempting to create a custom user exit.

01AD

01AD sends a specified attribute from an item in a file to a specified location.

Syntax

U01AD
```
[*] file item attrib.loc output_loc
error return
success return
```

An asterisk (*) indicates a dictionary file. *output_loc* can be one of the following.

output_loc	Description
T	Terminal
S	Current output buffer
A	Alternate output buffer
P	Primary output buffer

Note: This user exit is not allowed if you are attempting to create a custom user exit.

01B0

01B0 returns system date or time, or both.

Syntax

U01B0
`{T | D | TD} {T | S | P | Wid}`
error return
success return

The following tables lists the 01B0 options.

Option	Description
T	System time. Use the following: U01B0 - Milliseconds since midnight. U11B0 - Current time formatted *hh:mm:ss*. U31B0 - Milliseconds since midnight in hexadecimal.
D	System date. Use the following: U01B0 - Number of days since 31 DEC 1967. U11B0 - Current date formatted *dd mm yy*. U31B0 - Number of days since 31 DEC 1967 in hexadecimal.
TD	System time and date. Use the following: U01B0 - Time and date separated by quotation marks. U11B0 - Time and date separated by two spaces. U31B0 - Time and date results separated by quotation marks.
T	Terminal
S	Current output buffer.
P	Primary input buffer.
W*id*	Write in file *nn*P, where *nn* is the user number of the item specified by *id*.

11A2

11A2 fills the string in the current input buffer with zeros to the left of the data, until it reaches the specified length. It returns the result to the current input buffer.

Syntax

U11A2
length

11B0

11B0 returns system date or time, or both.

Syntax

U11B0
{T | D | TD} {T | S | P | *Wid*}
error return
success return

The following table lists the 11B0 options.

Option	Description
T	System time. Use the following: U01B0 - Milliseconds since midnight. U11B0 - Current time formatted hh:mm:ss. U31B0 - Milliseconds since midnight in hexadecimal.
D	System date. Use the following: U01B0 - Number of days since 31 DEC 1967. U11B0 - Current date formatted dd mm yy. U31B0 - Number of days since 31 DEC 1967 in hexadecimal
TD	System time and date. Use the following: U01B0 - The time and date results separated by quotes. U11B0 - The time and date separated by two spaces. U31B0 - The time and date results separated by quotes.
T	Terminal
S	Current output buffer.
P	Primary input buffer.
W*id*	Write in file *nn*P, where *nn* is the user number of the item specified by *id*.

2196

2196 lists the user's terminal number at the specified output location.

Syntax

U2196
{T | P | S | A}
normal return

The following table lists the 2196 options.

Option	Description
T	Terminal
P	Primary input buffer
S	Current output buffer
A	Alternate output buffer

21A2

21A2 deletes values from the current output buffer. If you specify the option as 0, all values are deleted. If you specify 1, the last value is deleted.

31AD

31AD returns the user's terminal number in file name form (with "P" appended) to the specified output location.

Syntax

U31AD
{T | P | S | A}

Option	Description
T	Terminal
P	Primary input buffer
S	Current output buffer
A	Alternate output buffer

Note: This user exit is not allowed if you are attempting to create a custom user exit.

31B0

31B0 returns system date or time, or both.

Syntax

U31B0
{T | D | TD} {T | S | P | *Wid*}
error return
success return

Option	Description
T	System time. Use the following: U01B0 - Milliseconds since midnight. U11B0 - Current time formatted *hh:mm:ss*. U31B0 - Milliseconds since midnight in hexadecimal.
D	System date. Use the following: U01B0 - Number of days since 31 DEC 1967. U11B0 - Current date formatted *dd mm yy*. U31B0 - Number of days since 31 DEC 1967 in hexadecimal
TD	System time and date. Use the following: U01B0 - The time and date results separated by quotation marks. U11B0 - The time and date separated by two spaces. U31B0 - The time and date results separated by quotation marks.
T	Terminal
S	Current output buffer
P	Primary input buffer
W*id*	Write in file *nn*P, where *nn* is the user number of the item specified by *id*.

41A D

41AD replaces the current item in the primary input buffer with a specified string(s). You may enter multiple items by separating the items with spaces. UniData converts imbedded spaces to attribute marks.

Syntax

```
U41AD
string1 string2 ...
```

61A2

61A2 toggles the output destination between the display terminal and the printer.

A1A2

A1A2 moves the pointer in the primary input buffer back one parameter.

Appendix B: Using UniData MENUS

This chapter describes the UniData MENUS utility that enables you to create menus to accomplish UniData tasks. With UniData MENUS, you can transform the UniData command line interface into a menu-driven interface.

Introduction to UniData MENUS

A menu in UniData is a front-end tool that transforms the UniData command line interface into a menu-driven user interface. UniData MENUS lets the user execute a UniData command from a menu, rather than at the colon prompt. The main menu of the MENUS utility is an example of the kind of menus you can create:

```
        MENU Maintenance                13:03:57 Mar 29 2011
 1= Enter/Modify         a MENU
 2= Enter/Modify         a formatted MENU
 3= Display              a summary of all MENUs on a MENU file
 4= Display              the contents of a MENU
 5= Enter/Modify         a VOC MENU selector
 6= Enter/Modify         a VOC stored sentence item
 7= Display              all MENU selector item on the VOC file
 8= Display              all stored sentence items on the VOC file
 9= Display              the dictionary of a file
 10= Print               a summary of all MENUs on a MENU file
 11= Print               the contentes of a MENU
 12= Print               the dictionary of a file
 13= Enter/Modify        a VOC stored paragraph item

 which would you like? (1 - 13)=
```

From a menu, you can execute a UniData SQL statement, a VOC sentence or paragraph, or a UniBasic program, or another menu. You can use a menu to call data entry programs and reports. Also, a UniData menu can include a line of help text.

You can use a menu to support database security. For example, you can provide menus for users to access data entry programs and reports, while denying them access to the UniData prompt.

The MENUS utility also allows you to create, modify, and display sentences and paragraphs in the VOC file. See UniData paragraphs, on page 102 for more information about VOC sentences and paragraphs.

Note: You can also create and modify menus using UniEntry or the Alternate Editor (AE), but the UniData MENUS utility can make the process easier.

Elements of UniData Menus

UniData menus are records in a menu file. Menus have an entry (type M) in the VOC file.

MENUFILE file

When you create a new menu, UniData stores it as a record in MENUFILE. MENUFILE is a UniData static hashed file, an F-type record in the VOC file.

UniData creates MENUFILE whenever you create a new UniData account with the system-level newacct command. You can create a different UniData file to store your menus by using the ECL CREATE.FILE command. (See UniData file types, on page 46 or the *UniData Commands Reference* for information about the CREATE.FILE command.

The following example shows a VOC record for the MENUS utility main menu, which is a formatted menu:

```
                    VOC RECORD ID==>MENUFILE
         0 @ID=MENUFILE
         1 F1=F
         2 F2=MENUFILE
         3 F3=D_MENUFILE
.
.
.
     29 F29=
```

MENUFILE dictionary

The MENUFILE dictionary looks like this:

```
:LISTDICT MENUFILE
SORT DICT MENUFILE TYP LOC CONV MNAME FORMAT SM ASSOC BY TYP BY
LOC BY @ID 13:21:20 Mar 29 2011 1
@ID............ TYP LOC.......... CONV MNAME.......... FORMAT SM
ASSOC.....

@ID             D               0      MENUPTR          10L    S
TITLE           D               1                       40L    S
DESCRIPTION     D               2                       15L    M
ASSOC1
COMMAND         D               3                       15L    M
ASSOC1
EXPLANATION     D               4                       20L    M
ASSOC1
PROMPTS         D               5                       15L    S
EXITS           D               6                       10L    S
STOPS           D               7                       8L     S
ASSOC1          PH  DESCRIPTION C
                    OMMAND EXPLAN
                    ATION
9 records listed
```

VOC menu records

Each menu must have a VOC record. The VOC record acts as a pointer to the menu. Menu records are always type M in the VOC file. A VOC record for a menu looks like this:

```
        VOC RECORD ID==>formattedmenu1
0 @ID=formattedmenu1
1 F1=M A test formatted menu
2 F2=MENUFILE
3 F3=formattedmenu1
```

Types of UniData Mmenus

UniData enables you to create two types of menus that differ only in format:

- Unformatted — UniData places the options and the prompt on the screen automatically using a default menu setup.
- Formatted — you stipulate the screen locations for the options and the prompt by designating a screen column and line numbers for the start of each line of menu text.

Note: Once you create a menu, you cannot change the format type for it.

Structure of a UniData menu

When you create a UniData menu, you control its elements, including the following:

- Layout — formatting of menu selections and menu prompts in columns and rows
- Action — how UniData responds when the user makes a selection
- Prompt messages — help text content
- Exit and stop commands — how users exit your menus

Formatted and unformatted menus

The MENUS utility provides two kinds of menu layouts. You decide the layout for the menu as the first step to creating it.

The Steps for creating a UniData menu, on page 148 section provides detailed instructions for creating both kinds of UniData menus.

- Formatted menu — UniData centers the title at the top of the menu, but you have the option of selecting the display columns for all menu options and the menu prompts:

```
A Test Formatted Menu           18:54:40 Nov 13 2011

            1 Formatted option one
            2 Formatted option two

    Enter the option number and a question mark for help; "q"
to quit; "x" to exit; "a" to abort
```

- Unformatted menu — UniData creates a default layout for the menu options by centering the menu title, placing the option number and option description in adjacent columns beginning at column 1, and placing the prompt text at the bottom of the screen starting at the leftmost column:

```
            This Is A Test Menu             18:43:47 Nov 13 2011

            1 option 1
            2 option 2
            3 option 3
Enter option number and a question mark for help; "Q" or "q"
to quit; "X" or "x" to exit; "A" or "a" to abort
```

Menu file dictionary records

Each menu file record consists of the attributes listed in the following table. When you create a menu, you designate the values for each attribute.

Detailed instructions and examples illustrate how to determine these attribute values in Steps for creating a UniData menu, on page 148.

Attribute	Description
Attribute — 1	Menu title; can be up to 55 characters. The MENUS utility centers the title and displays the current time and date for both unformatted and formatted menus.
Attribute — 2	Multivalued list of selection descriptions. A description is the text that appears on the screen for a menu option. If the menu is formatted, and if you specified an option position, then the column and row positions must precede the option text.
Attribute — 3	Multivalued list of selection actions. These correspond in a one-to-one manner with the descriptions in Attribute 2. In general, any command valid at the ECL prompt is valid here. After the action has been performed, UniData returns you to the menu for another selection. If this attribute is left blank, UniData assumes the associated text in Attribute 2 is only text and does NOT display that text as a menu option.
Attribute — 4	Multivalued list of online HELP messages for each of the options in Attribute 2. Users can display these messages by typing the appropriate selection number followed by a question mark (?).
Attribute — 5	Menu prompt text and position. If you leave the prompt blank, UniData uses the default message: “Which would you like?” If the menu is formatted, and if you specified a prompt position, then the column and row positions must precede the selection text. If you enter a cursor position, but not a prompt, then UniData uses the default prompt at the specified position.
Attribute — 6	List of valid exit commands. Each exit command returns to the level from which you invoke the menu. If you do not stipulate an exit command, UniData interprets a carriage return as the exit command.
Attribute — 7	List of valid stop commands. The stop commands abort the menu process and return to the colon prompt. UniData does not have a default stop command within the MENUS utility.

Note: Attributes 2, 3, and 4 are associated. Together, they make up a single menu selection. This selection displays as option 6 DESC ACTION EXP when you select the options to create formatted or unformatted menus.

Displaying menu attributes

You can view the contents of menu attributes through UniEntry, the AE line editor, or UniQuery statements.

The following example shows how TEST_MENU1 looks when you display it with AE.

```
:AE MENUFILE TEST_MENU1
Top of "TEST_MENU1" in "MENUFILE", 7 lines, 266 characters.
*--: p
001: Unformatted Test Menu
002: Clients RecordsyInventory RecordsyOrders Records
003: LIST CLIENTS NAME CITY COUNTRY}LIST INVENTORY PROD_NAME PRICE}LIST ORDERS
004: }}
005: To make a selection, enter the option number.
To return to the UniData colon prompt, enter "Q" or "q".
```

```
006: Q,q
007: ABORT,abort
Bottom.
*--:
```

Accessing the MENUS utility

To access the UniData MENUS utility, enter MENUS at the UniData colon prompt. UniData displays the MENU Maintenance main menu. From this menu, you begin the menu creation process. The MENU Maintenance menu looks like this:

```
            MENU Maintenance                11:55:50 Apr 04 2011
1= Enter/Modify         a MENU
2= Enter/Modify         a formatted MENU
3= Display              a summary of all MENUs on a MENU file
4= Display              the contents of a MENU
5= Enter/Modify         a VOC MENU selector
6= Enter/Modify         a VOC stored sentence item
7= Display              all MENU selector item on the VOC file
8= Display              all stored sentence items on the VOC file
9= Display              the dictionary of a file
10= Print               a summary of all MENUs on a MENU file
11= Print               the contentes of a MENU
12= Print               the dictionary of a file
13= Enter/Modify        a VOC stored paragraph item

which would you like? (1 - 13)=
```

The following table lists and describes the MENU Maintenance options.

Option	Description
1: Enter/Modify a MENU	This option builds a new menu or modifies an existing one. The menu you create or modify with this option is not formatted. UniData uses default positions for the selection options and the selection prompt.
2: Enter/Modify a Formatted MENU	This option is the same as Option 1, except that you create or modify a formatted menu. You assign the positions on the screen for the option text and the selection prompt.
3: Display a Summary of all MENUs on a MENU File	This option prompts for the name of a menu file, and then displays the menu records (M type records) in that file.
4: Display the Contents of a MENU	This option prompts for the names of a menu file and menu record, and then displays the contents of the record.
5: Enter/Modify a VOC MENU Selector	This option builds a VOC menu record so that users can run the menu by typing the menu name at the ECL prompt. You cannot run any menu you create until you create a record in the VOC file that points to that menu.
6: Enter/Modify a VOC Stored Sentence Item	This option builds a VOC record or UniQuery sentence. MENUS prompts you for each attribute in the sentence.
7: Display all MENU Selector items on the VOC File	This option functions like the VOC command LISTM and lists all the menu pointers (M type records) currently defined in the VOC file.

Option	Description
8: Display all Stored Sentence Items on the VOC File	This option functions like the VOC command LISTS and lists all the sentences (S type records) currently defined in the VOC file.
9: Display the Dictionary of a File	This option functions like the VOC command LISTDICT *filename* and displays the dictionary of an item.
10: Print a Summary of all MENUs on a MENU File	This option is similar to Option 3, except that it directs the output to the default printer.
11: Print the Contents of a MENU	This option is similar to Option 4, except that it directs the output to the default printer.
12: Print the Dictionary of a File	This option is similar to Option 9, except that it directs the output to the default printer.
13: Enter/Modify a VOC Stored Paragraph Item	This option builds a paragraph record in the VOC file. Option 13 is similar to Option 6, except that the item is a paragraph rather than a sentence.

Quitting the MENUS utility

To exit the MENUS utility from the MENU Maintenance menu, press `ENTER` without selecting any options.

To Exit a MENUS utility selection

To exit selections on the menu, you have several options:

- Use the exit option that appears at the bottom of the screen.
- Enter `q` or `Q` to quit, and then press `ENTER` twice.
- Press `ENTER` until you return to the main menu.

Steps for creating a UniData menu

This section provides instructions for creating UniData menus. Although there are two kinds of UniData menus, the procedure you follow is essentially the same for both kinds.

- Decide where to store the menu
- Establish the menu identification parameters
- Define the menu attributes
- Create a VOC record for the menu

Decide where to store the menu

In order to create any menu, you must have a file to store it.

A menu is a record in a menu file, and the first step to creating a menu is to identify the name of the menu file. UniData provides a default menu file called MENUFILE. If you want to use another menu file, you can create it using the ECL **CREATE.FILE** command.

Tip: See UniData file types, on page 46 or the *UniData Commands Reference* for more information about the `CREATE.FILE` command.

Establish the menu identification parameters

In this step you must decide whether to create a formatted or an unformatted menu; select the menu file where the menu will reside; pick a name for the menu; and decide on a title for the menu.

- At the **MENU Maintenance** menu, for an unformatted menu enter `1` to select **1=Enter/Modify a MENU**. To create a formatted menu, enter `2` for **Enter/Modify a formatted menu**. UniData displays the `FILE NAME` prompt.
- Enter the name of the file where you intend to install the menu (the default is MENUFILE). UniData displays the "`id(MENU NAME)`" prompt.
- The ID is the name in the menu file. This becomes the menu record @ID. UniData displays the `Title` prompt.
- Enter the menu title. For both formatted and unformatted menus, this text appears centered at the top of the menu with the time and date. Press `RETURN`. UniData displays the Desc prompt.

The following example illustrates the creation of a new menu through this step. The boldface type indicates user input:

```
                MENU Maintenance            17:12:32 Mar 29 2011
 1= Enter/Modify             a MENU
 2= Enter/Modify             a formatted MENU
 3= Display                  a summary of all MENUs on a MENU file
 4= Display                  the contents of a MENU
 5= Enter/Modify             a VOC MENU selector
 6= Enter/Modify             a VOC stored sentence item
 7= Display                  all MENU selector item on the VOC file
 8= Display                  all stored sentence items on the VOC file
 9= Display                  the dictionary of a file
 10= Print                   a summary of all MENUs on a MENU file
 11= Print                   the contentes of a MENU
 12= Print                   the dictionary of a file
 13= Enter/Modify            a VOC stored paragraph item

 which would you like? (1 - 13)=1
 FILE NAME=MENUFILE
 id(MENU NAME)=TEAMS
 Title=Company Sports Teams
 Desc=
```

Define the menu attributes

At this point you begin listing the menu options for the new menu.

When you create a menu, whether formatted or unformatted, you must define the menu options in the order you want them to appear on the screen. UniData automatically numbers menu options based on this order. Thus, the first option you define is numbered 1 on the menu, the second option you define is numbered 2, and so on.

If you are creating a formatted menu, this is where you also have an opportunity to assign the column and row layout for the menu options and the prompt text. You do not need to do that for an unformatted menu; UniData handles this task.

Create the menu selection options

Desc is the description text that appears on the screen next to the option number.

(Remember, the MENUS utility automatically assigns sequential option numbers.) Enter the description text, and press ENTER. If this is an unformatted menu, UniData displays the Action prompt.

Formatted menu

If this is a formatted menu, UniData displays the Display column prompt.

At this prompt, enter the column number where you want the description to begin and press ENTER. Valid values depend on the number of columns on your terminal screen. UniData displays the Display line prompt. Enter the row where you want the option to appear and press ENTER. Valid values depend on the number of lines on your terminal screen. UniData displays the Action prompt.

Creating double columns

You can create double columns by creating a group of options for each column, one column at a time.

Start with the first column, then proceed to the next column and create the option group.

- At the Action prompt, enter the action you want the selection to accomplish. This action can be a VOC sentence, a UniData command, a UniData SQL statement, a UniBasic program, or any command that you can enter at the ECL prompt (:). When you press ENTER, UniData displays the "Exp" prompt.

  Note: Set UDT.OPTIONS 37 to on to cause UniData to pause after displaying the text on the screen. With this option on, UniData keeps the display on the screen and prompts for a carriage return, "Q", or "q" to return to the menu. If this option set to off (the default setting), UniData displays the results of the option to the screen and then immediately returns to the menu.

- Enter an explanation of the menu option. This text does not appear when you display the menu, rather, it is help text that the user can invoke by entering the option number and a question mark (n?). You may enter up to 80 characters of help text. If you do not want to add an explanation, press ENTER.

When you finish entering the help text, press ENTER. UniData displays the Desc prompt again. This completes the definition of one menu selection for your new menu.

The next example shows a menu with several selection options defined:

```
FILE NAME=MENUFILE
id(MENU NAME)=TEAMS
```

```
Title=Company Sports Teams
Desc=UNISERVERS Volleyball
Action=LIST PLAYERS NAME PHONE DEPT WITH TEAM = "UNISERVERS"
Exp=
Desc=SCREAMERS Softball
Action=LIST PLAYERS NAME PHONE DEPT WITH TEAM = "SCREAMERS"
Exp=
Desc=RICOCHETS Squash
Action=LIST PLAYERS NAME PHONE DEPT WITH TEAM = "RICOCHETS"
Exp=
Desc=
Prompt=
```

To end the process of defining options to the menu, press `ENTER`. If this is an unformatted menu, UniData displays the Prompt prompt.

If this is a formatted menu, UniData displays the Prompt column prompt. Enter the column number where you want prompt text to begin and press `ENTER`. UniData displays the Prompt line prompt. Enter the row where you want the help text to begin and press `ENTER`. UniData displays the `Prompt` prompt.

Create the prompt option

You can add prompt text to your menu. To tell the user which keys to press to select a menu option, get help, or exit the menu.

- At the `Prompt` prompt, enter the text you want to use for the selection prompt. In an unformatted menu, this text begins in column 1 and appears at the bottom of the screen. In a formatted menu, this text begins at the column and row you designated previously.

We recommend that you always include two features in the prompt text:

- The input for users to quit the menu, emphasized by single or double quotation marks (for example, “EX” or “ex”).
- A space at the end of the text to separate it from user response.

The next example shows prompt text for a sample menu:

```
FILE NAME=MENUFILE
id(MENU NAME)=TEAMS
Title=Company Sports Teams
Desc=UNISERVERS Volleyball
Action=LIST PLAYERS NAME PHONE DEPT WITH TEAM = "UNISERVERS"
Exp=
Desc=SCREAMERS Softball
Action=LIST PLAYERS NAME PHONE DEPT WITH TEAM = "SCREAMERS"
Exp=
Desc=RICOCHETS Squash
Action=LIST PLAYERS NAME PHONE DEPT WITH TEAM = "RICOCHETS"
Exp=
Desc=
Prompt=Enter a number to select any option, ? to get help,
"EX" or "ex" to return to the main menu.
1 MENU NAME=TEAMS
2 TITLE=Company Sports Teams
3 PROMPT=Enter a number to select any option, ? to get help,
"EX" or "ex" to return to the main menu.
4 EXITS=
5 STOPS=
6 DESC ACTION EXP
CHANGE=
```

After you enter the prompt text and press ENTER, UniData does one of two things:

- Unformatted menu — displays the CHANGE prompt. At this prompt, choose **4** to select EXITS. UniData displays the EXITS prompt.
- Formatted menu — displays the Exits prompt. Proceed to step 5.

At the EXITS or exits prompt, enter the options to exit the menu, separating each with a comma but not a space. These should match the exit options in the prompt text. When a user uses these options to exit a menu, UniData returns to the level where the user accessed the menu. Enter the exit options in both upper case and lower case letters to provide users with more flexibility.

When you press ENTER, UniData returns to the CHANGE prompt.

- Enter 5 to select STOPS. UniData displays the STOPS prompt. Enter the options to stop a menu, separating each with a comma but not a space. After you enter the stop options, UniData displays the CHANGE prompt and the attributes of the current menu. Enter the stop options in both uppercase and lowercase letters to provide users with more flexibility. When a user enters these options to stop a menu, UniData returns to the colon prompt.

The following example shows a menu with exits and stops defined:

```
1 MENU NAME=TEAMS
2 TITLE=Company Sports Teams
3 PROMPT=Enter a number to select any option, ? to get help,
 "EX" or "ex" to return to the main menu.
4 EXITS=EX,ex
5 STOPS=ABORT,abort,QUIT,quit
6 DESC ACTION EXP
CHANGE=
```

At this point, your menu is complete. You cannot run it, however, until you create a record in the VOC file that points to the menu. To do so, proceed to the next step.

If you want to change any of the menu attribute definitions, see Modifying a menu, on page 154.

Create the VOC record

You cannot run a menu unless it has a record in the VOC file.

The VOC record must exist in the account where you want to run the menu. Follow the next steps to create the menu VOC record. The examples create a VOC record for the TEAMS menu created in previous steps.

- From the **MENU Maintenance** menu select **5, Enter/Modify a VOC MENU** selector. At the prompt for menu name, enter the name of the menu you created and press ENTER:

```
            MENU Maintenance            16:33:42 Apr 01 2011
 1= Enter/Modify         a MENU
 2= Enter/Modify         a formatted MENU
 3= Display              a summary of all MENUs on a MENU file
 4= Display              the contents of a MENU
 5= Enter/Modify         a VOC MENU selector
 6= Enter/Modify         a VOC stored sentence item
 7= Display              all MENU selector item on the VOC file
 8= Display              all stored sentence items on the VOC file
 9= Display              the dictionary of a file
 10= Print               a summary of all MENUs on a MENU file
 11= Print               the contentes of a MENU
 12= Print               the dictionary of a file
 13= Enter/Modify        a VOC stored paragraph item
```

```
which would you like? (1 - 13)=5
MENU NAME=TEAMS
DESC=
```

- At the DESC prompt, you have the option of entering text to describe the purpose of the menu. If you do not want to enter a description, press ENTER.
- The next prompt is for the menu file name. This is the file where the menu is stored. The TEAMS menu is stored in UniData's default menu file, MENUFILE. Enter the menu file name and press ENTER:

```
which would you like? (1 - 13)=5
MENU NAME=TEAMS
DESC=
MENU FILE NAME=MENUFILE
RECORD NAME IN MENU FILE=
```

- UniData prompts for the record name in the menu file you named at the last prompt. Enter the name of the menu that you created, and press ENTER:

```
which would you like? (1 - 13)=5
MENU NAME=TEAMS
DESC=
MENU FILE NAME=MENUFILE
RECORD NAME IN MENU FILE=TEAMS
    1 MENU NAME=TEAMS
    2 TYPE=M
    3 DESC=
    4 MENU FILE NAME=MENUFILE
    5 RECORD NAME IN MENU FILE=TEAMS
CHANGE=
```

This completes creation of the VOC record for your menu. If you want to change anything in the VOC record, enter the attribute number next to the item, press ENTER, and type in the changed information. UniData deletes the current information and replaces it with the new information.

Running a menu

To run the menu, at the ECL colon prompt enter the menu name.

Syntax

```
menu_name
```

The following example illustrates the TEAMS menu:

```
:TEAMS
        Company Sports Teams    17:43:10 Apr 01 2011
        1 UNISERVERS Volleyball
        2 SCREAMERS Softball
        3 RICOCHETS Squash
Enter a number to select any option, ? to get help, "EX" or "ex"
to return to the main menu.
```

Displaying a VOC menu record

You can use UniEntry, AE, or the MENUS utility to display the VOC record.

Displaying a VOC menu record with the MENUS utility

To use the MENUS utility to display a VOC menu record, enter the MENUS command at the colon prompt.

At the main menu, select **7, Display all MENU** selector items on the VOC file. UniData displays the menu record:

```
:MENUS
            MENU Maintenance            17:40:21 Apr 01 2011
1= Enter/Modify             a MENU
2= Enter/Modify             a formatted MENU
3= Display                  a summary of all MENUs on a MENU file
4= Display                  the contents of a MENU
5= Enter/Modify             a VOC MENU selector
6= Enter/Modify             a VOC stored sentence item
7= Display                  all MENU selector item on the VOC file
8= Display                  all stored sentence items on the VOC file
9= Display                  the dictionary of a file
10= Print                   a summary of all MENUs on a MENU file
11= Print                   the contentes of a MENU
12= Print                   the dictionary of a file
13= Enter/Modify            a VOC stored paragraph item

which would you like? (1 - 13)=7
File Name = VOC
name =TEAMS
type+description=M
menu file name =MENUFILE
menu name =TEAMS

Enter return to continue
```

Modifying a menu

You can use UniEntry, the AE line editor, or the MENUS utility to modify a menu. This section explains how to use the MENUS utility to modify menus.

1. From the **MENUS Maintenance** menu, enter option **1, Enter/Modify a MENU** or option **2, Enter/Modify a Formatted Menu**, depending on the format of the menu. UniData prompts you for the menu file name. If you do not remember the menu format, you can display the attributes for the menu record with a UniQuery statement. In the following example, notice that the Description attribute displays a pair of numbers in parentheses that precedes each menu option description. This pair of numbers is the column and row format for the option. Only formatted menus display the column and row formats.

   ```
   :LIST MENUFILE DESCRIPTION WITH @ID = "formattedmenu1"
   LIST MENUFILE DESCRIPTION WITH @ID = "formattedmenu1" 16:50:41 Nov 14 2011 1
   MENUPTR... DESCRIPTION....

   formattedm (15,10)option one
   enu1
           (15,11)option two
   1 record listed
   ```

2. Enter the name of the menu file where the menu resides, UniData prompts for the name of the menu, the @ID. Enter the menu name to get a display like the following for the menu you plan to modify. UniData displays the CHANGE prompt.

   ```
   FILE NAME=MENUFILE
   ```

```
id(MENU NAME)=TEAMS
  1 MENU NAME=TEAMS
  2 TITLE=Company Sports Teams
  3 PROMPT=Enter a number to select any option, ? to get help,
 "EX" or "ex" to return to the main menu.
  4 EXITS=EX,ex
  5 STOPS=ABORT,abort,QUIT,quit
  6 DESC ACTION EXP
CHANGE=
```

3. At the prompt, enter the number of the attribute you want to change and press ENTER. UniData deletes the value in the attribute and prompts for new text. Enter the new text and press RETURN. UniData returns to the CHANGE prompt.
   To continue making changes to other attributes, repeat step 3. Otherwise, at the CHANGE prompt, press ENTER to return to the MENU Maintenance menu.
   When you modify the @ID, the menu name, UniData creates a copy of the menu so that the same menu exists under two names. To delete the original menu record, use UniEntry, the AE editor, or the UniData ECL DELETE command, as you would for a record in any UniData file.

Modifying a multivalued attribute in a menu

The MENUFILE dictionary contains three multivalued attributes:

- DESCRIPTION
- COMMAND
- EXPLANATION

These multivalued attributes are associated to make up Attribute 6, DESC ACTION EXP, of the menus you create. The following table lists the MENUFILE multivalued attributes.

Multivalued attribute	Description
DESCRIPTION	DESC — Displays as a menu selection.
COMMAND	ACTION — A sentence or paragraph stored in the VOC record, a UniData SQL statement, or UniBasic program, or any command you can enter at the ECL colon prompt.
EXPLANATION	EXP — Explanatory help text that users can access by entering the option number followed by a question mark (*n*?).

You can use UniEntry to modify multivalued menu items. If you do, it will help you to know the following relationships.

MENUS Utility	UniEntry
DESC	Column 1, DESCRIPTION
ACTION	Column 2, COMMAND
EXP	Column 3, EXPLANATION

1. To modify the associated attribute, Attribute 6 of your menu, start from the **MENUS Maintenance** menu and select 1 or 2, depending on whether the menu you plan to modify is formatted or unformatted.
2. Enter the name of the menu file where the menu resides. If you enter an incorrect file name, UniData displays an error message and returns to the **MENU Maintenance** menu prompt. When you enter a file name, UniData prompts for the menu @ID (menu name).

3. Enter the name of the menu you want to modify. UniData displays the menu record and the `CHANGE` prompt. If you enter an @ID that does not exist in the menu file, UniData assumes that you are creating a new menu and prompts for the menu attribute information.
4. Enter the number of the attribute you want to change; for the multivalued attribute, enter `6` to select DESC ACTION EXP. UniData displays all selections for the attribute, including the option number and description, the action, and the explanation for the option.
5. At the bottom of your screen UniData prompts for the following:
   - (D)elete — delete one of the existing menu options
   - (I)nsert — insert a new menu option
   - line number change — modify an existing menu option

Delete an option

Enter `D` to delete a menu option and press `ENTER`. At the prompt, enter the line number of the option you want to delete and press `ENTER`. UniData deletes the description, action, and explanation for that option.

Insert a new option

Enter I to insert a new option and press `ENTER`. At the prompt, enter the line number of the line that is below where you want to insert the new option. For instance, if you want to insert a new option between the first and second options, enter `2`. UniData prompts for the description, action, and explanation for the new option. If you are modifying a formatted menu, you must precede the text with the format column and row numbers (*col,row*).

Change a line

If you entered a line number to modify an option, UniData prompts for all the option attributes beginning with the description. When you choose the line number selection, UniData erases the current entry for each attribute of the option, therefore, you must reenter the information.

Tip: To avoid having to reenter the attribute information for a multivalued attribute, make line changes with UniEntry. UniEntry allows you to change values on a column-by-column basis.

Viewing menu records in a menu file

There are at least four tools for displaying menu records in a MENUFILE:

- UniEntry
- AE (Alternate Editor)
- Query statements
- MENUS utility

This section explains how to display menu records using the MENUS utility.

Tip: A UniData menu file is a hashed file. Therefore, you can use UniEntry, the AE editor, and query tools to view menu records, as you would for any UniData record. For information about using query tools, see *Using UniQuery* and *Using UniData SQL*.

From the `MENU Maintenance` menu you can display various aspects of a menu, or you can print it. The MENUS utility provides options to display the following menu aspects:

- Summary of all menus on a menu file
- Contents of a menu
- VOC record for a menu and selected attributes of the VOC record
- Menu file dictionary

The MENUS utility allows you to print the following output:

- Summary of all menus on a menu file
- Contents of a menu
- Menu file dictionary

Note: UDT.OPTIONS 37 determines whether UniData clears the screen after displaying a screen or retains the display and prompts the to press `ENTER` to continue.

Displaying menu records

From the `MENU Maintenance` menu, use options 3, 4, 7, 8, and 9 to display information about menus. When you select any of these options, UniData prompts for the name of the menu file where the menu is stored. If necessary, UniData then prompts for the menu name (the @ID).

```
                MENU Maintenance 17:49:24 Apr 01 2011
 1= Enter/Modify                a MENU
 2= Enter/Modify                a formatted MENU
 3= Display                     a summary of all MENUs on a MENU file
 4= Display                     the contents of a MENU
 5= Enter/Modify                a VOC MENU selector
 6= Enter/Modify                a VOC stored sentence item
 7= Display                     all MENU selector item on the VOC file
 8= Display                     all stored sentence items on the VOC file
 9= Display                     the dictionary of a file
 10= Print                      a summary of all MENUs on a MENU file
 11= Print                      the contentes of a MENU
 12= Print                      the dictionary of a file
 13= Enter/Modify               a VOC stored paragraph item

 which would you like? (1 - 13)=
```

Printing menu records

To print menu record information, select options 10, 11, and 12 from the MENU Maintenance menu. UniData prompts for the name of the menu file that contains the menu information you want to see. If you select option 11, UniData prompts for the name of the menu. UniData prints it on the default printer and returns to the MENU Maintenance menu prompt.

```
             MENU Maintenance 17:49:24 Apr 01 201
 1= Enter/Modify                a MENU
 2= Enter/Modify                a formatted MENU
 3= Display                     a summary of all MENUs on a MENU file
 4= Display                     the contents of a MENU
 5= Enter/Modify                a VOC MENU selector
 6= Enter/Modify                a VOC stored sentence item
 7= Display                     all MENU selector item on the VOC file
```

```
8= Display                      all stored sentence items on the VOC file
9= Display                      the dictionary of a file
10= Print                       a summary of all MENUs on a MENU file
11= Print                       the contentes of a MENU
12= Print                       the dictionary of a file
13= Enter/Modify                a VOC stored paragraph item

which would you like? (1 - 13)=
```

Creating VOC sentences and paragraphs with the MENUS utility

Some of the options in your menus will execute sentences and paragraphs that are stored in the VOC file; the MENUS utility offers several selections to manage these sentences and paragraphs, as well as sentences and paragraphs that are not associated with a menu. You can accomplish the following from the `MENU Maintenance` menu:

- Create new sentences and paragraphs
- Modify any sentence or paragraph in the VOC file, whether you created the sentence or paragraph within the MENUS utility or not.
- Display and print S- and PA-type records.

Creating a new VOC sentence record

Follow this procedure to create a new VOC sentence (S type VOC record).

1. At the **MENUS Maintenance** menu, enter `6` to select option **6=Enter/Modify a VOC stored sentence item**. UniData prompts for the sentence name.
2. Enter the name of the sentence; this is the @ID for the sentence in the VOC record. UniData prompts for the description. If a record of the same name already exists, UniData displays the record. Press `ENTER` to exit the record. Then, choose another name and repeat steps 1 and 2.
3. Enter the description of the item, or press `ENTER`to continue. The description appears in the VOC record next to the @ID. It is optional. UniData prompts for the sentence.
4. Enter a UniQuery statement. This must be a valid UniQuery statement. UniData displays the VOC record and the Change prompt.
5. Make any necessary changes, or press `ENTER` to save the sentence and return to the main menu.

This concludes the process. The new sentence is now a record in the VOC file.

Displaying the new record

UniData offers many ways to verify the new sentence exists in the VOC file, including the following:

- At the UniData colon prompt, enter the sentence record ID to execute the sentence.
- At the UniData colon prompt, enter the .L command and the VOC record @ID. This is the most efficient way to display a record for a single VOC item.
- Use UniEntry: UNIENTRY VOC *record_ID.*
- Select option 8 from the MENU Maintenance menu. UniData displays all VOC sentences, not just the ones in MENUFILE. This displays VOC records without exiting the MENUS utility.

- At the UniData colon prompt, enter LISTS. UniData displays all sentences in the VOC file in alphabetic order, (Uppercase names are listed first.)
- Use the AE editor: AE VOC *record_ID*

Modifying a VOC sentence

To modify a sentence record in the VOC file, complete the following steps:

1. At the **MENUS Maintenance** menu, enter `6` to select option **6=Enter/Modify a VOC stored sentence item**.
2. Enter the name of the sentence you want to modify and press `ENTER`. UniData displays the sentence attributes and the Change= prompt.

   Note: If you enter a name that is not in the VOC file, UniData assumes you are creating a new VOC record and prompts you for the attribute information. You can exit the MENUS utility without saving by pressing the interrupt key, or you can press `ENTER` to get to the `CHANGE` prompt, and then press `ENTER` again to return to the main menu. To delete the entry from the VOC file, use UniEntry, the .D command, the `ECL DELETE` command, or the AE editor.

   See The UniData VOC file, on page 26 for information about the .D command.
3. At the `Change` prompt, enter the number of the attribute you want to change. UniData deletes the attribute value and prompts you for the new information. If you press `ENTER` without entering anything, the attribute value remains empty.
4. Repeat step 3 for every change you want to make. When you finish making changes, press `ENTER` to return to the main menu.

Creating a new VOC paragraph record

To create a new VOC paragraph record, complete the following steps:

1. At the **MENUS Maintenance** menu, enter `13` to select option `Enter/Modify a VOC stored paragraph item`.
2. Enter the name of the paragraph; this is the @ID for the paragraph in the VOC record. If the name you enter already exists in the VOC file, UniData displays the VOC record for the paragraph and prompts for changes. To exit the record, press `ENTER`. Choose a new name for the paragraph, then repeat steps 1 and 2. UniData prompts for the paragraph description.
3. Enter the description of the item, or press `ENTER` to continue. The description appears in the VOC record next to the @ID. It is optional.
4. After you enter the description, press `ENTER`. UniData prompts for the paragraph content on a sentence-by-sentence basis. Enter the first sentence of the paragraph and press `RETURN`. UniData prompts for the second sentence. Repeat this step to create the paragraph, then press `RETURN` twice. UniData displays the Change prompt.
5. Select the attribute number to make any changes, or press `ENTER`to save the paragraph and return to the main menu.

Displaying the new paragraph record

UniData provides several ways to verify that the new paragraph exists in the VOC file:

- At the UniData colon prompt, enter the paragraph record ID to execute the paragraph.

- At the UniData colon prompt, enter the `.L` command and the VOC record @ID. This is the most efficient way to display a record for a single VOC item.
- Use UniEntry: UNIENTRY VOC *record_ID.*
- At the UniData colon prompt, enter LISTPA. UniData displays all paragraphs in the VOC file in alphabetic order (Uppercase names are listed first.)
- Use the AE editor: AE VOC *record_ID*.

Modifying a VOC paragraph record

To modify a paragraph record in the VOC file, complete the following steps:

1. At the **MENUS Maintenance** menu, enter `13` to select **Enter/Modify a VOC stored paragraph item**. UniData displays the paragraph name prompt.
2. Enter the name of the paragraph you want to modify. UniData displays the VOC record and the Change prompt.If you enter a name that is not in the VOC file, UniData assumes that you are creating a new VOC record and prompts you for the attribute information. You can exit the MENUS utility without saving by pressing the interrupt key sequence, or you can press `ENTER` to get to the `CHANGE` prompt, and then press `ENTER` again to return to the main menu. To delete the entry from the VOC file, use UniEntry, the `.D` command, the `ECL DELETE` command, or the AE line editor.
3. At the `Change` prompt, enter the number of the attribute you want to change. UniData deletes the attribute value and prompts you for the new information. If you press `ENTER` without entering anything, the attribute value remains empty.
4. Repeat step 3 for every change you want to make. When you finish making changes, press `ENTER` to go back to the main menu.

Menu-related VOC commands

UniData provides commands that you execute at the UniData colon prompt to display menu-related VOC file information.

Command	Description
LISTM	Displays the menu items (M-type) in the VOC file. This is helpful if you are working with menus and are unsure of whether a menu has been set up as a VOC item. (Remember, UniData cannot execute a menu unless a record pointing to it exists in the VOC file.)
LISTPA	Displays paragraph (PA-type) VOC items.
LISTS	Displays the sentences (S-type) and UniData SQL commands (SQLV-type) in the VOC file.

Logging in to a menu

You can create a UniData login paragraph in the VOC file that takes the user directly to a menu when they log into UniData. This is a convenient way for users to access a menu, and it could provide additional database security by preventing users from accessing the UniData colon prompt.

You can use UniEntry, AE, or the MENUS utility to create a login paragraph that places users at a menu. This section illustrates how to create a login to a menu with the MENUS utility.

From the MENU Maintenance menu, select 13, Enter/Modify a VOC stored paragraph item. UniData prompts for a paragraph name. In the following example, the login paragraph name is LOGIN:

```
             MENU Maintenance                17:49:24 Apr 01 2011
1= Enter/Modify                       a MENU
2= Enter/Modify                       a formatted MENU
3= Display                            a summary of all MENUs on a MENU file
4= Display                            the contents of a MENU
5= Enter/Modify                       a VOC MENU selector
6= Enter/Modify                       a VOC stored sentence item
7= Display                            all MENU selector item on the VOC file
8= Display                            all stored sentence items on the VOC file
9= Display                            the dictionary of a file
10= Print                             a summary of all MENUs on a MENU file
11= Print                             the contentes of a MENU
12= Print                             the dictionary of a file
13= Enter/Modify                      a VOC stored paragraph item

which would you like? (1 - 13)=13
paragraph name=LOGIN
new paragraph
description=
```

UniData prompts for a description. This is optional information that does not display to the screen upon login. If you do not want to enter a description, press ENTER. UniData prompts for the first sentence of the paragraph. The next example shows a description, a menu name, and a sentence:

```
which would you like? (1 - 13)=13
paragraph name=LOGIN
new paragraph
description=This paragraph advances a user to
 the TEAMS menu at the start of a UniData session.
sentence1=TEAMS
sentence2=DISPLAY If you see this message, you are at the UniData colon prompt.
Enter QUIT or quit, or enter any valid ECL command or UniQuery statement.
sentence3=
```

When you finish entering sentences, press ENTER to get the change= prompt. At this point you can revise any of the paragraph elements, or press ENTER to return to the **MENU Maintenance** menu.

To test the login paragraph, press ENTER until you reach the UniData colon prompt. At the prompt, you can test the paragraph in two ways:

- Enter LOGIN
- Enter QUIT or quit to end the current UniData session, then start a new session.

Whichever way you decide to test the paragraph, the result is the same: UniData advances directly to the TEAMS menu, as shown in the next example. This is consistent with the login paragraph.

```
:LOGIN
Company Sports Teams 18:48:01 Apr 01 2011
1 UNISERVERS Volleyball
2 SCREAMERS Softball
3 RICOCHETS Squash
Enter a number to select any option, ? to get help, "EX" or
 "ex" to return to the main menu.
```

At the menu, select any option, or enter `EX` or `ex` to exit the menu. The next example shows that, instead of returning to the **MENU Maintenance** menu, UniData displays the message from the login paragraph:

```
Company Sports Teams 18:48:01 Apr 01 2011
1 UNISERVERS Volleyball
2 SCREAMERS Softball
3 RICOCHETS Squash
Enter a number to select any option, ? to get help, "EX" or "ex" to return
 to the main menu.ex
If you see this message, you are at the UniData colon prompt. Enter
 QUIT or quit, or enter any valid ECL command or UniQuery statement.
:
```

For more information about creating login paragraphs within UniData, see UniData paragraphs, on page 102.

These examples were generated with UDT.OPTIONS 37 set to on. This option causes UniData to pause between menu selections and prompt the user for a carriage return.

Appendix C: Dictionary conversion codes and format options

This section describes common conversion codes and format options available for dictionary records.

For a complete list of conversion codes and format options, see the *UniBasic Commands Reference*.

Conversion codes

The following table summarizes common conversion codes. A detailed discussion of each follows under separate headings.

Code	Format
MB	Binary
D	System date
MX	Hexadecimal
MCL	Lowercase
MD	Masked decimal
MO	Octal
MT	System time
MCU	Uppercase

Binary (MB)

Syntax

MB [OC]

The MB conversion code converts from decimal or ASCII characters into other numbering systems. The optional OC parameter converts from ASCII characters rather than decimal values.

Examples

The following table illustrates using the binary (base 2) conversion code.

Internal value	Code	Output
117 (decimal)	MB	01110101
X (ASCII)	MB02	01011000

Date (D)

Syntax

D [*y*] [*c*] [*"fmt"*]

The date (**D**) code converts an integer to date display format. If the input value or conversion code is invalid, UniData returns the input value. UniData ignores leading zeroes in the input date.

Note: The internal form of the date is the number of days before or since December 31, 1967. Days before are represented as negative numbers.
With UDT.OPTIONS 4 on, this command converts the text of dates to uppercase. For example, May 13 is converted to MAY 13.
With UDT.OPTIONS 29 on, this command converts Sunday to 7 instead of 0. For information on UDT.OPTIONS, refer to the *UDT.OPTIONS Commands Reference*.

The parameters of *conv.code.expr* are explained in the following table.

Parameter	Description
y	Number of digits between 0 and 4 to represent the year in the formatted date.
c	Delimiter separating day, month, and year. Can be a space, slash, period, comma, or hyphen.
fmt	Format for date display. The following formats are available: *mmddyy* [*yy*] *daymonthyear* *yy*[*yy*] *mm dd* Combine the output format codes in the following table to format the date. The syntax for format codes is the following: [D | W | WA] [[M] [Y] | Q | QA]] [J] [[,] {A*n* | Z*n*} [[,] {A*n* | Z*n*}...]

Examples

The following table illustrates output of an internal date using various conversion codes and format options.

Internal Value	Code	Output
10868	D2/	10/02/97
10868	D/E	02/10/1997
10868	DJ	275
10868	D2*JY	275*97
10868	D4YMD	1997 10 02
1006	DDMYZ0,Z,Z	2 10 1970
7334	DDMY,A,Z4	29 January 1988
7334	DDMY,A,Z4	29 January 1988
7334	DDMY,A3	29 Jan 1988
7334	DDMY	29 01 1988
7334	DD	29
7334	DM	01
7334	DW	5
7334	DWA	Friday
7334	DQ	1
7334	DQA	Winter

Internal Value	Code	Output
7334	DJY	029 1988

Format codes

The following table describes the valid output format codes.

Code	Description
E	International format (dd/mm/yy).
D	Include day in the output date.
M	Include month in the output date.
Y	Include year in the output date.
Q	Converts to quarter numeric format.
QA	Converts to quarter alphanumeric format.
W	Valid for day only. Specifies numeric format.
WA	Valid for day only. Specifies alphabetic format.
A*n*	Valid for month only. A specifies alphabetic format; *n* indicates number of characters.
Z*n*	Valid for month and day only. Specifies numeric format; *n* indicates number of digits to be represented; Z and Z0 suppress zeros.
J	Display Julian date (number of days since January 1).

Note: Following SMA Standards, Monday is the first day of the week (1). UDT.OPTIONS 29 ON converts Sunday to 7 instead of 0.

Hexadecimal (MX)

Syntax

MX[0C]

The hexadecimal conversion code converts from decimal or ASCII into the hexadecimal numbering system. The optional OC parameter converts from ASCII characters rather than decimal values.

Examples

The following table illustrates using the binary (base 16) conversion code.

Internal value	Code	Output
117 (decimal)	MX	75
X (ASCII)	MX0C	58

Lowercase (MCL)

Syntax

MCL

The MCL conversion code converts a string to lowercase.

Example

The following table illustrates output using the MCL conversion code.

Internal value	Code	Output
UNIDATA	MCL	unidata
UniData	MCL	unidata

Masked Decimal (MD)

Syntax

MD*n* [[[*prefix*], [*thsnd_mark*], [*dec_symbl*], [*suffix*]]]
[,] [$] [- | < | E | C | D] [P] [Z] [T] [*xc*]")

The OCONV masked decimal (**MD**) function scales, rounds, and formats a number for display with up to nine decimal places. If the input value or conversion code is invalid, UniData returns the input value.

The following table describes the masked decimal function parameters.

Note: Some parameters set the same values. Results are unpredictable if you make conflicting assignments.

Parameter	Description
n	Sets precision (number of decimal places represented). Can be a number between 0 and 9. If *n* is less than the number of decimal points in the input data, the resulting value is rounded.
prefix], [*thsnd_mark*], [*dec_symbl*], [*suffix*]	The square brackets enclosing these parameters are required. prefix — Nonnumeric character(s) to precede the formatted number. For example, MD2[**] produces "***nnn...*". thsnd_mark — Nonnumeric character(s) to delimit thousands. To specify a comma, enclose it in single quotation marks; then use double quotation marks to enclose the conversion code. For example, " MD2[,']" produces "*...,nnn,nnn*". dec_symbl — Alternate symbol to represent the decimal point. suffix — Nonnumeric character(s) to follow the formatted number. For example, MD2[, , ,**] produces "*...nnn***".
,	Inserts commas to separate thousands, millions, and so forth.
$	Precedes the output with a dollar sign ($).
-	Specifies the negative sign (for negative data) to precede negative numbers.
< or E	Specifies that negative data is surrounded by brackets.
C	Specifies that negative data is followed by CR, indicating a credit.

Parameter	Description
D	Specifies that negative data is followed by DB, indicating a debit.
P	Specifies that no scaling is performed if a decimal is present in the input value.
Z	Specifies that zeros are to be suppressed.
T	Specifies that the number is to be truncated, not rounded to the number of decimal places specified in *n*.
xc	Specifies that the character c is used to fill spaces left to the left of the number after output data is formatted. *x* indicates the size of the mask.

Examples

The following table describes some OCONV masked decimal conversion codes and their results.

Input	Code	Result
12.339	MD3	0.012
1234	MD2,$	$12.34
-1234	MD2$,-	<12.34>
912391	MD2$,D	$9,123.91DB
456789	MD2$,11*	$**4,567.89
123.456	MD24P	123.46
11111112339	MD12[@,*,,%]	@111*111*123.4%

Octal (MO)

Syntax

```
MO[0C]
```

The octal conversion code converts from decimal or ASCII into the octal numbering system. The optional OC parameter converts from ASCII characters rather than decimal values.

Examples

The following table illustrates using the binary (base 8) conversion code.

Internal value	Code	Output
117 (decimal)	MO	165
X (ASCII)	MO0C	130

Time (MT)

Syntax

MT [H] [S] [*c*]

The time (**MT**) function converts the internal time of day from an integer between 0 and 86400 (the number of seconds in a day) to display format. Time is output in the following form:

HH MM [SS]

If the input value or conversion code is invalid, UniData returns the input value.

Parameters

The following table describes each parameter of the time function.

Parameter	Description
H	Indicates that hours are to be formatted in 12-hour format with a.m. or p.m. suffixes.
S	Indicates that seconds are to be included in the output.
c	Specifies a delimiter to be placed between hours, seconds, and fractions of seconds.

Examples

Internal Value	Code	Output
35000	MT	09:43
35000	MTH	09:43AM
35000	MTS	09:43:20
35000	MT*	09*43
35000	MTHS	09:43:20AM

Uppercase (MCU)

Syntax

MCU

The MCU conversion code converts a string to uppercase.

Example

The following table illustrates output using the MCU conversion code.

Internal value	Code	Output
unidata	MCU	UNIDATA
UniData	MCU	UNIDATA

Appendix D: Using UniEntry

UniEntry is a basic tool for entering dictionary and database information. If you enter dictionary records, UniEntry prompts for each of the attributes in the dictionary record in a predefined order. If you enter a data record, UniEntry prompts based on the order of dictionary records defined for the data file.

Entering dictionary records

To access UniEntry and build dictionary records, enter the UNIENTRY command.

Syntax

UNIENTRY DICT *filename*

where *filename* is the file you are entering dictionary records for. The file must already exist in the database.

Entering the record ID

After you enter the UNIENTRY command, UniData prompts for the record ID of the dictionary item you are creating. In these examples, we are building a dictionary record called TEST.

```
:UNIENTRY DICT CLIENTS

DICT CLIENTS RECORD ID=TEST
```

Enter the type

After you enter the record ID, UniEntry prompt for the dictionary record type. Enter one of the following type codes, followed by a description of the dictionary record, if desired.

For more information about dictionary record types, see UniData relational database concepts, on page 10.

Type code	Description
D	Data type. The purpose of a D-type dictionary record is to define the location of an attribute in the data file.
V or I	Virtual type. 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.
PH	Phrase type. A phrase is a part of a UniQuery statement that does not contain a verb.
X	X type. An X-type dictionary attribute stores user-defined information, and is ignored by UniData.

In the following example, D is entered to designate a D-type dictionary record:

```
1 TYPE=D
```

Enter location, formula, or phrase

After you enter the dictionary type, UniEntry prompts for the location. The following table describes the valid options for location, depending upon the type of dictionary record.

Type Code	Description
D	The attribute location in the data record, such as 1, 2, or 3. A value of 1 indicates the first position in the data record, a value of 2 indicates the second position in the data record, and so on. A value of 0 in this attribute always specifies the @ID.
V or I	The virtual field formula to be evaluated.
PH	The contents of the phrase may be any part of a valid UniQuery statement that does not contain a verb.
X	User-defined data.

In the following example, 2 is entered for the location of this D-type dictionary record:

```
2 LOC=2
```

Enter the conversion code

UniEntry next prompts for the conversion code.

If you are entering data that you want to display differently than the internal format, such as a date, enter the conversion code here. For more information about conversion codes in dictionary records, see Dictionary conversion codes and format options, on page 163 and the *UniBasic Commands Reference* manual.

In the following example, the user enters D2/ for the conversion code:

```
3 CONV=D2/
```

If you do not want to enter a conversion code, press `ENTER`.

Enter the display name

The next prompt is for the display name. If a value is present in this field, UniData displays this value as the column heading in a report.

If this attribute is blank, UniData uses the dictionary record name as the column heading. For more information about display names, see UniData relational database concepts, on page 10.

In the following example, the user enters Order Date as the display name.

```
4 NAME=Order Date
```

Enter the format code

UniEntry next prompts for the format code.

This attribute specifies how you want the data attribute to be displayed. The contents of this attribute may include the length and justification of the output, along with any special characters to be included in the display. The format can be any valid parameter supported by the UniBasic FMT function. For a list of valid parameters, see Dictionary conversion codes and format options, on page 163.

In the next example, the user enters 10R (10 characters in length, right justification) for the format code:

```
5 FORMAT=10R
```

Enter the value code specifier

The next prompt is for the value code specifier. The value code specifier identifies whether the data attribute is singlevalued (S), multivalued (MV), or multi-subvalued (MS). If this attribute contains “M”, UniData assumes a multi-subvalued attribute.

In the following example, S is entered for the value code specifier:

```
6 SM=S
```

Enter the association

The next prompt is for the association name, if one exists. The association name is the name of a PH-type dictionary record that contains the association of several related dictionary records. Only multivalued and multi-subvalued fields may be associated. For example, you may have one multivalued attribute containing children’s names, with another multivalued attributed containing those children’s ages. If these two values are associated, each value of the name relates to the same value of the age.

If you do not want to enter an association, press `ENTER`, as shown in the following example:

```
7 ASSOC=
```

Make changes to the record

UniEntry displays all the attributes you have entered, and prompts for any changes that you may wish to make, as shown in the following example:

```
0 @ID=TEST
1 TYPE=D
2 LOC=2
3 CONV=D2/
4 NAME=Order Date
5 FORMAT=10R
6 SM=S
7 ASSOC=

Enter 'DELETE' 'UNCHANGE' 'QUIT' or NUMBER to change
Change=
```

The `Change` prompt allows you to enter one of the following four options.

Option	Description
DELETE	Deletes the entire dictionary record.
UNCHANGE	Reverses your most recent changes and saves the dictionary record unchanged.
QUIT	Saves the most recent changes and exits the dictionary record. Prompts for the @ID of another dictionary record.
NUMBER	Allows you to change the attribute number you enter.
ENTER	Saves the most recent changes, exits the dictionary record, and prompts for another dictionary record ID.

Entering data records

UniEntry prompts for all associated attributes based on the phrase (PH-type) dictionary record identifying the association, prompting sequentially beginning with the first attribute defined in the association.

If the associated attributes are multivalued, UniEntry prompts once for each attribute in the group of associated items, then loops back to the first prompt in the association and repeats for the second value in each group. This loop continues until you press `ENTER` at the first prompt without entering any data.

Entering singlevalued attributes

When you use UniEntry to enter a data record, UniEntry prompts for all singlevalued attributes in location number order, the all multivalued attributes defined in the dictionary for the file.

In the following example, UniEntry prompts for each D-type attribute defined in the CLIENTS file:

```
:UNIENTRY CLIENTS

CLIENTS RECORD ID=3434
 1 First Name=Andrew
 2 Last Name=Samuals
 3 Company Name=Eocket Software
 4 City=Denver
 5 State/Territory=CO
 6 Postal Code=80202
 7 Country=USA
```

Entering multivalued attributes

After you enter all the singlevalued attributes, UniEntry prompts for multivalued attributes, as shown in the following example:

```
Column No.          1
No.               Address
1
Address=1099 -18th Street
Column No. 1
No. Address
1 1099 -18th Street
2
Address=Suite 2200
```

```
Column No.          1
No.               Address
1       1099 -18th Street
2                 Suite 2200
3
Address=
```

The ADDRESS attribute is multivalued, but not associated with another attribute. UniEntry prompts for each multivalue until you press ENTER without entering any data.

UniEntry next prompts each D-type attribute defined in an association. In the next example, UniEntry prompts for the Phone Number, then Phone Category. These multivalued attributes have an association of PHONE. UniEntry continues to loop through these two attributes until you press RETURN without entering any data, as show in the following example:

```
Column No.      1                           2
No  Phone Number        Phone Category
  1
Phone Number=303-294-0800
Phone Category=Main
Column No.      1                           2
No    Phone Number  Phone Category
 1      303-294-0800                  Main
 2
Phone Number=
```

Modifying values

After you have entered all the D-type attributes in a record, UniEntry displays the record, and prompts for any changes you want to make, as shown in the following example:

```
            CLIENTS RECORD ID==>3434

  0 @ID=3434
  1 FNAME=Andrew
  2 LNAME=Samuals
  3 COMPANY=Unidata, Inc.
  4 CITY=Denver
  5 STATE=CO
  6 ZIP_CODE=80202
  7 COUNTRY=USA
  8 ==>ADDRESS
  9 ==>PHONE_NUM PHONE_TYPE

FROM 8 to 9 ARE MULTI VALUED FIELDS SCREENS.

Enter 'DELETE' 'UNCHANGE' 'QUIT' or NUMBER to change
Change=
```

In the next example, the number 2 is entered to change the LNAME attribute. After the value is reentered, press `RETURN` at the `Change`prompt. UniEntry prompts for the next record ID.

```
Change=2
  2 Last Name=Samuels

@ID==3434
  0 @ID=3434
  1 FNAME=Andrew
```

```
  2 LNAME=Samuels
  3 COMPANY=Unidata, Inc.
  4 CITY=Denver
  5 STATE=CO
  6 ZIP_CODE=80202
  7 COUNTRY=USA
  8 ==>ADDRESS
  9 ==>PHONE_NUM PHONE_TYPE

FROM 8 to 9 ARE MULTI VALUED FIELDS SCREENS.

Enter 'DELETE' 'UNCHANGE' 'QUIT' or NUMBER to change
Change=

CLIENTS RECORD ID=
```

If you want to change a multivalued attribute, enter the line number that you want to change. UniEntry then displays each multivalue. Enter the number of the multivalue you want to change, as shown in the following example:

```
            CLIENTS RECORD ID==>3434

  0 @ID=3434
  1 FNAME=Andrew
  2 LNAME=Samuels
  3 COMPANY=Rocket Software,Inc.
  4 CITY=Denver
  5 STATE=CO
  6 ZIP_CODE=80202
  7 COUNTRY=USA
  8 ==>ADDRESS
  9 ==>PHONE_NUM PHONE_TYPE

FROM 8 to 9 ARE MULTI VALUED FIELDS SCREENS.

Enter 'DELETE' 'UNCHANGE' 'QUIT' or NUMBER to change
Change=8

            CLIENTS RECORD ID==>3434

        MULTI VALUED FIELDS SCREEN
    Column No.              1
    No.                   ADDRESS
    1       1099 -18th Street
    2                    Suite 2200
    3
Change which line=2
Line = 2
Enter 'DELETE'/ 'INSERT' /Column number to change=
```

To delete the multivalue, enter `DELETE`. To insert another multivalue, enter `INSERT`. To modify the multivalue, enter the column number you want to change. UniEntry then prompts for the new value, as shown in the following example:

```
Enter 'DELETE'/ 'INSERT' /Column number to change=1
Address=Suite 2500
Change which column=
            CLIENTS RECORD ID==>3434
Column No.                  1
No.                       ADDRESS
1           1099 -18th Street
```

```
2                     Suite 2500
3
Change which line=
```

In the previous example, Suite 2200 is changed to Suite 2500. UniEntry prompts for the next line number to change. Press ENTER without entering any data to go to the next prompt, as shown in the following example:

```
@ID==3434
  0 @ID=3434
  1 FNAME=Andrew
  2 LNAME=Samuels
  3 COMPANY=Unidata, Inc.
  4 CITY=Denver
  5 STATE=CO
  6 ZIP_CODE=80202
  7 COUNTRY=USA
  8 ==>ADDRESS
  9 ==>PHONE_NUM PHONE_TYPE

FROM 8 to 9 ARE MULTI VALUED FIELDS SCREENS.

Enter 'DELETE' 'UNCHANGE' 'QUIT' or NUMBER to change
Change=
```

Enter `ENTER` or `QUIT` to file the record. Enter `UNCHANGE` to exit the record without filing any changes you may have made. Enter `DELETE` if you want to delete the entire record from the database.

Ending a UniEntry session

To end the UniEntry session, press `ENTER` until you return to the ECL command line (:) prompt.