Rocket UniVerse

Guide to ProVerb

Version 11.3.1

October 2016
UNV-1131-PROV-1

Notices

Edition

Publication date: October 2016
Book number: UNV-1131-PROV-1
Product version: Version 11.3.1

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: Overview of ProVerb

ProVerb (commonly called PROC) is a UniVerse processor that interprets command statements stored in a proc (prestored procedure). A proc is made up of one or more ProVerb commands that are stored as a record in a file. A proc defines a sequence of operations to be performed by the ProVerb processor.

This chapter, about the ProVerb command language, discusses why you use procs, the basic operations of the ProVerb processor and how you create and execute procs. Sample procs are presented to demonstrate how to create and run simple procs.

Why use ProVerb?

ProVerb provides a bridge from a Pick system to UniVerse, allowing those of you with Pick backgrounds to port Pick applications to UniVerse. ProVerb is a good tool for minor programming tasks that use the resources of UniVerse.

You can use procs to perform common tasks such as the following:

- Create formatted menus
- Execute a series of UniVerse commands
- Provide interactive user-defined commands
- Build simple file maintenance systems

For example, you can prompt for user input such as a range of dates, perform a select on a file, and feed the selected records into a program that does special formatting for a sales report for the month of December.

A proc is in some ways similar to a shell script in the UNIX environment or to a job control language (JCL) on some mainframe computers, but a proc is more versatile because it can do the following:

- Perform relational data testing
- Transfer control to local and external subroutines
- Branch and loop
- Accept input from users
- Test and verify input data
- Perform arithmetic calculations
- Display buffer contents for debugging
- Format output for the screen or printer

You can call and execute a proc from the UniVerse BASIC SQL Client Interface or the UniVerse Call Interface (UCI).

A proc can perform the operations of a stored paragraph, but it is more versatile because it has input buffers to assemble queries, and it uses variables. Unlike paragraphs, procs can use the GO command to transfer control forward or backward to a statement specified by a label.

ProVerb is intended to complement rather than replace UniVerse paragraphs, menus, or UniVerse BASIC programs. In fact, since the introduction of the EXECUTE statement, most UniVerse users prefer to execute UniVerse commands from within UniVerse BASIC programs instead of using procs. UniVerse BASIC programs are generally easier to maintain because the commands are less cryptic.

About procs

This guide describes PQN format for REALITY compatibility. (UniVerse also supports Pick-style procs, designated with a code of PQ in field 1.)

The basic function of a proc is to build one or more UniVerse commands and then execute them. ProVerb does this by moving data between buffers and using arguments passed from the command line to build a UniVerse command. Any UniVerse command can be executed from a proc. The ProVerb processor submits this command to the UniVerse command processor for execution. Because procs are interpreted by the ProVerb processor, they do not have to be compiled.

A proc communicates with the system and the user through different kinds of buffers and registers. ProVerb uses input and output buffers, file buffers, and select registers to process data. Buffers and registers are discussed in detail in Using ProVerb, on page 9.

Creating procs

Most ProVerb commands consist of one or two letters, such as H or IP. Some ProVerb commands can use specific arguments, such as a string of text and direct or indirect references to a field in a buffer.

Using the UniVerse Editor or UV.VI you can create a simple proc as a record in a file. Use a file that does not already contain data so you do not get unexpected results from mixing programs and data in the same file. The first sample proc consists of the identifying `PQN` and the `O` command to output text to the terminal. You can use the name SAMPLE.PROC if there is no existing entry in your file with that name.

Enter the following proc that prints `Hello everyone!`:

```
>ED PROCS
SAMPLE.PROCNew record.

----: I
0001= PQN proc to display text on the terminal
0002= OHello everyone!
0003=
Bottom at line 2.
----:FI
"SAMPLE.PROC" filed in file "PROCS".
>
```

To run this proc, enter `SAMPLE.PROC` at the UniVerse prompt (<). You should see the following on your terminal:

```
>SAMPLE.PROC
Hello everyone!
>
```

You can insert an `X` command before the `O` command to terminate the proc before outputting the message. The new SAMPLE.PROC looks like this:

```
>ED PROCS SAMPLE.PROC
5 lines long.
----:  P0001: PQN proc to display text on the terminal
0002: XStop the proc before the end
0003: OHello everyone!
Bottom at line 3.
----:
```

When you run the new SAMPLE.PROC, you see the following output:

```
>SAMPLE.PROC
Stop the proc before the end
>
```

Each statement in a proc is executed in sequence unless control is transferred to another statement. Numeric labels identify a statement for branching or looping within the proc. Numbers greater than 0 can be used as labels.

The following example shows a simple proc that lists sales receipts:

```
 >ED PROCS SHOW.SALES
8 lines long.
----:  P
0001: PQN proc to display SHOW.SALES field name
0002: Comment - List receipts by breakpoint field.
0003: HLIST SALES BY
0004: A2
0005: HBREAK.ON
0006: A2
0007: HTOTAL RECEIPTS
0008: P
Bottom at line 8.
----:
```

SHOW.SALES constructs a RetrieVe sentence using an argument passed from the command line. Line 1 contains the code `PQN` followed by an optional description. Line 2 is a comment line beginning with the `C` command that describes the function of the proc. ProVerb ignores the text after the `C` command. Lines 3 through 7 are ProVerb command statements that place data in the output buffer where the RetrieVe sentence is built. Line 8 submits the sentence for execution.

The `H` command at line 3 places the text LIST SALES BY in the primary output buffer. The `A` command at line 4 copies the second field from the primary input buffer to the primary output buffer. The `H` command at line 5 adds the text BREAK.ON to the contents of the primary output buffer, and the `A` command in line 6 again copies the second field from the primary input buffer to the primary output buffer. The `P` command in line 8 submits the contents of the primary output buffer (a RetrieVe sentence) to the UniVerse command processor for execution. For a detailed explanation of each proverb command, see ProVerb commands, on page 22.

Chapter 2: Using ProVerb

This chapter discusses proc format, executing procs, creating calling procs, and the use of input and output buffers, file buffers, and select registers. Examples demonstrate some of the most frequently used ProVerb commands.

Proc format

A proc is created using the UniVerse Editor and is stored as a record in a UniVerse file. A proc consists of a number of ProVerb statements. There is no limit to the number of lines in a proc.

A proc record contains the following:

Line #	Field Contents
001:	PQ [N] [*optional description*]
002:	*sentence 1*
003: . . . *nnn:*	*sentence 2* *last sentence*

Line 1 of a proc must contain either the code `PQN` or the code `PQ` beginning in column 1 of the line to identify the type of entry. The `PQN` format is required for compatibility with a REALITY proc.

Following the `PQN` code, on the same line, you can specify an optional description of the function of the proc. Comment lines, beginning with the letter C, can go anywhere in a proc. A comment can appear as C or as Comment—only the first letter is significant.

Executing procs

The name of a proc is its record ID in the file where it is stored. To execute a proc that is stored in the VOC file, enter its name at the UniVerse prompt.

The UniVerse command processor invokes the ProVerb processor, executes the command submitted to it from ProVerb's primary output buffer, and, in most cases, passes control back to ProVerb.

Once a proc is invoked, the ProVerb processor remains in control until a `P` (Process) or `X` (Exit) command is encountered, or the end of the proc is reached.

For instance, the SHOW.SALES proc could be executed by entering the following command at the UniVerse prompt:

```
>SHOW.SALES CUSTOMER
```

As the proc is executed, a RetrieVe sentence that uses the argument CUSTOMER as a breakpoint field is created and submitted to the UniVerse command processor as the following:

```
LIST SALES BY CUSTOMER BREAK.ON CUSTOMER TOTAL RECEIPTS
```

The report produced by this sentence is displayed on the screen, just as though the entire sentence had been entered directly at the prompt.

Creating calling procs

A proc can be stored as a record in any data file or file dictionary. Storing large procs in a file other than the VOC file is, in fact, recommended. To see the procs stored in the UNIVERSE.VOCLIB file, enter the following command:

```
>LISTPQ
```

To execute a proc that is not stored in the VOC file, there must be an entry in the VOC file that calls the proc by identifying the file where the proc is stored and the name of the proc. This entry can be thought of as a pointer to the location of the proc, but it is actually a proc that consists of a one-way ProVerb control transfer command. The `( )` command identifies the file that contains the stored proc and its name (record ID). The simplified syntax is as follows:

```
(filename record.ID)
```

For instance, you can store the sample proc SHOW.SALES in the PROCS file in the same way programs are stored in the BP file. The VOC file entry can be identified by the same name as the stored proc or by a different name. The following entry for SHOW.SALES is placed in the VOC file:

```
          SHOW.SALES
0001: PQN
0002: (PROCS SHOW.SALES)
```

In line 2, PROCS is the name of the file, and SHOW.SALES is the name of a proc stored in that file. If the proc is stored in a file dictionary, the file name must be preceded by the keyword DICT. The `( )` command does not return control to the calling proc.

You can enter the following command at the UniVerse prompt to run the SHOW.SALES proc:

```
>SHOW.SALES
```

Another way to run the SHOW.SALES proc in the PROCS file is to create the following entry in the VOC file, with the record ID SHOW.SALES:

```
       SHOW.SALES
0001: PQN
0002: (PROCS)
```

The first token on the command line SHOW.SALES is the name of the proc in the PROCS file. The first token on the command line is also the first field in the primary input buffer. To run the proc, enter the name of the proc at the UniVerse prompt as in the previous example.

The previous examples show how to transfer control to a proc in another file using the `( )` command. For an alternate method, see the `[ ]` command in ProVerb commands, on page 22.

Input and output buffers

The ProVerb processor uses two input buffers and two output buffers, nine file buffers, and eleven select registers. Each buffer is treated as a dynamic array with fields separated by field marks. You can use buffers to do common tasks like the following:

- Display the contents for debugging purposes
- Move data between the two sets of buffers to build UniVerse commands
- Process data in UniVerse files
- Store command line arguments and user-prompted input

The four variable-length input and output buffers are divided into pairs:

- For input from the UniVerse command processor or the user
  - Primary input buffer (PIB)
  - Secondary input buffer (SIB)
- For output to the UniVerse command processor
  - Primary output buffer (POB)
  - Secondary output buffer (SOB)

Usually the two primary buffers are the active buffers. Data is stored in a buffer as a sequence of fields or parameters. ProVerb commands can pass a field from an input buffer to an output buffer, insert a string of text, or read input from the terminal into a buffer. As a result of these operations, a UniVerse command is built in the output buffers.

File buffers (different from the input and output buffers previously mentioned) are used to read, write, and delete records in UniVerse files. The select registers are used for processing select lists or multivalued records.

The primary input buffer

The primary input buffer initially holds the contents of the command line that invoked the proc, that is, it holds the name (record ID) of the proc and any arguments entered when the proc was invoked. Each argument is stored as a separate field surrounded by field marks. Text or user input can also be placed in this buffer.

When a field in the input buffer is moved to one of the output buffers, a copy of the field is moved, so that the contents of the input buffer are unchanged by the move.

The secondary input buffer

The secondary input buffer holds data that is input by the user in response to an `IN` command. A second `IN` command overwrites data placed in this buffer.

You can make the secondary input buffer the active input buffer with the `SS` command, and return input to the primary input buffer with the `SP` command. The secondary input buffer is not often used.

The primary output buffer

The primary output buffer is where the command that is to be submitted to the UniVerse command processor is built.

When the contents of the primary output buffer are sent to the command processor, a carriage return is also sent at the end of the command. However, the secondary output buffer requires explicit carriage returns at the end of every line except the last one.

The secondary output buffer

The secondary output buffer is called the stack. It holds data to be used in responding to interactive UniVerse processors invoked by the command from the primary output buffer. Usually the stack is not active and must be selected with the `STON` (Stack on) command.

You must end each line in the stack, except the last, with a carriage return symbol (<) that is placed in the buffer with an H command. When an interactive processor prompts for terminal input, a line from the stack is sent as a response.

Each field in a buffer comprises a string of characters surrounded by field marks (^254). The use of the field mark as a delimiter allows for data fields which are empty or that contain embedded spaces. The IH, IN, and IP commands break a string of text into fields, replacing blanks with field marks. The H, IH, IBH, IBN, and IBP commands insert a string of text as one field in the buffer.

The ProVerb processor recognizes one input buffer and one output buffer as active. Initially the primary buffers—both input and output—are active. A field can be moved only from the active input buffer to the active output buffer. To make the secondary input buffer active, use the SS command. SP returns the primary input buffer to active status. To make the secondary output buffer active, use the STON (Stack on) command. STOFF (Stack off) returns the primary output buffer to active status. If control returns to the ProVerb processor from the UniVerse command processor, the primary input and primary output buffers again become active.

The fields in the primary input buffer can be passed to the output buffers in any order, along with strings of text, to form a UniVerse command.

Pointers

Three pointers mark the current position in a buffer: one for the active input buffer and one for each of the two output buffers. The input buffer pointer is initially positioned on the first field in the primary input buffer, which contains the name of the proc. The output buffer pointers are always positioned at the end of the output buffers.

The F (Forward) command moves the input buffer pointer forward to the first character after the field mark of the next field. If the pointer was at the beginning of the last field in the buffer, the pointer moves to the end of the buffer.

The B(Backward) command moves the pointer to the first character of the previous field. If there is no previous field, the pointer remains at the beginning of the buffer.

The Scommand positions the input buffer pointer at a specific field in the buffer. If the primary input buffer is active, S3 moves the pointer to the first character of the third field.

The BOcommand moves the output buffer pointer back one field, removing the field from the active output buffer.

Most commands move the pointer to the appropriate position for subsequent commands. Fields can also be referenced in ways that do not require positioning the pointer.

The explicit movement of pointers is usually not necessary. For example, once an A command has been executed moving the first field in the primary input buffer to the primary output buffer, the input buffer pointer is positioned at the second field in the primary input buffer, and the output buffer pointer is positioned at the end of the primary output buffer.

Inserting text: The H, IH and IBH commands

There are separate commands for inserting text in the input and output buffers. Text can be inserted in an input buffer as a single field or as a series of fields.

The H command inserts a string of text in the active output buffer. The simplest form of the syntax for the H command is as follows:

```
H text
```

The H command appends the text to the last field in the active output buffer. Put a space between the command and the text to add the text to the end of the buffer as a new field.

In the secondary output buffer, explicit carriage returns must be put at the end of each line of input. The `H` command is used to place the carriage return symbol (<) in that buffer.

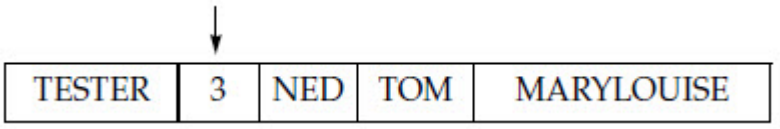
The `IH` and `IBH` commands insert text in the active input buffer. The difference between the `IH` and `IBH` commands is that `IH` interprets each blank space in the specified text as a field delimiter. `IBH` inserts the specified text as a single field in the buffer with blanks intact.

The simple syntaxes are as follows:

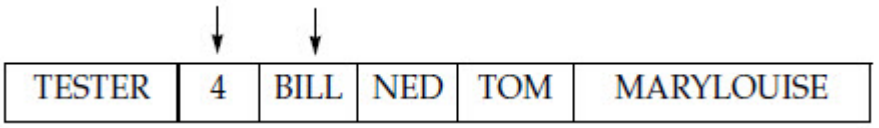
`IH` *`text`*

`IBH` *`text`*

In the following example, the input buffer has five fields with the pointer positioned at the second field:

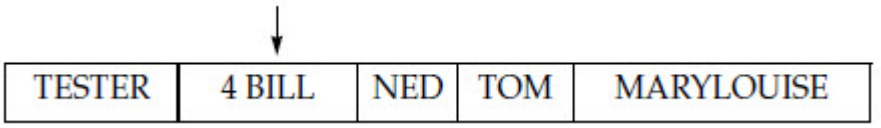


The `IH` command `IH4 BILL` inserts the text “4 BILL” as two fields:



The `IH` command substitutes a field mark for the blank in the string of text. The second field in the input buffer is replaced by two fields. There are now six fields.

The `IBH` command `IBH4 BILL` inserts the text “4 BILL” as a single field. The following diagram illustrates the result. The text replaces the second of five fields in the input buffer.



The `IH` and `IBH` commands can get the text from any buffer, file buffer, or select register by a direct or indirect reference. In PQN procs, a backslash (\ ) can be specified instead of text to make an empty field. For more information about these commands, see ProVerb commands, on page 22.

Copying fields: The A command

When copying data from the currently active input buffer to either output buffer, there are two ways to indicate which field is to be copied:

- Use the current field at the position of the pointer.
- Specify the number of the field in sequence.

The A command copies fields or a part of a field from the active input buffer to the active output buffer. The A command copies the field; it does not remove the field from the input buffer.

The two basic versions of the `A` command are as follows:

A [*c*]

A [*c*] *p*

The first version has no arguments and moves the field at the input buffer pointer.

The second version uses the numeric argument *p* to indicate a field by its location in the input buffer. It sets the input buffer pointer to the first character of this field before moving the field to the output buffer.

An optional surround character, *c*, can be specified to surround the field being sent to the primary output buffer. If *c* is not specified, the field is copied to the output buffer surrounded by blanks. A part of a field can be extracted by specifying either the column position of the first character to move or the number of characters to extract, or both.

Specifying the current field

The first version of the `A` command moves the field pointed to by the input buffer pointer. The input buffer pointer is initially at the first field of the buffer that contains the name of the proc or the name of a VOC entry that transfers control to a proc.

Generally the `A` command follows `F, B`, or `S` commands that position the pointer at a particular field. For instance, to move the fifth field from the input buffer to the output buffer, the input buffer pointer must be moved to the fifth field before the `A` command is executed. After the operation, the input buffer pointer is at the beginning of the next field or at the end of the buffer if there are no remaining fields.

To demonstrate the movement of the pointer, the sample proc could be rewritten using a different version of the `A` command. Here are the steps involved:

1. The `S2` command positions the input buffer pointer at the beginning of the second field.
2. The `A` command at line 5 copies that field (containing the text CUSTOMER) and sends it to the output buffer.
3. After line 5 is executed, the input buffer pointer is at the end of the primary input buffer because the input buffer contained only two fields. If you executed the SHOW.SALES proc with more than one field name, the input buffer pointer would be at the third field after line 5 was executed.
4. In order to send the same field again, after the `H` command has placed the BREAK.ON keyword in the output buffer, the input buffer pointer must be reset to the second field.
5. The `B` command at line 7 moves the input buffer pointer back to the previous field. (The `S` command could have been used instead to select the second field.)

```
       SHOW.SALES
0001: PQN SHOW.SALES field name
0002: Comment - List receipts by breakpoint field.
0003: HLIST SALES BY
0004: S2
0005: A
0006: HBREAK.ON
0007: B
0008: A
0009: HTOTAL RECEIPTS
0010: P
```

You could execute this proc by entering the following command at the UniVerse prompt:

```
>SHOW.SALES CUSTOMER
```

Specifying the field position

A numeric argument with the A command designates a field by its sequential position in the input buffer. Using this version of the command has the same effect, shown in lines 4 and 5 in the example proc, as an S command followed by an A command.

The statement A2 copies the second field of the active input buffer to the output buffer. When indicating field by location, it is not necessary to follow the position of the input buffer pointer.

The following version of SHOW.SALES uses A2 twice to move the user-supplied argument CUSTOMER to the output buffer. When the operation is complete, the input buffer pointer is positioned at the first character of the next field in the buffer or at the end of the buffer if there are no remaining fields.

```
      SHOW.SALES
0001: PQN - SHOW.SALES field name
0002: Comment - List receipts by breakpoint field.
0003: HLIST SALES BY
0004: A2
0005: HBREAK.ON
0006: A2
0007: HTOTAL RECEIPTS
0008: P
```

Direct and indirect references: % and

References can be used to access fields in the file buffer for later manipulation of the data. Direct and indirect references can be made to fields in the primary input buffer and the currently active output buffer. Not all commands can use direct or indirect references (such as the A command), but other commands require this type of referencing (such as the MV command).

Direct and indirect references can also select fields in file buffers and select registers. This section discusses direct and indirect referencing of input and output buffers.

A direct reference uses a buffer reference symbol followed by a number that indicates one of the fields in the buffer. The buffer reference symbol for the active input buffer is %. Thus, %2 refers to field 2 in the active input buffer. The buffer reference symbol for the currently active output buffer is #. If the stack is on, #4 refers to field 4 in the secondary output buffer. Referencing a field with % or # does not reposition the pointer.

Unlike a direct reference, an indirect reference does not explicitly specify the field number as an argument. Instead, the field number is obtained by referencing the value of another field in one of the buffers. This value provides the number of the field being referenced in the primary input buffer or currently active output buffer. If a nonnumeric value is referenced, 0 is used as the field number.

For example, the primary input buffer contains the following:

TESTER	3	TOM	NED	MARYLOUISE

In this example, the indirect reference %%2 obtains the value TOM. The first % refers to the primary input buffer. Then %2 refers to the value in field 2 of the primary input buffer. The value is 3. The number 3 refers indirectly to the value in field 3 of the primary input buffer, which in this case is TOM.

To make a similar indirect reference, but this time to the active output buffer, you might use the indirect reference #%2. This refers indirectly to the value of the field whose number is contained in field 2 of the currently active output buffer.

Using the stack: The STON command

The stack, or secondary output buffer, can pass lines of input to an interactive UniVerse processor invoked by the command in the primary output buffer. The `STON` (Stack on) command activates the secondary output buffer. Once it is active, data can be placed in the stack to form lines of input. Each line except the last must end with an explicit carriage return symbol (<).

The sample proc REVISING shows how to use the secondary output buffer. When ReVise is invoked on a file, the user is prompted to enter the record ID. This proc lets a user invoke ReVise by supplying the record ID on the command line.

```
      REVISING
0001: PQN REVISING filename record.id
0002: HREVISE
0003: A2
0004: STON
0005: A3
0006: P
```

When this proc is invoked, the primary input buffer contains the name of the proc and the two arguments supplied on the command line:

```
>REVISING SUN.MEMBER 6100
```

The primary input buffer can be visualized like this:

REVISING	SUN.MEMBER	6100

The H statement on line 2 inserts the command `REVISE` in the primary output buffer. The A2 statement copies the second field into the primary output buffer. The `STON` command activates the secondary output buffer so that the A3 statement can pass the third field from the primary input buffer to the secondary input buffer.

The primary and secondary output buffers, just before they are processed, can be visualized like this:

REVISE	SUN.MEMBER

6100

When the contents of the primary output buffer are submitted to the UniVerse command processor, the ReVise processor is invoked on the file SUN.MEMBER. When ReVise prompts for the record ID, the first field from the secondary output buffer is submitted. This proc could be expanded to include

additional lines of ReVise commands to update a record. For more information about ReVise, see *UniVerse System Description*.

Transferring control: The IF, GO, and GOSUB commands

There are several ways to transfer control within a proc, including those that are summarized as follows:

- The `IF` command can test a value and transfer control under certain conditions.
- The `GO` command transfers control unconditionally.
- The `GOSUB` command transfers control to a local subroutine, and the `RSUB` command passes control back.
- Control can also be passed to an external subroutine. In the discussion of calling procs, the `( )` command was shown to execute a one-way control transfer to another proc.

With the `( )` command, control does not return to the invoking proc. To execute an external subroutine that does return to the invoking proc, use the `[ ]` command.

Transferring control to a subroutine

The following example is a new version of SHOW.SALES that accomplishes the same basic task of building and executing a RetrieVe sentence. In this version the field name can be passed from the command line or entered by the user at a prompt. This is accomplished by transferring control to a subroutine if an argument is not supplied on the command line.

```
      SHOW.SALES
0001: PQN SHOW.SALES field name
0002: Comment - List receipts by breakpoint field.
0003: HLIST SALES BY
0004: IF # A2 GOSUB 200
0005: A2
0006: HBREAK.ON
0007: A2
0008: HTOTAL RECEIPTS
0009: P
0010: GO 999
0011: 200 OENTER NAME OF FIELD IN SALES FILE
0012: OOR PRESS Q TO QUIT:
0013: S2
0014: IP
0015: IF A2 = Q GO 999
0016: RSUB
0017: 999 X
```

This version introduces several new commands. Note the following actions:

- The `IF` command is used twice for conditionally transferring control. In line 4 the `IF` command tests for the presence of a second field in the primary input buffer. If the value does not exist, the `GOSUB` command transfers control to a local subroutine that prompts the user to supply the value. The subroutine is identified by the statement label 200.
- The `O` command sends the following string of text to the terminal screen to prompt the user for input:

  ```
  ENTER NAME OF FIELD IN SALES FILE OR PRESS Q TO QUIT:
  ```

- The `IP` command reads input from the terminal and inserts it in the primary input buffer.

- The second IF command occurs in the subroutine and provides the user with a way to quit without executing the RetrieVe sentence. It tests whether or not Q is the value of the second field in the input buffer. Control is passed to the exit statement, with the label 999, or returned with the RSUB command for processing the RetrieVe sentence.
- The GO command unconditionally transfers control to an exit statement that ends the proc.

In PQN procs, more than one IF command can go on a line. Separate these multivalued IF commands with value marks. The following example shows a multivalued IF command that compares a value with one of several different values. Based on this comparison, the command executes one of several commands:

```
IF A = 1V2V3V4 GOSUB 100VGO 200VRTNVXExiting...
```

File buffers

File buffers are used to access data in UniVerse files. There are nine file buffers available for reading and writing records. Once a UniVerse file is assigned to any of the file buffers, a record from that file can be read into the buffer. Any field of that record can be changed and new fields can be added. ProVerb commands can reference the data in a file buffer and move fields to and from the input and output buffers. The following five commands are used with file buffers:

Command	Description
F-CLEAR	Clears one of the nine file buffers.
F-OPEN	Opens a UniVerse file and assigns it to a numbered file buffer.
F-READ	Reads a record into the file buffer.
F-WRITE	Writes the record back to the disk.
F-DELETE	Deletes the record from the file.

The F-OPEN and F-READ commands both require that an error return statement immediately follow them on the next line. If an error is encountered while opening the file or reading the record, the error return statement is executed. The error return statement can invoke a subroutine that allows the user to recover from the error (such as inputting a new argument), or it can simply be an exit statement. If you do not want the proc to do anything when an error occurs, you can put a comment statement on the error line.

It is important to know which commands require error return statements, because it determines whether or not the subsequent command is executed.

To use a file buffer, follow this sequence of commands:

1. Assign a UniVerse file to one of the file buffers. In the following example, the SALES file is assigned to file buffer 2:

   ```
   F-OPEN 2 SALES
   X...Cannot Open File...
   ```

2. Follow the F-OPEN command with an error return statement. The X (Exit) command is executed only if an error results from trying to open the file.

   A record is read into the file buffer.
3. Specify the file buffer number and a record ID. For example:

   ```
   F-READ 2 61000
   X...Cannot Open Record...
   ```

4. Follow the F-READ command with an error return statement.
5. A variety of operations can be performed on the data in the file buffer.

6. After those operations have been completed, write the record back to the file and save to disk. For example:

```
F-WRITE 2
```

The record ID need not be specified since it is stored in field 0 of the file buffer.

Referencing file buffers with an ampersand

A direct or indirect reference can be used to access fields in the file buffer. An ampersand (&) is the symbol for file buffers. It is followed by the number of the assigned file buffer. A period is used as a separator, and it is followed by the number of the field being referenced (the same number that defines its Location).

For instance, &1.3 directly refers to field 3 in file buffer 1. If the number of the field being referenced is beyond the number of fields in the file buffer, additional empty fields are established to create the field.

An indirect reference can be used to obtain a value indicating the field number from either the primary input buffer or the active output buffer. An indirect reference, such as &4.#3, refers to field 3 in the active output buffer (#) to obtain the number of the field being referenced in file buffer 4.

Note: Field number 0 refers to the record ID in a file buffer. The direct reference &2.0 supplies the record ID of the record in file buffer 2. Also, if an indirect reference is made to a nonnumeric field, its value is zero and thus refers to the record ID.

In the following example, a portion of a proc is shown that opens the SALES file, reads in a record, and uses the `IH` command to get a string of text from a file buffer and insert it in the primary input buffer. The direct reference &2.4 gets field 4 in file buffer 2.

```
022: F-OPEN 2 SALES
023: X...Unable to open SALES file
024: F-READ 2 61000
025: X...Cannot find record
026: IH&2.4
```

Fast file buffer

The fast buffer is a single buffer available for read-only operations involving a single record from a file. The `FB` command is used to open the fast buffer and read in a record, combining the operations of `F-OPEN` and `F-READ`. The `FB` command also clears the previous contents of the fast buffer.

The `FB` command requires that an error return statement immediately follow it on the next line. If an error is encountered while opening the file and reading the record, the error return statement is executed.

A direct or indirect reference to the fast buffer is the ampersand without a file buffer number, followed by the field number. &5 refers to field 5 in the fast buffer.

In the following example, the proc ID.CHECK opens a file and reads a record into the fast buffer. Both the file name and the record ID are supplied on the command line. The `T` command displays the record ID and fields 1 and 2 of the fast buffer on the terminal screen.

```
       ID.CHECK
0001: PQN ID.CHECK file name record.id
```

```
0002: C DISPLAY FIRST TWO FIELDS ON RECORD
0003: FB (%2 %3)
0004: GO 999
0005: T %3,S5,&1,&2
0006: 999 X
```

Direct references obtain the values of fields 2 and 3 of the primary input buffer (%2 and %3) and fields 1 and 2 of the fast buffer (&1 and &2). The `T` command permits a wide range of format specifications for positioning the cursor and clearing the screen. In this proc, the `T` command displays the record ID on the screen, moves five spaces, and displays two fields from the fast file buffer.

For details on the terminal output options, the `O` and `T` commands, and the `L` command for printed output, see ProVerb commands, on page 22.

The ID.CHECK proc verifies a record ID as in the following command:

```
>ID.CHECK SUN.MEMBER 6100
```

The member's first and last names are displayed on the screen, since they correspond to fields 1 and 2 in the record. For practice, you might want to create ID.CHECK and REVISING. Try adding to ID.CHECK an external transfer to REVISING that would allow a user to verify a record ID before actually invoking ReVise. You could ask the user whether or not to invoke ReVise on the specified record.

Update locks

An update lock prevents other users from changing a record while it is being accessed by ProVerb. `F-UREAD` is a version of the `F-READ` command and `FBU` is a version of the `FB` command that are used to place an update lock on a record being read into one of the file buffers or the fast buffer.

While an update lock remains in effect, other users cannot retrieve and update any record in the group. The lock is removed when an `F-FREE` command is executed, when a write or delete operation is performed, or when the proc ends.

Select registers

There are 11 select registers, numbered from 0 through 10, that make select lists available to the ProVerb processor. The select list is assigned a numbered register by adding a TO *n* clause to a `SELECT, SSELECT, GET.LIST`, or `FORM.LIST` sentence. This is also the way a numbered select list is created from the UniVerse command processor.

The select register contains a select list whose elements can be accessed by a direct or indirect reference. Each reference to a select register returns a single element of the select list and removes that value from the select register. The next reference returns the next element in the register.

Another use of select registers is to store multivalued fields. A multivalued field can be moved from a file buffer to a select register. References to the select register obtain each value in succession.

Referencing select registers with an exclamation point

The contents of a select register can be accessed directly or indirectly. An exclamation point (!) is the symbol for select registers. It is followed by the number of the select register. When a select register is referenced, it returns a single element of the select list (a single value if the register contains a multivalued field). The next reference to the register returns the next value in sequence.

In the following example, the UniVerse sentence is submitted from a proc to the UniVerse command processor:

```
SELECT SALES WITH SALE.DATE GT 8/1/91 TO 4
```

Select register 4 contains the list of selected record IDs from the SALES file. The `IH!4` command will move the first record ID in select register 4 into the active input buffer.

Moving Data: The MV command

The `MV` command moves data between the primary input buffer, the currently active output buffer, and any file buffer or select register. Direct and indirect references must be used in an `MV` command. The syntax of the `MV` command is as follows:

MV *destination source*

The `MV` command makes a copy of the field designated as the source and moves it into the position designated as the *destination*. The moved field does not replace an existing field but is inserted at that position. Direct and indirect references indicate the target field and the source field. *source* can also be specified as a constant.

Direct and indirect referencing symbols

The following table provides a summary of the symbols used in direct and indirect referencing.

Symbol	Reference	Direct	Indirect
%	Primary input buffer	%2	%#7
#	Active output buffer	#2	#2&3
&*n.*	File buffer *n*	&2.3	&%2.#1
&	Fast file buffer	&3	&%3
!*n*	Select register *n*	!2	!#3

Direct references consist of the symbol of the buffer to be referenced, followed by the number of the field to be referenced. A percent sign (%) references the primary input buffer, for example,%2 refers to field 2 in the primary input buffer.

A hash sign (**#**) references the active output buffer. If the stack is off, #4 references field 4 in the primary output buffer. If the stack is on, #4 refers to field 4 in the secondary output buffer. Referencing a field with % or # does not reposition the pointer.

Chapter 3: ProVerb commands

This chapter describes the commands that are available with ProVerb. The commands are arranged in alphabetical order.

Command summary

The following table briefly describes all ProVerb commands.

Command	Description
A	Moves the field to the output buffer.
B	Moves the input buffer pointer backward.
BO	Backs up the output buffer pointer.
C	Specifies a comment.
D	Displays input buffers.
F	Moves the input buffer pointer forward.
F;	Performs arithmetic operations.
F-CLEAR	Clears a numbered file buffer.
F-DELETE	Deletes a record from the file.
F-FREE	Releases an update lock.
F-OPEN	Opens a UniVerse file.
F-READ	Reads the record into the file buffer.
F-UREAD	Sets the record update lock.
F-WRITE	Writes the record from the file buffer.
FB	Reads to the fast buffer.
FBU	Sets the record update lock.
GO	Transfers control.
GOSUB	Executes the local subroutine.
H	Adds text to the output buffer.
IF	Executes commands conditionally.
IH	Inserts text into the input buffer.
IN	Enters data in the secondary input buffer.
IP	Reads input from the terminal.
IS	Same as the `IN` command.
IT	Reads the tape to the primary input buffer.
L	Formats output for the printer.
M	Marks a location.
MV	Moves data between buffers, registers, or both.
MVA	Moves data to the buffer as multivalues.
MVD	Deletes multiple values.
O	Outputs text to the terminal.
P	Processes output buffers.
PP	Displays and processes output buffers.
Q	Exits the proc and returns to the command processor prompt.

Command	Description
RI	Clears the input buffers.
RO	Resets the output buffers.
RSUB	Returns from the local subroutine.
RTN	Returns from the external subroutine.
S	Sets the input buffer pointer.
SP	Selects the primary input buffer.
SS	Selects the secondary input buffer.
STOFF	Selects the primary output buffer.
STON	Selects the secondary output buffer.
T	Formats terminal output.
TR	Traces execution.
U	User exits.
X	Exits the proc and returns to the calling process.
+	Adds the integer.
-	Subtracts the integer.
()	Executes another proc.
[]	Executes an external subroutine.

A

Use `A` to copy a field from the input buffer to the output buffer. The `A` command does not remove the field from the input buffer.

Syntax

```
A [ c ] [ p ] [ , ] [ m ]
A ( [ n ] | n,m | ,m ] )
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
c	The surround character. You can use any nonnumeric character except an open parenthesis or comma to surround the string being sent to the primary output buffer. For instance, you can specify single quotation marks to surround record IDs. If you do not specify *c*, the string is surrounded by blanks.
p	The number of a field in the input buffer. After copying field *p*, the A command resets the input buffer pointer to the first character of the *p*th field.
m	Specifies a string of *m* characters starting from the current position of the input buffer pointer. If A encounters a field mark or the end of the input buffer before moving the specified number of characters, the operation quits.

Parameter	Description
n	Specifies a string beginning with column *n* and continuing up to the next field mark in the active input buffer.

Description

Without options, the A command moves a field in the active input buffer to the end of the active output buffer. The designated field consists of the characters from the buffer pointer to the next field mark. After copying a field, A positions the input buffer pointer at the first character of the next field.

If the secondary output buffer is active, the A command moves field values to the secondary output buffer without blanks or the surround character *c*.

You can set the configurable parameter PROCACMD to a nonzero value to cause the A command using the syntax A(*n*,*m*) or A(,*m*) to ignore a field mark.

B

Use B (Backward) to move the input buffer pointer backward to the previous field.

Syntax

```
B
```

Description

The B command positions the input buffer pointer at the first character of the previous field, unless the pointer is in the middle of a field. In that case, B moves the pointer to the beginning of the current field.

BO

Use BO to move the output buffer pointer back to the end of the previous field, discarding the characters it moves over. BO redefines the end of the active output buffer. Output buffer pointers always indicate the end of the buffer.

Syntax

```
BO
```

Description

If the buffer contains only one field or is empty, the BO command moves the pointer to the beginning of the buffer.

C

Use C to enter a comment.

Syntax

```
C [ text ]
```

Parameter

The following table describes the parameter of the syntax.

Parameter	Description
text	The comment.

Description

The C command indicates that the accompanying *text* is a comment. Comments are ignored during processing. Users frequently make the second line of a proc a comment that explains the purpose or function of the proc. A comment can appear as C or Comment; only the first letter is significant.

Note: Too many comments can lengthen a proc and slow down execution.

D

Use D to display one or more fields from the active input buffer. This command is useful for debugging a proc.

Syntax

```
D [ ref | p ] [ ,m ] [ + ]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description	
ref	A direct or indirect reference to a buffer or select register containing the field number of the input buffer. You can use the following buffer reference symbols:	
	Symbol	**Reference**
	%	Primary input buffer
	#	Active output buffer
	&*n*.	File buffer *n*
	&	Fast file buffer
	!*n*	Select register *n*
p	A field number in the input buffer. If *p* is 0, D displays the entire input buffer.	
m	The number of characters to display starting from the specified position. D displays *m* characters, up to the next field mark.	
+	Suppresses a carriage return.	

Description

Without qualifiers, the D command displays the field designated by the input buffer pointer. The D command does not move the pointer.

F

Use `F` (Forward) to move the input buffer pointer forward to the next field mark in the buffer.

Syntax

```
F
```

Description

If the pointer is at the end of the buffer or in the last field, the `F` command moves the pointer to the end of the buffer.

F;

Use `F;` to perform arithmetic functions on values in a stack. You can add, subtract, multiply, and divide using fixed-point, integer arithmetic.

Syntax

```
F;element [ ;element … ]
```

Parameter

The following table describes the parameter of the syntax.

Parameter	Description
;	Separates each element.

elements is one or more of the following operands or operators:

Operand	Description
ref	Stack value obtained by an indirect or direct reference to a buffer or select register.
[C]*n*	Stack numeric constant *n*.
+	Add the top two values.
-	Subtract the first value on the stack from the second.
*	Multiply the top two values.
/	Divide the second value on the stack by the first.
R	Get the remainder from dividing the second value on the stack by the first.
{	Reverse the order of the top two values.
?P	Put the result at the current position in the primary input buffer.
?*ref*	Put the result at the location specified by the direct or indirect reference.

You can use the following buffer reference symbols:

Symbol	Reference
%	Primary input buffer

Symbol	Reference
#	Active output buffer
&*n*.	File buffer *n*
&	Fast file buffer
!*n*	Select register *n*

Description

The `F;` command is a stack processor that calculates functions by reading the elements from left to right, putting each new operand or value on top of the stack. The operation indicated by an operator is performed on the top two values on the stack. The two operands are deleted from the stack and the result of the operation is the new value at the top of the stack. This value can be moved into the primary input buffer by specifying the ?P operator at the end of the command.

The stack can hold up to 64 values.

F-CLEAR

Use `F-CLEAR` to clear a numbered file buffer before it is made available for an `F-OPEN` command.

Syntax

```
F-CLEAR fb
```

Parameter

The following table describes the parameter of the syntax.

Parameter	Description
fb	The numbered file buffer (1 through 9) to clear.

Description

Use an `F-CLEAR` command to make a file buffer available as scratch space.

F-DELETE

Use `F-DELETE` to delete a record from an opened file.

Syntax

```
F-DELETE fb
```

Parameter

The following table describes the parameter of the syntax.

Parameter	Description
fb	The numbered file buffer (1 through 9) that contains the record to delete from disk.

Description

The F-DELETE command identifies a file buffer containing a record to delete from the file. You do not need to specify the record ID because it is defined as field 0 (&*fb*.0) in the file buffer.

F-DELETE does not clear or change the contents of the file buffer.

This command does not provide for an error condition, as F-OPEN and F-READ do.

A proc terminates if it executes an F-DELETE command that refers to an unopened file buffer.

F-FREE

Use F-FREE to release record update locks set by F-UREAD or FBU commands. When the lock is released, other users can access and update the record.

Syntax

```
F-FREE [ fb [ record.ID | ref ] ]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description	
fb	The number of the file buffer (1 through 9) assigned to the file when it was opened.	
record.ID	The ID identifying a locked record in the numbered buffer.	
ref	A direct or indirect reference to a buffer or select register containing the record ID. You can use the following buffer reference symbols:	
	Symbol	**Reference**
	%	Primary input buffer
	#	Active output buffer
	&*n*.	File buffer *n*
	&	Fast file buffer
	!*n*	Select register *n*

Description

Without any qualifiers, the F-FREE command releases all the locks set by the proc. If a numbered file buffer and record ID are specified, it releases the group lock for that file.

F-OPEN

Use F-OPEN to open a UniVerse file and assign it to one of nine file buffers. After a file is opened, you can use the F-READ and F-WRITE commands to read and write records from a proc.

Syntax

```
F-OPEN fb [ DICT ] [ filename | ref ]
error statement
```

```
next program statement
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description	
fb	The number of the file buffer (from 1 through 9) to open.	
DICT	Opens only the file dictionary.	
filename	The name of the file to open.	
ref	A direct or indirect reference to a buffer or select register containing the file name. You can use the following buffer reference symbols:	
	Symbol	**Reference**
	%	Primary input buffer
	#	Active output buffer
	&*n*.	File buffer *n*
	&	Fast file buffer
	!*n*	Select register *n*
error statement	The command to be executed if the named file does not exist.	
next program statement	The command to be executed if the named file exists and can be opened.	

Description

If the file cannot be found, the command on the line immediately following the `F-OPEN` command is executed. If the file is found, this command is skipped.

A file buffer must be specified explicitly by number in a ProVerb command. However, the contents of a file buffer can be accessed by the ampersand (&) form of the direct or indirect reference.

Once a file is opened, it remains available throughout proc execution. At the end of a proc, files are automatically closed.

A file buffer can be reused by issuing another `F-OPEN` command.

F-READ

Use `F-READ` to read a record from a UniVerse file into a numbered file buffer. Before a record can be read into the file buffer, `F-OPEN` must be executed to open the file and assign it to a file buffer.

Syntax

```
F-READ fb [ record.ID | ref ]
error statement
next program statement
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description	
fb	The number of the file buffer (1 through 9) assigned to the file when it was opened.	
record.ID	The ID of the record to be read into the numbered buffer.	
ref	A direct or indirect reference to a buffer or select register containing the record ID. You can use the following buffer reference symbols:	
	Symbol	**Reference**
	%	Primary input buffer
	#	Active output buffer
	&*n*.	File buffer *n*
	&	Fast file buffer
	!*n*	Select register *n*
error statement.	The command to be executed if the record does not exist	
next program statement	The command to be executed if the record exists.	

Description

The `F-READ` command specifies the file buffer to which the file is assigned and the record ID that identifies a particular record in the file. `F-READ` clears the file buffer before it tries to locate the record in the file.

If the record cannot be found, the command on the line immediately following the `F-READ` command is executed. If the record is found, this command is skipped.

Use F-UREAD if a record might be updated by other processes while the proc executes.

F-UREAD

Use `F-UREAD` to set an update record lock and read a record from a UniVerse file into one of the numbered file buffers. An update record lock prevents the record from being updated by other processes. This command has the same purpose as the READU statement in UniVerse BASIC.

Syntax

```
F-UREAD fb record.ID | ref
error statement
next program statement
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
fb	The number of the file buffer (1 through 9) assigned to the file when it was opened.
record.ID	The ID of a record to read into the numbered buffer.

Parameter	Description	
ref	A direct or indirect reference to a buffer or select register containing the record ID. You can use the following buffer reference symbols:	
	Symbol	**Reference**
	%	Primary input buffer
	#	Active output buffer
	&*n*.	File buffer *n*
	&	Fast file buffer
	!*n*	Select register *n*
error statement	The command to be executed if the record does not exist.	
next program statement.	The command to be executed if the record exists	

Description

The `F-UREAD` command specifies the file buffer and the record ID that are used to identify the group where a record belongs in the file. The `F-OPEN` command must be executed before `F-UREAD` to specify the name of the UniVerse file and the number of the opened file buffer.

If a record is already locked and a proc executes an `F-UREAD` command, execution is suspended until the record lock is released.

If the record cannot be found, the command on the line immediately following the `F-UREAD` command is executed. If the record is found, this command is skipped.

The update record lock is set for the group to which the record hashes. Even if the read is unsuccessful, a lock is set on the group where the record was expected to be located.

Unless the proc executes an F-WRITE, F-DELETE, or F-FREE command, the lock is maintained until the proc terminates.

Use the `F-FREE` command to unlock records.

F-WRITE

Use `F-WRITE` to write a record from a file buffer to disk.

Syntax

```
F-WRITE fb
```

Parameter

The following table describes the parameter of the syntax.

Parameter	Description
fb	The numbered file buffer (1 through 9) that contains the record to write to disk.

Description

Only the number of the file buffer is specified. You do not need to specify the record ID because it is defined as field 0 (&*fb*.0) in the file buffer.

`F-WRITE` does not clear or change the contents of the file buffer.

This command does not provide for an error condition, as F-OPEN and F-READ do.

A proc terminates if it executes an F-WRITE command that refers to an unopened file buffer.

FB

Use FB to open a file and read a record into the fast buffer. It combines the function of F-OPEN and F-READ, but it can be used to read only a record. It is best used for processing a single record in a file. There is only one fast buffer, and its contents are destroyed by the next FB command.

Syntax

```
FB [ U ] ( [ [ DICT ] filename | ref ] [ record.ID | ref ] )

error statement

next program statement
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description	
U	Puts an update record lock on the requested record.	
DICT	Opens only the file dictionary.	
filename	The name of the file to be opened.	
record.ID	The ID of a record to read into the fast buffer.	
ref	A direct or indirect reference to a buffer or select register containing the file name or record ID. You can use the following buffer reference symbols:	
	Symbol	**Reference**
	%	Primary input buffer
	#	Active output buffer
	&*n*.	File buffer *n*
	&	Fast file buffer
	!*n*	Select register *n*
error statement	The command to be executed if the record or file does not exist.	
next program statement	The command to be executed if the record or file exists.	

Description

The FB command specifies the file to be opened and the record to be read into the fast buffer. If you do not specify a record ID, the current field in the primary input buffer is used. If either the file or the record cannot be found, the command on the line immediately following the FB command is executed. If both the file and the record are found, this command is skipped.

The fields in the fast buffer are referenced with a different form of the ampersand (&) reference. No file buffer number is specified and no period (.) is needed. For example, &5 references the fifth field in the fast buffer.

Use the F-FREE command to unlock records.

GO

Use GO to transfer control to another statement in a proc.

Syntax

```
GO | G | GOTO [ label | A-command | ref | F | B ]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description	
label	A number specifying a labelled ProVerb statement. Execution continues at this statement.	
A-command	Any form of the A command. The result is used to compute the label number to branch to.	
ref	A direct or indirect reference to a buffer or select register containing the label number to branch to. You can use the following buffer reference symbols:	
	Symbol	**Reference**
	%	Primary input buffer
	#	Active output buffer
	&*n*.	File buffer *n*
	&	Fast file buffer
	!*n*	Select register *n*
F	A forward location marked by the M command. Execution continues at the statement following the M command.	
B	The location of the last executed M command. Execution continues at the statement following the M command.	

Description

If a label cannot be found, an error message appears. If multiple labels exist, control is transferred to the first statement from the beginning that has the label.

If you use GO F or GO B and an M command cannot be found, an error message appears.

The GO B command returns to the last M command executed, which may not be the previous M command in the stack.

GOSUB

Use GOSUB to transfer control to a labelled ProVerb statement.

Syntax

```
GOSUB label
```

Parameter

The following table describes the parameter of the syntax.

Parameter	Description
label	A number specifying a labelled ProVerb statement. Execution continues at this statement.

Description

The statements in the local subroutine are executed up to an `RSUB` command, at which point control passes back to the statement following the GOSUB statement.

H

Use `H` to add a text string to the active output buffer.

Syntax

```
H [ text | ref ]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description	
text	The text string to put in the active output buffer.	
ref	A direct or indirect reference to a buffer or select register containing text to put in the active output buffer. You can use the following buffer reference symbols:	
	Symbol	**Reference**
	%	Primary input buffer
	#	Active output buffer
	&*n*.	File buffer *n*
	&	Fast file buffer
	!*n*	Select register *n*

Description

The `H` command inserts text at the end of the active output buffer, moving the pointer after the last character in the string. Blank spaces in the string, including leading and trailing spaces, are replaced by field marks. Consecutive blank spaces are replaced by a single field mark.

Primary output buffer

When the primary output buffer is active, the `H` command automatically supplies a carriage return. Blank spaces are substituted for field marks before the command goes to the UniVerse command processor.

When text is entered in the primary output buffer, a space between the command and the text adds the text as a new field. If there is no space, the text is appended to the last field.

Secondary output buffer

When putting text in the secondary output buffer, the last character of the string must be a carriage return symbol (<). No blanks or other characters can follow. Two carriage return symbols (<<) are ignored. They are not interpreted as two carriage returns.

IF

Use `IF` to execute commands conditionally. If the condition is true, `IF` executes a specified command; if the condition is false, the command is not executed.

Syntax

`IF [ N ]` *`condition command`*

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
`N`	Specifies a numeric comparison. It converts a numeric value up to the first nonnumeric to a binary equivalent, and compares the first value algebraically with the second. You can specify numeric values with a leading plus or minus sign and a decimal point. If one of the values is an empty string or if the first character in a value is nonnumeric, it is evaluated as 0.
condition	Specifies one of four conditions: [A-*command* | *ref* | E | S [*n*]]. This condition tests true if the specified value exists. #[A-*command* | *ref* | E | S [*n*]]. This condition tests true if the specified value does not exist. [A-*command* | *ref* | E | S [*n*]] *op* [*text* | *ref*]. This condition compares two values. [A-*command* | *ref* | E | S [*n*]] *mop lit* [V] *lit* . . . . Used with this condition, the command has two important functions: it allows comparing a single value with a number of different values, and it permits execution of one of several different commands based on which value is evaluated as true. A value mark (V) separates the values in the string. The string of multivalues can be made up of any combination of constants (within double quotation marks if a value contains embedded spaces), patterns (in parentheses), and direct or indirect references. If a reference is to a multivalued field, the contents of the field are inserted in the original multivalue string. See the following table for the syntax of the conditions.

Parameter	Description
command	The multiple command string consisting of a number of ProVerb commands separated by value marks. It can be used only with the equal operator. The command string associates each ProVerb command with a corresponding value in the comparison string. The values in the string are tested until the condition is true, then the corresponding ProVerb command is executed. Only one of the commands is executed. For instance, if the first value in the string is evaluated as true, the first command in the command string is executed. The rest of the values are not tested. If the first value in the string tests false, the second value is tested. If it tests true, the second command in the command string is executed. When the number of values in the string exceeds the number of commands, the last command serves as the corresponding command for the remaining values. If the number of commands exceeds the number of values, the remaining commands are ignored. The string cannot contain spaces because the ProVerb processor interprets what follows the blank space to be the ProVerb command that is executed if the condition is true. Surround values with spaces by double quotation marks (") and enclose patterns in parentheses. The `GO` and `GOSUB` commands have a special version for multiple command strings that permits using a single command with multiple labels. Each label is separated by a value mark and identifies a ProVerb statement associated with a particular value. IF Z = AVBVCV GO 100V150V200 If the value Z is equal to B, control is transferred to statement 150.

The following table describes the syntax of the *condition* parameter.

Parameter	Description	
A-command	A value supplied by the `A` command, which specifies a value in the active input buffer.	
ref	A value supplied by a direct or indirect reference to a buffer or select register. You can use the following buffer reference symbols:	
	Symbol	**References**
	%	Primary input buffer
	#	Active output buffer
	&*n*	File buffer *n*
	&	Fast file buffer
	!*n*	Select register *n*
`E`	Evaluates an error condition returned by a command that the UniVerse command processor executes. `E` must be used after a `P` command.	
`S`	Tests whether an active select list is available for processing. It is evaluated as true only if a `SELECT` or `SSELECT` sentence was executed by the UniVerse command processor.	
n	The number of a numbered select list.	

Parameter	Description	
op	Any singlevalued operator:	
	<	Less than
	>	Greater than
	[	Less than or equal to
	]	Greater than or equal to
mop	One of the two multivalued operators:	
	=	Equal to. This is equivalent to logical OR. The condition evaluates as true if the value is equal to any value in the multivalued string.
	#	Not equal to. This is equivalent to logical AND. The condition evaluates as true if the value is not equal to any value in the multivalued string.
lit	Can be any specified text, (*pattern*) in parentheses, or *ref* (direct or indirect reference).	
V	A value mark.	

Description

The `IF` command does the following:

- It tests whether a field has a value in it.
- It performs a relational operation on a value in a field.

If the condition is true, `IF` executes a ProVerb command. A common use of the `IF` command is to test for the existence of a field in the input buffer and then transfer control, as in the following example:

```
IF A3 GO 75
```

If the condition specifies a relational operation, the ASCII value of each character is compared, moving left to right. If characters do not match, the string with the higher ASCII equivalent is evaluated as greater.

Using the A command in the condition makes the value available for testing. It does not copy the value to the output buffer.

IH

Use `IH` to insert a text string in the active input buffer.

Syntax

```
I [ B ] H [ text | ref | [ ]\ ]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
B	Retains blank spaces in the moved data.
text	The text string for insertion. It should immediately follow the command.

<table>
<tr><th>Parameter</th><th colspan="2">Description</th></tr>
<tr><td>ref</td><td colspan="2">A direct or indirect reference to a buffer or select register containing the text string. The reference must immediately follow the command. You can use the following buffer reference symbols:</td></tr>
<tr><td></td><td>Symbol</td><td>Reference</td></tr>
<tr><td></td><td>%</td><td>Primary input buffer</td></tr>
<tr><td></td><td>#</td><td>Active output buffer</td></tr>
<tr><td></td><td>&n.</td><td>File buffer n</td></tr>
<tr><td></td><td>&</td><td>Fast file buffer</td></tr>
<tr><td></td><td>!n</td><td>Select register n</td></tr>
<tr><td></td><td colspan="2">You can specify either an input conversion (";iconv;") or an output conversion (":oconv:"). This conversion is applied to the string before it is put in the input buffer.</td></tr>
<tr><td>\</td><td colspan="2">Specifies an empty field. If the pointer is at the beginning of a field, <code>IH\</code> (no space before the backslash) makes the current field empty by removing all characters between the two field marks. <code>IH \</code> (with a space before the backslash) inserts a new, empty field at that point. If the pointer is located within a field, <code>IH</code> removes characters to the end of the field, shortening the field. <code>IH</code> replaces those characters with an empty field. If the pointer is at the end of the buffer, both versions of the command append a new empty field.</td></tr>
</table>

Description

The `IH` command inserts a text string into the current field at a position indicated by the input buffer pointer.

If the input buffer pointer is at the beginning of a field, this command replaces that field with the specified text. If the input buffer pointer is at some location within a field, the text is concatenated from that point without a leading field mark. If the input buffer pointer is at the end of the input buffer, the text is appended as a new field.

Blank spaces in the text string are interpreted as field delimiters and replaced with field marks. Consecutive blanks are replaced by a single field mark. Leading and trailing blank spaces in the text string are discarded. Use the `IBH` command if the blank spaces in a string do not delimit fields.

The `IH` command does not change the position of the input buffer pointer.

IN

Use `IN` to read input from the terminal and put it in the secondary input buffer.

Syntax

```
I [ B ] N [ c ]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
B	Retains blank spaces in the moved data.

Parameter	Description
c	The prompt character. The default setting is the last prompt specified. If none has been specified, a question mark (?) is the prompt, providing the proc was invoked from the UniVerse prompt. The prompt character remains the same until an IP, IBN, IN, IBP, or a BASIC PROMPT statement changes it. *c* is optional except when an indirect or direct reference is specified. Note that if the PROCPRMT configurable parameter contains a value of 0, the UniVerse BASIC PROMPT statement changes the ProVerb prompt. Any value other than 0 retains the ProVerb prompt, which can be changed only by the `IP, IBN, IN`, and `IBP` commands (the UniVerse BASIC PROMPT statement has no effect). For more information about configurable parameters configurable parameters, see *Administering UniVerse*.

Description

The `IN` command activates the secondary input buffer, and the terminal input overwrites its contents.

The `IN` command discards leading and trailing blanks from the terminal input and replaces embedded blank spaces with field marks to enter each “word” in a field. If the user presses Return from the terminal, the buffer is set to an empty state.

If you want to treat input as a single field, use the `IBN` command.

After executing this command, the pointer is positioned at the beginning of the secondary input buffer.

The `IS` command is a synonym for the `IN` command.

IP

Use `IP` to accept input from the terminal and assign it to any buffer or register.

Syntax

```
I [ B ] P [ B ] [ c [ ref ] ]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
B	Retains blank spaces in the moved data.
c	The prompt character. The default is the last prompt specified. If none has been specified, a question mark (?) is the prompt, providing the proc was invoked from the UniVerse prompt. The prompt character remains the same until an IP, IBN, IN, IBP, or a UniVerse BASIC PROMPT statement changes it. *c* is optional except when an indirect or direct reference is specified. Note that if the PROCPRMT configurable parameter contains a value of 0, the UniVerse BASIC PROMPT statement changes the ProVerb prompt. Any value other than 0 retains the ProVerb prompt, which can be changed only by the `IP, IBN, IN`, and `IBP` commands (the UniVerse BASIC PROMPT statement has no effect).

Parameter	Description	
ref	A direct or indirect reference to a buffer or select register containing the input. You can use the following buffer reference symbols:	
	Symbol	**Reference**
	%	Primary input buffer
	#	Active output buffer
	&*n*.	File buffer *n*
	&	Fast file buffer
	!*n*	Select register *n*

Description

The input replaces the field or fields in the buffer or register indicated by a direct or indirect reference. If no reference is specified, the input replaces the field marked by the pointer in the active input buffer.

`IP` discards leading and trailing blanks and replaces embedded blank spaces with field marks. An input field replaces the designated field in the buffer, and succeeding fields are created if the input contains more than one field. The input is put in the active input buffer at the field indicated by the pointer, unless a direct or indirect reference indicates another buffer or register. If a direct reference specifies a field in a file buffer, input fields replace consecutive fields in the file buffer.

If the user presses Enter at the terminal, one of the following occurs:

- If the proc is a PQN proc, the buffer is set to an empty state.
- If the proc is a PQ proc, the value currently in the input buffer is retained.

If you want to treat input as a single field, use the `IBP` command.

The `IP` command does not change the position of the input buffer pointer.

IT

Use `IT` to read a tape record into the primary input buffer.

Syntax

```
IT [ C ] [ A ] [ n ]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
C	Converts tape data from EBCDIC to ASCII.
A	Inverts the high-order bit.
n	Indicates the MTU number from which to read.

Description

The `IT` command activates the primary input buffer and overwrites its contents with input from a tape record. This command does not convert blank spaces to field marks as it reads data from the tape. After executing the `IT` command, the pointer is positioned at the beginning of the primary input buffer.

If the tape record is not found, the command simply clears and resets the input buffer.

L

Use `L` to send formatted output to the printer.

Syntax

```
L [ N | C | E | HDR [ , header.elements ] … | [ n | ref ] [ ,
other.elements ] … ]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
N	Redirects output to the terminal for any L statement to follow in the proc. This element is used for debugging and cannot appear with other elements in an L statement.
C	Closes the print output file and sends it to the printer. This element cannot be used with the HDR element.
E	Generates top-of-form eject. This element cannot be used with the HDR element.
HDR	Prints a heading at the top of each page. HDR must be the first element in a statement.

header.elements can be any of the following:

Parameter	Description	
P	Generates page numbers for the heading.	
T	Supplies the current date and time for the heading.	
Z	Resets current page number to 0.	
'*text*'	The text string to print.	
ref	A direct or indirect reference to a buffer or select register containing the value to print. You can use the following buffer reference symbols:	
	Symbol	**Reference**
	%	Primary input buffer
	#	Active output buffer
	&*n*.	File buffer *n*
	&	Fast file buffer
	!*n*	Select register *n*
(*col*)	Sets the horizontal position at column *col* in the current line.	
n	Specifies *n* blank lines. This header element must be first in the statement.	
ref [;*iconv*;]	Uses direct or indirect reference to obtain value to print. An input conversion can be applied to the value before it is displayed. (See the `ICONV` function in *UniVerse BASIC*.)	
ref [:*oconv*:]	Uses direct or indirect reference to obtain value to print. An output conversion can be applied to the value before it is displayed. (See the `OCONV` function in *UniVerse BASIC*.)	

other.elements can be any of the following:

Parameter	Description
+	Specifies no carriage return at the end of output.
'*text*'	The text string to print.
ref	A direct or indirect reference to a buffer or select register containing the value to print.
(*col*)	Sets the horizontal position at column *col* in the current line.

Description

The `L` command specifies complex formatting of printer output. Following the `L` command and a blank space, the statement contains one or more elements, each separated by a comma.

You can extend the statement to the following line by ending the current line with a comma and beginning the next line with an element (without the `L` command).

You can use a literal or resolve the numeric value for *n* by direct or indirect buffer reference. If the value for *n* is resolved to be nonnumeric, it is printed as a literal.

M

Use `M` to mark a location in a proc. Marking a location is an alternative to using a statement label.

Syntax

```
M
```

Description

In a proc you can transfer control to the marked location using the `GO F` and `GO B` commands. If a proc is to execute a number of loops, transferring to a marked location rather than a label speeds up the search for the location, because a label must be searched for from the beginning of the proc.

MV

Use `MV` to copy (move) data from one location to another. These locations can be input buffers, output buffers, file buffers, or select registers. You can use direct or indirect references to specify the source and destination of the moved data. You can also move literal strings to the destination.

Syntax

```
MV destination source
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description	
destination	The location where data movement is directed. Refer to this location indirectly or directly by buffer and field number. If the number referring to a field is greater than the number of existing fields in the input, output, or file buffers, empty fields are allocated to provide for that field.	
source	One or more values copied to *destination*. Refer to these values by buffer and field number or specify them as literal strings surrounded by double quotation marks. If you specify two double quotation marks with no space in between as the source, an empty string is sent to the destination. If the source consists of more than one value, separate each value by either a comma (,) or an asterisk (*).	
	,	A comma between multiple values copies each value to *destination* as a separate field. If a comma immediately follows a comma, the location in the destination buffer is advanced and the value at that location is retained. You can use a comma by itself or a series of commas to successively skip the fields you do not want `MV` to change.
	*	An asterisk between multiple values concatenates the values, making one field to be sent to *destination*. You can also use an asterisk after a file buffer reference (&) to copy all the remaining fields in the file buffer to the destination. It must be the last item in a string of multiple values. A numeric value immediately following the asterisk (**n*) specifies movement of *n* of the remaining fields from the previously referenced file buffer. The (**n*) can appear anywhere in a string of multiple values.
	_	An underscore (_) as the last item in a string of multiple values specifies that the end of the destination buffer is the end of the string being sent and that any fields existing beyond that should be discarded.

Description

The `MV` command does not change the values obtained at the source. Nor does the moved field replace an existing field. The field is inserted at the specified position.

MVA

Use `MVA` to move data from the source to the destination and store it as values in a multivalued field.

Syntax

```
MVA destination source
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
destination	The location where data movement is directed. Reference this location indirectly or directly by buffer and field number. The value moved to *destination* is stored in ascending ASCII order. If the value already exists, it is not duplicated. If you specify an input buffer as the destination, the pointer is positioned at the beginning of the specified field.
source	The location of the value that is copied and moved to the destination. Reference this value by buffer and field number. Do not use double quotation marks when you specify literal strings. If the value is multivalued, it remains multivalued at the destination, possibly creating duplicate values and upsetting the ascending ASCII sequence.

MVD

Use `MVD` to delete a value from a multivalued field in the active input or output buffer. Only the first occurrence of the value is deleted from the multivalued field.

Syntax

```
MVD destination source
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
destination	The location of the multivalued field. Reference this location indirectly or directly by buffer and field number. If you specify the active input buffer, the pointer is positioned at the beginning of the specified field.
source	The value to be deleted from the multivalued field. Reference this value by buffer and field number. Do not use double quotation marks when you specify literal strings. If *source* is multivalued, the command is ignored.

Description

If values in the multivalued field are not in ascending ASCII sequence, the `MVD` command may not work properly.

O

Use `O` to output a text string to the terminal.

Syntax

```
O [ text ] [ + ]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
text	The string to display on the terminal.
+	Specifies that no carriage return be executed after displaying the text.

Description

You can use `O` before an `IP` command to display the text that prompts the user for input to the `IP` command.

P

Use `P` (Process) to execute the UniVerse command in the primary output buffer. Any data in the secondary output buffer is used in response to interactive prompts. After processing, both output buffers are cleared and the stack is turned off.

Syntax

```
P [ P | H | X | W ] … [ Ln ]
```

Parameters

You can use any combination of the following parameters:

Parameter	Description	
P	Displays the contents of the output buffer before processing.	
H	Suppresses terminal output from the UniVerse command.	
X	Prevents returning control to the proc after execution.	
W	Displays the contents of the output buffer and prompts the user with a ?. The user can enter the following at the ? prompt:	
	`Y`	Go ahead with processing the contents of the output buffers.
	`S`	Do not process, but clear the output buffers and continue executing the proc.
	`N`	Do not process and do not continue executing the proc. This response returns the user to the UniVerse prompt (>).
L*n*	Sets an execution lock. *n* is the number (0 through 63) of an execution lock. An execution lock lets only one user at a time execute the same proc.	

Description

The `P` command with no options sends the contents of the two output buffers to the UniVerse command processor.

When a `P` command is executed, control passes from the ProVerb processor to the UniVerse command processor until processing finishes.

Q

Use `Q` to stop execution of a proc and return control to the command processor.

Syntax

```
Q [ text ]
```

Parameter

The following table describes the parameter of the syntax.

Parameter	Description
text	The message to display on exiting.

Description

The Q command displays a message and terminates the current proc and any preceding procs. The user returns to the UniVerse prompt (>) or to the process that started the proc. You typically use the Q command to terminate nested procs. The Q command is like the ABORT statement in UniVerse BASIC.

RI

Use RI to clear the input buffers and activate the primary input buffer.

Syntax

```
RI [ f | (col) ]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
f	A field number. The primary input buffer is cleared from this field to the end. The pointer is reset at the end of the buffer (after the *f*-1st field). If *f* is less than 2, the entire buffer is cleared. If *f* is greater than the number of fields in the buffer, the pointer is positioned at the end of the buffer.
(*col*)	A column position. The primary input buffer is cleared from the character in this column to the end. The end of the buffer is at this column where the pointer is now positioned. If *col* is less than 2, the entire buffer is cleared. If *col* exceeds the number of columns in the buffer, the pointer is reset at the end of the buffer.

Description

The RI command completely clears both input buffers. The pointer is reset at the beginning of the primary input buffer.

RO

Use RO to clear the output buffers and activate the primary output buffer.

Syntax

```
RO
```

Description

The RO command empties the two output buffers and resets the pointers at the beginning of each buffer. RO puts the output buffers in their initial state, with the primary buffer active.

Note: The P command also clears the primary and secondary output buffers automatically and activates the primary output buffer.

RSUB

Use RSUB to transfer control back to a GOSUB statement.

Syntax

```
RSUB [ n ]
```

Parameter

The following table describes the parameter of the syntax.

Parameter	Description
n	Specifies that control be returned to the *n*th statement following the GOSUB statement. Normally RSUB returns to the first statement following the GOSUB.

Description

Put an RSUB command at the end of a local subroutine to return control to the GOSUB statement that called the subroutine. Execution continues at the first or *n*th statement after the GOSUB command.

RSUB is ignored if a GOSUB statement was not previously executed.

RTN

Use RTN to return control to a calling proc.

Syntax

```
RTN [ n ]
```

Parameter

The following table describes the parameter of the syntax.

Parameter	Description
n	Specifies that execution should resume at the *n*th statement after the [] command. Normally it resumes at the first statement following the [ ] command.

Description

Put an RTN command at the end of an external subroutine to return to the proc that called the subroutine. Execution continues in the calling proc at the first or *n*th statement following the [ ] command.

If an RTN statement is executed in a proc that has not been called by the [ ] command, the proc exits to the UniVerse prompt (>).

S

Use S to set the pointer of the active input buffer or to select a field in a numbered register.

Syntax

```
S [ f | ref | (col) ]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
f	A field number. The pointer is set just after the field mark at the beginning of this field. If *f* is less than 2, the pointer is set at the beginning of the buffer. If *f* is greater than the number of fields in the buffer, the pointer is positioned at the end of the buffer.
ref	A direct or indirect reference to a field in the input buffer (%*r*) or in a select register (!*r*).
(*col*)	A column position. The pointer is set at this column in the primary input buffer. If *col* is less than 2, the pointer is set at the beginning of the buffer. If *col* exceeds the number of columns in the buffer, the pointer is reset at the end of the buffer.

SP

Use SP to activate the primary input buffer. SP turns off the secondary input buffer.

Syntax

```
SP
```

Description

When a proc begins, the primary input buffer is active. After the RI command is executed, the primary input buffer is again made active.

SS

Use SS to activate the secondary input buffer. SS turns off the primary input buffer.

Syntax

```
SS
```

Description

When a proc begins, the primary input buffer is active. After the `RI` command is executed, the primary input buffer is again made active.

STOFF

Use `STOFF` (Stack off) to activate the primary output buffer. `STOFF` turns off the stack, or secondary output buffer.

Syntax

```
STOFF
```

```
ST OFF
```

Description

When a proc begins, the primary output buffer is active. After such commands as `P` and `RO` are executed, the primary output buffer is again made active. Use the `STOFF` command when the stack is on and the primary output buffer is inactive.

STON

Use `STON` (Stack on) to activate the secondary output buffer, or stack.

Syntax

```
STON
```

```
ST ON
```

Description

Use `STON` when you want to move data to the secondary rather than to the primary output buffer. The secondary output buffer, or stack, holds responses for an interactive processor. The processor is invoked by a command in the primary output buffer.

When a proc begins, the stack is off. The stack is emptied and turned off following such commands as P and RO.

Use the STOFF command to reactivate the primary output buffer.

T

Use `T` to produce formatted screens on a terminal. `T` can control the position of the cursor, clear the screen, output text for user prompts, and display nonkeyable character codes.

Syntax

```
T element [ , element … ]
```

Parameters

element can be any of the following:

Element	Description	
text	The text string to display.	
ref [; *iconv* ;]	A direct or indirect reference to a buffer or select register containing the value to display. You can apply an input conversion to the value before displaying it. (See the ICONV function in *UniVerse BASIC*.) You can use the following buffer reference symbols:	
	Element	**Reference**
	%	Primary input buffer
	#	Active output buffer
	&*n*.	File buffer n
	&	Fast file buffer
	!*n*	Select register *n*
ref [: *oconv* :]	A direct or indirect reference to get the value to display. You can apply an output conversion to the value before displaying it. (See the OCONV function in *UniVerse BASIC*.)	
(*col*,*row*)	Positions the cursor at column *col* and row *row*. Either value may be obtained by direct or indirect reference.	
(*col*)	Positions the cursor at column *col* in the current line.	
(,*row*)	Positions the cursor at row *row* of the current column.	
+	Specifies no carriage return at the end of output.	
B	Sounds the bell on the terminal.	
C	Clears the screen.	
D	Causes a delay. Used with L and T to cause the display to blink on and off.	
I*r*	Specifies that an integer from 0 through 255 be converted to its equivalent ASCII character.	
L	Closes a loop started by a T element.	
S*n*	Displays *n* spaces.	
S*ref*	Displays the number of spaces indicated by an indirect or direct reference.	
T	Specifies the start of a loop. The elements between the T element and L element are executed three times.	
U	Moves the cursor up one line.	
X*ref*	Converts the hexadecimal value from 00 through FF obtained by a direct or indirect reference to its equivalent ASCII character.	
(-1)	Clears the screen and puts the cursor in home position.	
(-2)	Puts the cursor in home position.	
(-3)	Clears the screen from the cursor to the bottom of the screen.	
(-4)	Clears to the end of the line from the cursor position.	
(-5)	Starts blinking while data that follows is output.	
(-6)	Stops the blinking display.	
(-7)	Protects data on screen from overwriting.	

Element	Description
(-8)	Cancels screen protection.
(-9)	Moves the cursor back one space.
(-10)	Moves the cursor up one line.

Description

The `T` command specifies complex formatting of output screens. Following the `T` command and a blank space, the statement contains one or more elements, each separated by a comma.

You can extend the statement to the following line by ending the current line with a comma and beginning the next line with an element (without the `T` command).

You can incorporate comments into alphabetic elements. For example, you can enter `C` as `CLEAR` or `D` as `DELAY`.

TR

Use `TR` as a debugging tool to invoke a trace of a proc during execution. Each command is displayed on the terminal as it is processed.

Syntax

```
TR ON | OFF
```

Qualifiers

The following table describes each parameter of the syntax.

Parameter	Description
ON	Turns on proc trace.
OFF	Turns off proc trace.

Description

ProVerb statements parsed more than once are displayed for each parsing.

U

Use `U` to access BASIC programs from a proc.

Syntax

```
U text
```

Parameter

The following table describes the parameter of the syntax.

Parameter	Description
text	Refers to a cataloged BASIC program named $*text*. Typically, *text* is a four-digit hexadecimal number, but it can be any string of characters or numerals.

Description

The interface to the UniVerse BASIC program is described here so that you can write your own user exit routines. For information about UniVerse user exit codes used in ProVerb, see *UniVerse Guide for Pick Users*.

For proc access, you should set up your UniVerse BASIC program to accept the following arguments:

Argument	Description
item	The actual proc being run. Remove set to next executable line.
ibn	0 = primary input buffer is active. 1 = secondary input buffer is active.
pib	Primary input buffer.
sib	Secondary input buffer.
ip	Pointer to current input buffer. 0 = first character.
obn	0 = primary output buffer is active. 1 = secondary output buffer is active.
pob	Primary output buffer.
sob	Secondary output buffer.

X

Use X (Exit) to stop execution of a proc and return control to the calling environment.

Syntax

```
X [ text ]
```

Parameter

The following table describes the parameter of the syntax.

Parameter	Description
text	The message to display on exiting.

Description

The X command displays a message and exits the proc. The user returns to the UniVerse prompt (>) or to the calling proc if the current proc is called from another proc as a subroutine.

You typically use the X command to display a message and exit when an error condition occurs. The X command is like a STOP statement in UniVerse BASIC.

+

Use + to add an integer to a numeric field.

Syntax

```
+ n
```

Parameter

The following table describes the parameter of the syntax.

Parameter	Description
n	A positive integer to be added or subtracted. Leading zeros are ignored.

Description

The + command adds *n* to the field indicated by the pointer in the active input buffer.

If the pointer is at the end of the input buffer, the command has no effect.

The field must contain a numeric value. If it is nonnumeric, its value is assumed to be 0. Negative values in the input buffer are permitted. A plus sign preceding any value is replaced by a 0.

–

Use – to subtract an integer from a numeric field.

Syntax

```
– n
```

Parameter

The following table describes the parameter of the syntax.

Parameter	Description
n	A positive integer to be added or subtracted. Leading zeros are ignored.

Description

The – command subtracts *n* from the field indicated by the pointer in the active input buffer.

If the pointer is at the end of the input buffer, the command has no effect.

The field must contain a numeric value. If it is nonnumeric, its value is assumed to be 0. Negative values in the input buffer are permitted. A plus sign preceding any value is replaced by a 0.

( )

Use `( )` to transfer control to another proc. Unlike the `[ ]` command, control does not return to the originating proc.

Syntax

```
( [ DICT ] filename [ record.ID ] ) [ label ]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
DICT	Specifies that the proc is stored in the file dictionary.
filename	The name of the file that contains the proc.
record.ID	The name of the proc record. If you do not specify *record.ID*, the value of the current field designated by the input buffer pointer is used.
label	Transfers control to the labelled statement in the proc. If you omit *label*, execution begins at the first statement.

Description

The `( )` command lets you store procs outside the VOC file. It also links together a series of procs, because a series of smaller procs can function more efficiently than one large proc.

The contents of the buffers and registers do not change during the transfer from one proc to another.

`( )` works like the `CHAIN` command in UniVerse BASIC.

[ ]

Use `[ ]` to transfer control to another proc. Unlike the `( )` command, control can return to the calling proc.

Syntax

```
[ [ DICT ] [ filename ] [ record.ID ] ] [ label ]
```

Parameters

The following table describes each parameter of the syntax.

Parameter	Description
DICT	Specifies that the proc is stored in the file dictionary.
filename	The name of the file that contains the proc. If you do not specify *filename*, control passes to the line in the current proc specified by *label*.
record.ID	The name of the proc record. If omitted, the value of the current field designated by the input buffer pointer is used.
label	Transfers control to the labelled statement in the proc. If you omit *label*, execution begins at the first statement.

Description

The `[ ]` command lets a proc call other procs as external subroutines. You can return control to the calling proc by including the `RTN` command at the end of the external subroutine. However, because the `[ ]` command takes longer to process, you should use the `( )` command unless you need to return to the calling proc.

The contents of the buffers and registers do not change during the transfer from one proc to another.

If there are no more statements to be executed in the originating proc when control is passed back, the proc terminates. You can call any number of external subroutines.

`[ ]` works like the CALL statement and EXECUTE statement in UniVerse BASIC.

Appendix A: ProVerb commands by function

This appendix summarizes the ProVerb commands described in detail in ProVerb commands, on page 22. The commands are grouped according to use as follows:

- Buffer selection commands
- Buffer pointer commands
- Moving parameter commands
- User input commands
- Displaying and printing commands
- Transferring of control commands
- File I/O commands
- Miscellaneous commands

Buffer selection commands

Command	Synonym	Description
S	NS	Selects the active input buffer.
SP	NSP	Selects the primary input buffer.
SS		Selects the secondary input buffer.
STOFF	ST OFF	Selects the primary output buffer.
STON	ST ON	Selects the secondary output buffer.

Buffer pointer commands

Command	Synonym	Description
B	NB	Moves the input buffer pointer backward.
F	NF	Moves the input buffer pointer forward.
S	NS	Sets the active input buffer pointer.
SP	NSP	Sets the primary input buffer pointer.
SS		Sets the secondary input buffer pointer.

Moving parameter commands

Command	Synonym	Description
A	NA	Copies a field from the input to the output buffer.
MV		Copies a field from buffer to buffer.
MVA		Copies data from buffer to buffer as multivalues.

User input commands

Command	Synonym	Description
IN	IS, NIN, NIS	Enters data in the secondary input buffer.
IP	NIP	Reads input from a terminal.

Displaying and printing commands

Command	Synonym	Description
D		Displays input buffer contents.
O		Outputs text to the terminal.
T		Formats terminal output.

Transferring of control commands

Command	Synonym	Description
GO	G, GOTO	Transfers control.
GOSUB	GS	Executes a local subroutine.
IF		Alters execution based on a condition.
()		Executes another proc. Does not return to the calling proc.
[]		Executes an external subroutine. Returns to the calling proc.

File I/O commands

Command	Synonym	Description
F-CLEAR		Clears a file buffer.
F-DELETE		Deletes a record from a file.
F-FREE		Releases an update lock.
F-OPEN		Opens a UniVerse file.
F-READ		Reads a record into a file buffer.
F-UREAD		Sets an update record lock.
F-WRITE		Writes a record from a file buffer.
FB		Reads to a fast buffer.
FBU		Sets an update record lock.

Miscellaneous commands

Command	Synonym	Description
BO		Backs up the output buffer pointer.

Command	Synonym	Description
C	*	Specifies a comment.
F;		Evaluates arithmetic expressions.
H		Adds text to the output buffer.
IH		Inserts text into the input buffer.
IT		Reads the tape to the primary input buffer.
L		Formats output for the printer.
M		Marks a location.
MVD		Deletes multiple values.
P		Processes output buffers.
Q		Exits the proc and returns to the command processor prompt.
RI		Clears the input buffers.
RO		Resets the output buffers.
RSUB		Returns from the local subroutine.
RTN		Returns from the external subroutine.
TR		Traces execution.
U		Calls a user exit subroutine.
X		Exits the proc and returns to the calling process.
+		Adds the integer.
-		Subtracts the integer.