



**QUICK
REFERENCE
GUIDE**

UNIFACE **SIX**

Quick Reference Guide

UNIFACE V6.1

10110006100
Revision 0
30 April 1995
QRG

UNIFACE V6.1

Quick Reference Guide

Revision 0

© Compuware Corporation, 1995.

Rights to the contents of this document

This document contains proprietary information belonging to Compuware Corporation. It may not be reproduced without written permission. Disclosure of the contents of this document to third parties is strictly prohibited.

Trademarks

UNIFACE™, PolyServer™ and Universal Presentation Interface™ are trademarks of Compuware Corporation. This document contains references to third party hardware and software products and manufacturers. For a complete list of trademarks, see the *Using UNIFACE Six* document or *Release Notes*.

About this guide

The information in this guide is correct at the time of printing.

Reactions

Your suggestions and comments about this version and the documentation are highly valued. Send, fax or email this information to:

Compuware Technology B.V.

Technical Publications

Post Box 12933

1100 AX Amsterdam

The Netherlands

tel.: +31 (0)20 3116 222

fax: +31 (0)20 3116 200

email: techpubs@uniface.nl

Contents

- 1 Function keys and Super keys
- 2 Profile characters
- 3 Naming objects
- 4 Video attributes
- 5 Command line switches
- 6 Assignment files
- 7 Assignment settings
- 8 Models of the Repository
- 9 Structure editor functions
- 10 Interface definitions
- 11 Syntax definitions
- 12 Layout definitions
- 13 Handling data in Proc

- 14 Proc statements
- 15 Proc functions
- 16 Debugger commands
- 17 Trigger mnemonics
- 18 I/O messages
- 19 Widgets
- 20 Loading images
- 21 Search order for global objects







Chapter 1 Function keys and Super keys

This chapter summarizes function key and Super key combinations. The GOLD key combinations used as profile characters are described in chapter 2 *Profile characters*.

The information in this chapter is generic to all keyboards that support the GOLD key. For more details of keyboard mapping, including device-specific information, see the *Keyboard layouts* chapter in the ► *Using UNIFACE Six* document and the *Keyboard Translation Tables* chapter in the ► *UNIFACE Reference Manual*.


Note: Except for the GOLD GOLD combinations shown in the following table, if you press the GOLD key twice in succession, the second press cancels the first GOLD request.

Function keys

Press	Followed by	To
 	Z . ,	^QUICK_ZOOM deselect an object find profile
		insert named file
		remove named file

Key combinations defined as function keys.

part 1 of 2

Press	Followed by	To
	A	^ACCEPT
	B	toggle Bold mode
	C	display the Command menu
	D	^DETAIL
	E	^ERASE
	F	^FRAME
	G	^CLEAR
	H	^HELP
	I	toggle Italic mode
	J	compose a character
	K	^KEYBOARD_HELP
	L	^PULLDOWN
	M	^MESSAGE
	N	^RETRIEVE_SEQUENTIAL
	O	toggle Overstrike mode
	P	^PRINT
	Q	^QUIT
	R	^RETRIEVE
	S	^STORE
	T	toggle The ruler
U	toggle Underline mode	
V	toggle View mode	
W	work with the SQL Workbench	
X	toggle the session panel	
Y	^SWITCH_KEYBOARD	
Z	^ZOOM	
.	select an object	
,	find an occurrence	
Y	^SWITCH_KEYBOARD	
Z	^ZOOM	
.	select an object	
,	find an occurrence	

Key combinations defined as function keys.

part 2 of 2

Super keys

In the following table, each bullet (•) indicates a combination that is defined as a Super key and each empty cell indicates a combination that has no effect. The first character sets a mode; the second character applies that mode to a suitable object.

		GOLD		SPACE		followed by ...									
First character (mode)	Second character (object)	Character	Word	Line	selected block	Data (text window)	Field	Occurrence	Entity window	Screen	Page	named file	none *		
		C	W	L	.	D	F	O	E	S	P	X			
A	Add							•							
I	Insert	•	•	•	•	•	•	•					•		
R	Remove	•	•	•	•	•	•	•					•		
T	Top or first	•	•	•		•		•		•			•		
B	Bottom or last	•	•	•		•		•		•			•		
N	Next	•	•	•		•	•	•	•	•	•		•		
P	Previous	•	•	•		•	•	•	•	•	•		•		
	none **	•	•	•		•	•	•	•	•					

Key combinations defined as Super keys.

Table notes:

* Only the Next mode (the default mode) or the Previous mode remains current after you apply it to an object. For example, after using 'GOLD SPACE P O' to perform the ^PREVIOUS ^OCCURRENCE function, you can repeat the function continually during the current session by typing 'GOLD SPACE O' (without the 'P') as necessary.

** Current mode (Next or Previous) applies.

In some cases you can use a Super key to set a mode without applying it to an object immediately (by omitting the second character). To apply such a mode to an object—where applicable—omit the first character from the next Super key that you use in the same edit session.

Chapter 2 Profile characters

See the ► *Using UNIFACE Six* document for further information.

Character	Meaning
GOLD *	Zero through <i>n</i> characters.
GOLD ?	Any single character.
GOLD =	Equals (if without a qualifier: 'is null').
GOLD >	Greater than.
GOLD <	Less than.
GOLD !	Logical NOT.
GOLD &	Logical AND.
GOLD	Logical OR.

Profile characters.

GOLD * and GOLD ? can be used only to match characters in a string; this means they cannot be used for matching numeric values.

Chapter 3 Naming objects

This chapter summarizes the rules for naming Repository objects (that is, objects stored in the Application Objects Repository). For general information about defining and using Repository objects, see the ► *UNIFACE Reference Manual*. For detailed information about defining and using:

- Model-level objects, see the ► *Designers' Guide*.
- Form-level objects, see the ► *Developers' Guide*.
- Version control objects, see the ► *Project Managers' Guide*.

Caution: Your DBMSs, networks and operating systems have their own naming rules, which in some cases can affect your applications. Read your DBMS, network and operating system documentation to familiarize yourself with the rules and syntax that apply to the names you define.

Naming tables and files created for entities

UNIFACE uses uppercase to name DBMS tables and files created for entities. If this causes problems, you can use assignments to direct entities to the DBMS table or file name that you prefer.



Naming Repository objects

The following table shows the rules that apply to the names of Repository objects:

Repository object	Maximum length (characters)	Notes (see page 3-3)
Application model	32	1, 2, 3
Break frame	32	1, 2, 3, 4, 5
Device translation table	16	1, 2
Entity	32	1, 2, 3, 4, 5, 6, 7, 8, 9
Entity interface template	32	1, 2, 3, 10
Field	32	1, 2, 3, 4, 5
Field interface template	32	1, 2, 3, 10
Field layout template	32	1, 2, 3, 10
Field syntax template	32	1, 2, 3, 10
Field template	32	1, 2, 3, 4, 10
Form	16	1, 2, 3, 5, 11, 12, 13
Glyph	16	1, 2
Keyboard translation table	16	1, 2
Language	3	1, 2
Library	16	1, 2, 14
Menu	16	1, 2
Message and help text	16	1, 2
Panel	16	1, 2
Print job model	16	1, 2, 3
Proc label	16	1, 2
Proc module	16	1, 2, 14
Start-up shell	16	1, 2, 3, 5, 11, 12
Variable (global and local)	32	1, 2, 3, 15
Version control branch	3	16
Version control release	16	1, 2
Version control version	16	17

Rules for naming Repository objects.

Notes concerning the previous table

- Any combination of the following characters is valid:
 - Letters, A–Z.
 - Digits, 0–9.
 - Underscore (_).
- Do not use any of the reserved words shown on page 3-4.
- The first character must be a letter (A–Z).
- The name can include—but must not begin with—one dollar sign (\$).
- Avoid names starting with the letter ‘U’ to prevent conflicts with UNIFACE objects.
- Avoid names starting with the letter ‘O’ to prevent conflicts with overflow tables or files.
- Each entity maps to a table or file with the same name in the target DBMS. If the entity name is longer than the target DBMS allows, you can use assignments to redirect the entity to a table or file with a shorter name (as described in chapter 6 *Assignment files*).
- Depending on how the fields in the entity are defined, UNIFACE can create an overflow table or file and use the entity name preceded by the letter ‘O’ to generate a name for it. The name of the overflow table or file is also restricted to 32 characters, so limit the name of the entity to 31 characters if you expect an overflow table or file to be created. (For details of overflow tables, see the ► *DBMS Specific Guide*.)
- If the target DBMS or operating system requires table or file names with fewer than 32 characters, you can consider these options:
 - Use assignments to redirect the entity to a table or file with a shorter name. (See chapter 6 *Assignment files*.)
 - Establish a requirement for shorter entity names in your site’s standards and guidelines.
- Avoid using the names of templates from the meta dictionary.
- Consider the limits of the target operating system when you select the name of a start-up shell or form, because when you compile a start-up shell or form, UNIFACE creates a file named *shell.apr* or *form.frm*.
- Avoid defining a start-up shell and a form with the same name.
- Avoid names starting with UU, UV and USYS to prevent conflicts with UNIFACE forms.
- Avoid using the library name USYS. (Instead, use the library SYSTEM_LIBRARY for system-wide objects.)
- Do not use library USYS for global variables.
- Only letters (A–Z) are valid.
- The format of the name must be *n.n.n*, where each *n* is any number in the range 0–99.

Repository object	Reserved words			
All	Names of: Repository entities. Tables or files associated with Repository entities. Overflow tables or files associated with Repository entities. Proc statements. Proc special functions. Anything considered by your DBMSs, networks and operating systems to be a reserved word. For details, and to validate syntax of names, see the appropriate documentation.			
Application model	APS DICT	FRM PRINTER	SYSENV TEXT	UVCS
Entity	HEADER	TRAILER	UNIS	
Field interface template	ATEST B193 B194 B195 B196 B197 B198 B199 B200 B201 B202 B203 B204 B205	B206 B207 B208 B209 B210 B211 B212 B220 B221 B221SEP B222 B224 B225 B226	B227 B228 B229 B230 B231 DESCR EMPTY F1 F128 F16 F192 F2 F3 F31	F32 F4 F6 F64 F8 FIX40 PREFVAL STAMP TEXT TIMESTAMP UCOMMENT UDATE UTIME V2
Field layout template	DATE DATEDMY	INV80 NOINV340	TESTD TIME	TIMEHM
Field syntax template	ACCNAM C3 DISPROM DISPROMPT DTYP DTYPREG ENTITYNAME FIELDNAME FIELDNAMEKEY	FLDNAME FORMNAMEKEY INHERIT LABEL LIBNAME LIBRNAME MAN2 MODELNAME MODELNAME0	NAME032 NAME16 NAME32 NOMANALL NUM2 NUMS3 NUMSX PRNUM SCHEMANAME	SUBFLD TESTC3 TEXTLAN TEXTNAME YN YNB

Reserved words.

Chapter 4 Video attributes

This chapter summarizes the information about video attributes in the ► *Developers' Guide* and the ► *UNIFACE Reference Manual*.

You can customize video attributes for UNIFACE objects at:

- Application level, by using assignment settings.
- Form level, by changing the properties of the appropriate forms.
- Object level, by specifying any of the codes shown in the following table either as defaults—during the installation process—or in Proc statements.

Code	Effect on corresponding object
BOR	Show border (ignored if \$GUI=\$CHR).
BLI	Change brightness sporadically to make object blink (flash).
BRI	Increase brightness permanently.
INV	Invert colors (valid only in combination with system colors, that is, when COL=0; ignored if COL≠0).
UND	Underline.
COL= <i>n</i>	Set colors, where <i>n</i> is the sum of the appropriate values from the following table.

Video attribute codes.

Color	Foreground value	Background value
System	0	0
Blue	8	1
Green	16	2
Cyan	24	3
Red	32	4
Purple	40	5
Yellow	48	6
White	56	7

Values for defining colors (see COL=n in the previous table).

Example

To define white letters on a blue background, specify COL=57 (where 57 is the sum of the values for a white foreground and blue background). If you have problems displaying colors, try using the assignment setting \$SWAP_COLORS, as described in the ► *UNIFACE Reference Manual*.

Chapter 5 Command line switches

This chapter summarizes the command line switches that you can use when you start a UNIFACE application. See the *Command line switches* chapter in the ► *UNIFACE Reference Manual* for complete information.

Points to remember

- When you combine command line switches and sub-switches, these must appear on the command line (in any order) *before* any parameters.
- Wherever a *datetime* parameter appears, it should have the following format:
`dd-mm-yyyy:hh:nn:ss`
All the separators are optional; all the digits must be present.

Name	<code>/all</code> Compile all global objects.
Use	UNIFACE Six development environment application.
Synopsis	<code>/all (/inf /war) (/lis) (/aft=datetime) (/bef=datetime) (library)</code>
Notes	<ul style="list-style-type: none"> • Using <code>/all</code> is equivalent to using <code>/obj</code>, <code>/frm</code> and <code>/app</code>. • <code>/all</code> is often used in combination with <code>/cro</code> and <code>/zip</code>.

Name /app
Compile and link start-up shells.

Use UNIFACE Six development environment application.

Synopsis /app {/inf | /war} {/lis} {/aft=*datetime*} {/bef=*datetime*} (*shell*)

Name /asn
Specify the assignment file used by the application.

Use All UNIFACE applications.

Synopsis /asn=*file_name*

Name /bat
Start the application in batch mode.

Use All UNIFACE applications.

Synopsis /bat (*param1 param2 ...*)

Name /c1n
Clean up forms.

Use UNIFACE Six development environment application.

Synopsis /c1n (*form*)

Name /con
Analyze application models.

Use UNIFACE Six development environment application.

Synopsis /con (*application_model*)

Name /cpy
Copy data from one DBMS to another.

Use UNIFACE Six development environment application.

Synopsis /cpy {/apf} {/com=*com*} {/cut=*cut*} {/int=*int*} {/nos} {/whr=*whr*}
source target (mapfile)

Name /cro
Produce cross-reference information when compiling.

Use UNIFACE Six development environment application.

Synopsis /cro(*ss*)

Name /dev
Compile device translation tables.

Use UNIFACE Six development environment application.

Synopsis /dev {/aft=*datetime*} {/bef=*datetime*} (*library*)

Name /dis
Create a set of application distribution files.

Use UNIFACE Six development environment application.

Synopsis /dis *dol* | *numgen*

Name /do1
Create or recreate USYS:uobj.dol.

Use UNIFACE Six development environment application.

Synopsis /do1

Name /exe
Run a form of the UNIFACE Six development environment.

Use UNIFACE Six development environment application.

Synopsis /exe request [parameters]

Name /exp
Export an application.

Use UNIFACE Six development environment application.

Synopsis /exp [/app] [/cut=cut] [/int=int] [/var=library] [/lan=language] application exportfile

Name /frm
Compile forms.

Use UNIFACE Six development environment application.

Synopsis [/frm] [/inf | /war] [/lis] [/aft=datetime] [/bef=datetime] {form}

Note

- /frm is often used in combination with /cro and /zip.

Name /gly
Compile glyphs.

Use UNIFACE Six development environment application.

Synopsis /gly [/aft=datetime] [/bef=datetime] {library}

Name /hlp
Show a summary of command line switches.

Use UNIFACE Six development environment application.

Synopsis /hlp

Name /imp
Import an application export file.

Use UNIFACE Six development environment application.

Synopsis /imp [/com=com] [/int=int] [/nos] exportfile

Name /ini
Specify the GUI resource file used by the application.

Use All UNIFACE applications.

Synopsis /ini=file_name

Name /ins
Install some objects after installation time.

Use UNIFACE Six development environment application.

Synopsis /ins {demo | demodat | dol {object1 object2 ...} | meta}

Name /key
Build keyboard translation tables.

Use UNIFACE Six development environment application.

Synopsis /key *file_name*

Name /lib
Compile global Procs and variables in a library.

Use UNIFACE Six development environment application.

Synopsis /lib {/aft=*datetime*} {/bef=*datetime*} {*library*}

Name /lin
Link an imported application.

Use UNIFACE Six development environment application.

Synopsis /lin *application*

Name /log
Provide logon information to a database or network.

Use All UNIFACE applications.

Synopsis /log=*path* : {*database* | *nodename*} | {*user_id*} | {*password*}

Name /lse
Copy language setups to a file.

Use UNIFACE Six development environment application.

Synopsis /lse *language.lib file*

Name /men
Compile menu bars, menus and menu items.

Use UNIFACE Six development environment application.

Synopsis /men {/aft=*datetime*} {/bef=*datetime*} {*library*}

Name /mes
Compile global messages, help texts and language setups.

Use UNIFACE Six development environment application.

Synopsis /mes {/aft=*datetime*} {/bef=*datetime*} {*library*}

Name /obj
Compile global objects.

Use UNIFACE Six development environment application.

Synopsis /obj {/aft=*datetime*} {/bef=*datetime*} {*library*}

Note

- Using /obj is equivalent to using /tra, /dev, /lib, /mes, /gly, /men and /pan.
-

Name /pan
Compile panels.

Use UNIFACE Six development environment application.

Synopsis /pan {/aft=*datetime*} {/bef=*datetime*} {*library*}

Name /pret
Prepare an application for export.

Use UNIFACE Six development environment application.

Synopsis /pret application

Name /pri
Select the I/O messages sent to the message frame.

Use All UNIFACE applications.

Synopsis /pri=value
For information about determining *value*, see chapter 18 *I/O messages*.

Name /rma
Start the UNIFACE deployment environment.

Use UNIFACE deployment environment.

Synopsis /rma

Name /tfi
'Play back' the TFO (test file output) file.

Use All UNIFACE applications.

Synopsis /tfi={file_name}

Note

- The application must be running in a character mode environment (that is, \$GUI=\$CHR).

Name /tfo
Record keystrokes.

Use All UNIFACE applications.

Synopsis /tfo={file_name}

Note

- The application must be running in a character mode environment (that is, \$GUI=\$CHR).
-

Name /tra
Compile keyboard translation tables.

Use UNIFACE Six development environment application.

Synopsis /tra {/aft=datetime} {/bef=datetime} {/fil file_name} {library}

Name /tst
Run a form in test mode.

Use UNIFACE Six development environment application.

Synopsis /tst form

Name /upd
Perform V5-to-V6 upgrade actions.

Use UNIFACE Six development environment application.

Synopsis /upd action

Perform the V5-to-V6 conversion step requested by one of the actions described in the following table:

Action	Description
v6{a11}	Convert a V5 application dictionary to a V6 Application Objects Repository.
v6menu	Build V6 menu definitions from V5 definitions.
v6lib	Build V6 library definitions.
v6cleanup	Remove V5 menu definitions from a V6 Application Objects Repository.
v6language <i>lib1, lib2, ...</i>	Move language setups from a V5 application dictionary to a V6 Application Objects Repository.

Actions for /upd.

Name	/who Display the UNIFACE installation parameters.
Use	UNIFACE Six development environment application.
Synopsis	/who

Name	/zip Generate compressed forms when compiling.
Use	UNIFACE Six development environment application.
Synopsis	/zip

Notes

- Use **/zip** alone to start the development environment with compressed forms enabled.
- **/zip** is often used in combination with **/a11** and **/frm**.

Name	? Cause the Command Line dialog box to appear.
Use	All UNIFACE applications.
Synopsis	?

Chapter 6 Assignment files

This chapter summarizes the information about assignment files found in the *Assignment files* chapter in the ► *UNIFACE Reference Manual*.

Global and local assignment files for UNIFACE applications

When you start an application, UNIFACE builds an internal assignment file from a global assignment file (one that is available to all UNIFACE applications) and a local assignment file (one that is available to the current application), in that order.

- The global assignment file is `usys.asn` in the installation directory (USYS). If this is not found, there are no global assignments.
- The local assignment file used is the first file found that is:
 - Named with the `/asn` command line switch when the application was started.
 - Specified in the Assignment File entry of the Define Start-up Shell form.
 - Named `shell.asn` in the directory where you started the application.

Sections in an assignment file

An assignment file may be divided into a number of sections, each started by a section header:

- [SETTINGS], for UNIFACE system settings.
- [FILES], for locating non-DBMS files.
- [ENTITIES], for directing model entities onto paths.
- [PATHS], for directing paths to DBMS, network or GUI drivers.
- [WIDGETS], for defining widget properties.

Syntax for settings assignments:

```
$setting { [=] parameters }
```

(These are summarized in chapter 7 *Assignment settings*.)

Syntax for non-DBMS file assignments:

```
logical_filename [=] actual_filename
```

Syntax for path-to-driver assignments:

```
$path [=] driver: {name} | {username} | {password}
```

Syntax for path-to-path assignments:

```
$path1 [=] $path2
```

Syntax for entity assignments:

- Target path leads to the driver for a *record level DBMS*:
`entity.appl_model [=] $path: {directory}table.*`
- Target path leads to the driver for a *field level DBMS*:
`entity.appl_model [=] $path: table.*`
- Target path leads to a *network driver*:
`entity.appl_model [=] $path: entity.*`

Tips for using wildcards

- When matching a logical file name against a wildcard profile, UNIFACE does not distinguish between non-DBMS and DBMS names. Be extremely careful defining your assignments if you choose to use a 'match everything' profile, *.*.
- Remember that the installation directory (USYS) contains important non-DBMS files.

Global and local assignment files for PolyServer

- The global assignment file is named `psys.asn` and is located in the PolyServer installation directory, `PSYS`.
- The local assignment file used is the first file found that is:
 - Defined with the `/asn` command line switch when PolyServer is started. For further information, see the ► *Distributed Computing Guide* module for your platform.
 - Named `psv.asn` in the PolyServer login directory.

Chapter 7 Assignment settings

This chapter summarizes the UNIFACE system settings that you can use in your assignment files to define the environment for your UNIFACE application. For complete information, see the *System setting assignments* chapter in the ► *UNIFACE Reference Manual*.

Name	<code>\$ACTIVE_FIELD</code> Set the video attributes for the active field.
Synopsis	<code>\$ACTIVE_FIELD [=] attribute_1 {, attribute_2, ...}</code>
Notes	<ul style="list-style-type: none"> • The codes for various video attributes are described in chapter 4 <i>Video attributes</i>. • When combining video attributes, you may optionally enclose them in parentheses.
Name	<code>\$CHECK_BOX</code> Set the video attributes for a check box in character mode.
Synopsis	<code>\$CHECK_BOX [=] {on} , {off} , {null}</code>
Notes	<ul style="list-style-type: none"> • The codes for various video attributes that can be used for <i>on</i>, <i>off</i> and <i>null</i> are described in chapter 4 <i>Video attributes</i>. • You can define more than one video attribute for each of the <i>on</i>, <i>off</i> and <i>null</i> parameters. When combining video attributes, separate them with commas and enclose them in parentheses.

Name **\$DBMS_OBJECTS**
Search for global objects in UOBJ table or file before searching DOLs.

Synopsis **\$DBMS_OBJECTS**

Name **\$DEFAULT_TERM**
Set the default keyboard and display table name.

Synopsis **\$DEFAULT_TERM** [=] *device_translation_table*

Name **\$DEF_CHARSET**
Set the default character set.

Synopsis **\$DEF_CHARSET** [=] *code*
The available values for *code* are shown in the following table:

Code	Explanation
DEC	DEC Multinational (ISO Latin-1).
PC	IBM PC code page 437.
IBMRT	IBM RT.
BIG5	Traditional Chinese.
CNS	Traditional Chinese.
ETEN	Traditional Chinese.
IBM5500	Traditional Chinese.
TCA	Traditional Chinese.
GB	Simplified Chinese.
EUC	Japanese.
JIS	Japanese.
SJIS	Japanese.
KSC5601	Korean.

Character sets for data storage supported by UNIFACE.

Name **\$DEF_VIDEO**
Set the default video attributes for fields.

Synopsis **\$DEF_VIDEO** [=] *attribute_1*{, *attribute_2*, ...}

Notes

- The codes for various video attributes are described in chapter 4 *Video attributes*.
- When combining video attributes, you may optionally enclose them in parentheses.
- This assignment also sets the value of the DEF parameter used with the `field_video` Proc instruction.

Name **\$DISPLAY**
Load the named device translation table for display.

Synopsis **\$DISPLAY** [=] *device_translation_table*{-80|-132|:mode}
The available values for *mode* are shown in the following table:

Mode	Explanation
0	Normal screen dimension, 80 columns, 24 lines.
1	Wide screen dimension, 132 columns, 24 lines.
4	Print layout for 80 columns, 66 lines.
5	Print layout for 132 columns, 66 lines.
6	Print layout for 80 columns, 72 lines.
7	Print layout for 80 columns, 72 lines.

Modes for VT device translation tables (except VT340G).

Name **\$DOUBLE_WIDTH**
Set the display to 16-bit double-width characters.

Synopsis **\$DOUBLE_WIDTH** [=] 33

Name **\$ENHANCED_EDIT**
Enable GUI-dependent editing functions.

Synopsis **\$ENHANCED_EDIT** [=option[, option, ...]]

The available options are shown in the following table:

Option	Meaning
CURSOR	Put cursor on the logical, rather than the physical, cursor position in the field. That is: <ul style="list-style-type: none"> • In insert mode, if the mouse is clicked right of the text, put the cursor before the line terminator. • In overstrike mode, if the mouse is clicked right of the text, put the cursor where the click occurs. • In either mode, if the mouse is clicked left of the text, put the cursor on the first character.
HIGHLIGHT	Highlight text based on the logical, rather than the physical, cursor position in the field. That is: <ul style="list-style-type: none"> • In insert mode, if the mouse is clicked right of the text, highlight to the line terminator. • In overstrike mode, if mouse is clicked is right of the text, highlight to the position where the click occurred. • In either mode, if the mouse is clicked left of the text, highlight to the first character.
DELETE	Delete any existing selection before a delete or paste action.
DRAGMOVE	Allow selected text to be dragged to another location in the same field.
ALL	Enable all of these options.

Options available with **SENHANCED_EDIT**.

Notes

- This setting is ignored if the **\$GUI** setting is **\$CHR**.
- If no options are provided, all applicable options are assumed.

Name **\$FILL_DBMS_FIELDS**
Fill DBMS fields like a record level driver does.

Synopsis **\$FILL_DBMS_FIELDS**

Name **\$FORM_TITLE**
Assign form title properties in character mode.

Synopsis **\$FORM_TITLE** [=] option[, video_attributes]

The available values for *option* are shown in the following table:

Option	Description
TRUE	Form titles always appear.
FALSE	Form titles do not appear.
BORDER	Form titles appear only on forms that have a border.

Options for **\$FORM_TITLE**.

Notes

- The codes for various video attributes are described in chapter 4 *Video attributes*. When combining video attributes, separate them with commas and enclose them in parentheses.
- **\$FORM_TITLE** is effective only when the **\$GUI** setting is **\$CHR**.

Name **\$GUI**
Select the GUI driver to use.

Synopsis **\$GUI** [=] *gui_path* | *gui_driver*: {*X_options*}

Name **\$KEYBOARD**
Assign the keyboard translation table to be used.

Synopsis **\$KEYBOARD** [=] *keyboard_translation_table*

Name **\$LANGUAGE**
Assign the language for global objects.

Synopsis **\$LANGUAGE (=) *language***

Name **\$LIST_BOX**
Set the video attributes for a list box in character mode.

Synopsis **\$LIST_BOX (=) {*on*}, {*off*}**

Notes

- The codes for various video attributes that can be used for *on* and *off* are described in chapter 4 *Video attributes*.
- You can define more than one video attribute for each of the *on* and *off* parameters. When combining video attributes, separate them with commas and enclose them in parentheses.

Name **\$MAXFILES**
Assign the maximum number of simultaneously open files.

Synopsis **\$MAXFILES (=) *number***

Name **\$MAX_QUE**
Assign the size of the input queue for asynchronous interrupts.

Synopsis **\$MAX_QUE (=) *number***

Name **\$MENU_BAR**
Assign the position, video attributes and behavior of menu bars.

Synopsis **\$MENU_BAR (=) *location*, {*background*}, {*item*}, {*mnemonic*}, {*TRIG*}**

Notes

- The *location* parameter should be **T** to indicate the top of the screen or **B** for the bottom.
- The codes for various video attributes that can be used for *background*, *item* and *mnemonic* are described in chapter 4 *Video attributes*. You can define more than one video attribute for each of these parameters. When combining video attributes, separate them with commas and enclose them in parentheses.
- The **TRIG** parameter tells UNIFACE to activate the <PULLDOWN> trigger before the menu bar becomes active. Without **TRIG**, any Proc coding in <PULLDOWN> triggers is ignored.
- The default, as used under character mode, is **T**, **, INV, BRI**.
- For most GUIs, only **TRIG** is relevant.

Name **\$MESSAGE_LINE**
Assign the position and video attributes for the message line in character mode.

Synopsis **\$MESSAGE_LINE (=) *location*{, *video_attributes*}**

Notes

- The *location* parameter should be **TOP** to indicate the top of the screen or **BOTTOM** for bottom.
- The codes for various video attributes that can be used for *location* are described in chapter 4 *Video attributes*. When combining video attributes, separate them with commas and enclose them in parentheses.

Name `$NEWLINE`
Specify the character stored for end-of-line.

Synopsis `$NEWLINE [=] option`
The available values of *option* are shown in the following table:

Option	Meaning
CR	Store end-of-line as carriage return (13 ₁₀).
LF	Store end-of-line as line feed (10 ₁₀).
CRLF	Store end-of-line as carriage return, line feed (13 ₁₀ , 10 ₁₀).

End-of-line options available with `$NEWLINE`.

Name `$NO_BUSY`
Disable the **busy** indicator.

Synopsis `$NO_BUSY`

Name `$NO_LMK_NULL`
Handle empty primary key triggers as in Version 5.1.

Synopsis `$NO_LMK_NULL`

Name `$OLDASKMESS`
Handle *askmess* statements as in V5.1.

Synopsis `$OLDASKMESS { [=] TRUE | FALSE }`

Name `$OLDLIBPATH`
Use the pre-V6 search sequence for libraries of global variables.

Synopsis `$OLDLIBPATH`

Notes

- When `$OLDLIBPATH` is present, UNIFACE searches for global variables in these libraries:
 1. The library specified on the form.
 2. The library specified on the start-up shell.
 3. The library `SYSTEM_LIBRARY`.
- The V6 search paths are described in chapter 21 *Search order for global objects*.

Name `$PROFILE_VIDEO`
Set the video attributes for profile characters in unifields.

Synopsis `$PROFILE_VIDEO [=] attribute_1 {, attribute_2, ...}`

Notes

- The codes for various video attributes are described in chapter 4 *Video attributes*.
- When combining video attributes, you may optionally enclose them in parentheses.

Name `$PUTMESS_LOGFILE`
Copy the message frame contents to a file.

Synopsis `$PUTMESS_LOGFILE [=] filename`

Name `$RADIO_BUTTON`
Set the video attributes for radio group buttons in character mode.

Synopsis `$RADIO_BUTTON [=] {on}, {off}`

Notes

- The codes for various video attributes that can be used for *on* and *off* are described in chapter 4 *Video attributes*.
- You can define more than one video attribute for each of the *on* and *off* parameters. When combining video attributes, separate them with commas and enclose them in parentheses.

Name `$REMOTE_path`
Specify login information for a DBMS or network login on a remote machine accessed via PolyServer.

Synopsis `$REMOTE_path {=} driver:database |user |password`

Name `$SWAP_COLORS`
Invert the color matrix.

Synopsis `$SWAP_COLORS {=} ON`

Name `$TIMEOUT`
Generate an asynchronous interrupt after an inactive period.

Synopsis `$TIMEOUT minutes`

Name `$TWO_PHASE_COMMIT`
Enable two-phase commit functionality.

Synopsis `$TWO_PHASE_COMMIT`

Name `$VARIATION`
Assign the library to use when the application is started.

Synopsis `$VARIATION {=} library`

Chapter 8 Models of the Repository

For more detailed information, see the *Application Objects Repository* chapter in the ► *UNIFACE Reference Manual*.

Model	Path	Contents
DICT	\$IDF	Entities needed by the UNIFACE X4 development environment.
PRINTER	\$SYS	Entity for print job models.
SYSENV	\$SYS	Entities needed to define permissions and preferences.
TEXT	\$UUU	Entities needed by both the development environment and the UNIFACE run-time system.
UVCS	\$IDF	Entities needed for version control.

Application models of the Application Objects Repository.

Application model **DICT**

Entity	Can create overflow table or file?	Description
<i>Application model definitions:</i>		
UCFIELD	Y	Definitions for fields.
UCGROUP	Y	Definitions for entity subtypes.
UCKEY	Y	Key definitions for entities.
UCRELSH	Y	Relationships between entity subtypes.
UCSCH	Y	Definitions for application models.

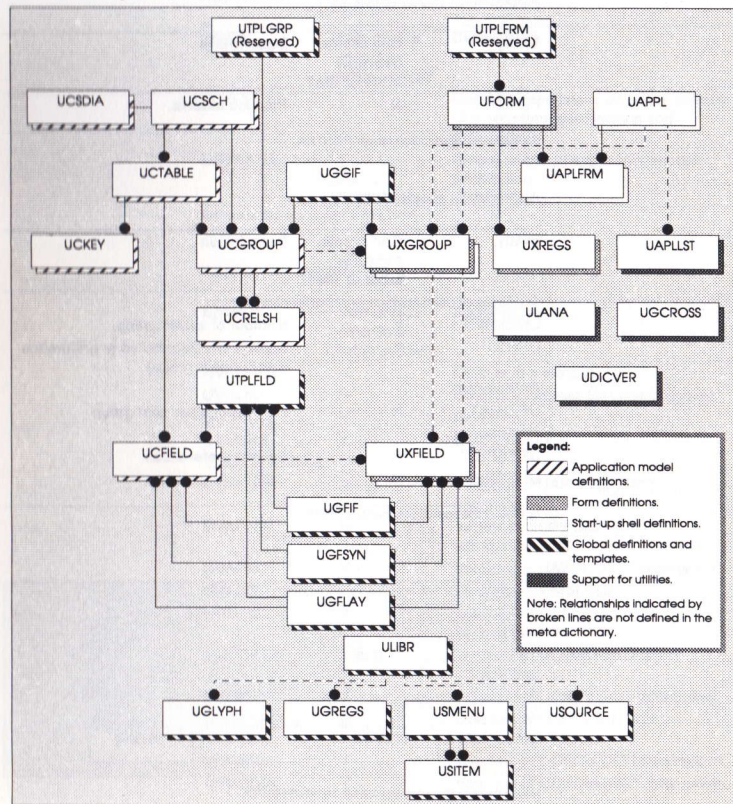
Entities of the application model DICT.

part 1 of 2

Entity	Can create overflow table or file?	Description
UCSDIA	Y	Application model diagrams.
UCTABLE	Y	Definitions for entities (supertypes).
<i>Form definitions:</i>		
UFORM	Y	Definitions for forms.
UXFIELD	Y	Definitions for form variations of fields.
UXGROUP	Y	Definitions for form variations of entities.
UXREGS	Y	Definitions for local variables.
<i>Start-up shell definitions:</i>		
UAPLFRM	N	'Scan list' of forms per start-up shell.
UAPPL	Y	Definitions for start-up shells.
<i>Global definitions and templates:</i>		
UGFIF	Y	Interface templates for fields.
UGFLAY	Y	Layout templates for fields.
UGFSYN	Y	Syntax templates for fields.
UGGIF	Y	Interface templates for entities.
UGLYPH	Y	Glyph definitions.
UGREGS	Y	Definitions for global variables.
ULIBR	N	Definitions for libraries.
USITEM	Y	Definitions for menu items.
USMENU	Y	Definitions for menus.
USOURCE	Y	Source for global Procs, messages and so on.
UTPLFLD	Y	Templates for fields.
UTPLFRM	Y	Templates for forms. (Reserved.)
UTPLGRP	Y	Templates for entities. (Reserved.)
<i>Support for utilities:</i>		
UAPLLST	N	Application distribution list.
UDICVER	N	Record of current Repository version.
UGCROSS	N	Cross-reference information.
ULANA	Y	Local analyzed application models information.

Entities of the application model DICT.

part 2 of 2



Relationships within the application model DICT.

Application model **PRINTER**

Entity	Can create overflow table or file?	Description
PRATT	N	Print job models.

Entities of the application model PRINTER.

Application model **SYSENV**

Entity	Can create overflow table or file?	Description
--------	------------------------------------	-------------

Subjects of permissions and preferences:

UMEMBER	N	Member of a user group.
USUBJ	Y	Subject of a permission or preference (user or user group).

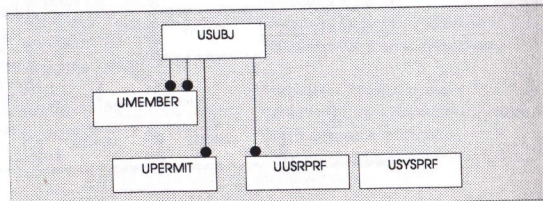
Permissions:

UPERMIT	N	Permissions per user group.
---------	---	-----------------------------

Preferences:

USYSPRF	N	System preferences.
UUSRPRF	N	User preferences.

Entities of the application model SYSENV.



Relationships within the application model SYSENV.

Application model **TEXT**

Entity	Can create overflow table or file?	Description
UOBJ	Y	Compiled versions of all global objects, plus counters, permissions and preferences.
USYSANA	Y	Global analyzed application models information.

Entities of the application model TEXT.

Application model **UVCS**

Entity	Can create overflow table or file?	Description
UVELMLS	N	Objects in a release.
UVELMT	N	Objects submitted for version control.
UVSYSLS	Y	Releases.
UVVERS	Y	Object versions.
<i>'Shadow' entities:</i>		
UVAPLFR	N	Versions of UAPLFRM.DICT information.
UVAPLLS	N	Versions of UAPLLST.DICT information.
UVAPPL	Y	Versions of UAPPL.DICT information.
UVASCI	Y	Versions of ASCII files.
UVCFIEL	Y	Versions of UCFIELD.DICT information.
UVCGROU	Y	Versions of UCGROUP.DICT information.
UVCKEY	Y	Versions of UCKEY.DICT information.
UVCRELS	Y	Versions of UCRELSH.DICT information.
UVCSCH	Y	Versions of UCSCH.DICT information.
UVCSDIA	Y	Versions of UCSDIA.DICT information.

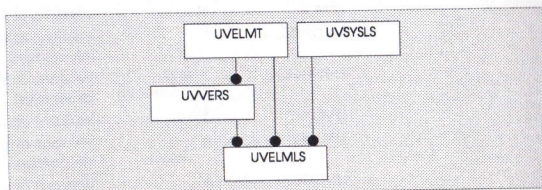
Entities of the application model UVCS.

part 1 of 2

Entity	Can create overflow table or file?	Description
UVCTABL	Y	Versions of UCTABLE.DICT information.
UVFORM	Y	Versions of UFORM.DICT information.
UVGFIF	Y	Versions of UGFIF.DICT information.
UVGFLAY	Y	Versions of UGFLAY.DICT information.
UVGFSYN	Y	Versions of UGFSYN.DICT information.
UVGGIF	Y	Versions of UGGIF.DICT information.
UVGLYPH	Y	Versions of UGLYPH.DICT information.
UVGREGS	Y	Versions of UGREGS.DICT information.
UVSITEM	Y	Versions of USITEM.DICT information.
UVSMENU	Y	Versions of USMENU.DICT information.
UVSOURC	Y	Versions of USOURCE.DICT information.
UVTPLFL	Y	Versions of UTPLFLD.DICT information.
UVXFIEL	Y	Versions of UXFIELD.DICT information.
UVXGROU	Y	Versions of UXGROUP.DICT information.
UVXREGS	Y	Versions of UXREGS.DICT information.

Entities of the application model UVCS.

part 2 of 2



Relationships within the application model UVCS.

Chapter 9 Structure editor functions

This information can also be found in the *Structure editor functions* appendix in the ► *UNIFACE Reference Manual*.

The list of structure editor functions in the table that follows is sorted alphabetically by the name of the function.

- The 'Numeric value' column shows the internal decimal value of the function. This value can be used in defining keyboard translation tables and in **macro** statements.
- The 'Level' column shows the level at which the function is valid; a function is valid for the level specified, plus all lower levels. The levels, in order of priority, are application, form, entity and field.
- The 'Associated trigger' column shows the debugger name of the trigger directly activated by the function. (Chapter 17 *Trigger mnemonics* lists the debugger names for the triggers.)
- The 'Classification' column provides a general classification for the function, supplemental to the information provided in the 'Purpose' column.
- The 'Purpose' column explains what each function does when it is encountered in a valid context.



Note: Most navigation functions have meaning only for unfield widgets. All functions, except for those designated as enhanced editor functions, are valid in character mode, if the context is correct. All functions starting with CURSOR are also valid at form level in character mode.

Function	Numeric value	Level	Associated trigger	Classification	Purpose
ACCEPT	^127^009	Form	ACPT	Session control	Activate <ACCEPT> trigger; intended to end edit session for current form on a positive note. (See also QUIT.)
ADD_OCC	^127^044	Entity	AIO	Data entry	Append a new occurrence after the current one, that is, at position \$cazccc+1.
ATTRIBUTE	^127^078	Field	none	Text editing	Define character attributes (bold, italic and so on).
BEGIN_LINE	^127^188	Field	none	Navigation	Move the cursor to the beginning of the line.
BOLD	^127^147	Field	none	Text editing	Toggle bold character attribute.
BOTTOM	^127^023	Form	none	Navigation	Move the cursor to the window bottom.
BOT_OF_FORM	^127^021	Form	none	Navigation	Move the cursor to the form bottom.
CHAR	^255^001	Field	none	Navigation or text editing	Move cursor as NEXT_CHAR or PREV_CHAR (depending on direction mode), or apply a composite function like REMOVE to a character.
CLEAR	^127^012	Form	CLR	Data entry	Activate <CLEAR> trigger (intended to clear all data from the form and hitlist without saving, or release primary key controls).
COMPOSE	^127^088	Field	none	Text editing	Compose character.
CURSOR_DOWN	^127^017	Field	none	Navigation	Move the cursor down one line.
CURSOR_FAST_DOWN	^127^026	Field	none	Navigation	Move the cursor eight lines down.
CURSOR_FAST_LEFT	^127^027	Field	none	Navigation	Move the cursor eight spaces left.
CURSOR_FAST_RIGHT	^127^028	Field	none	Navigation	Move the cursor eight spaces right.
CURSOR_FAST_UP	^127^025	Field	none	Navigation	Move the cursor eight lines up.
CURSOR_LEFT	^127^018	Field	none	Navigation	Move the cursor left one position.
CURSOR_RIGHT	^127^019	Field	none	Navigation	Move the cursor right one position.
CURSOR_UP	^127^016	Field	none	Navigation	Move the cursor up one line.
DETAIL	^127^094	Field, entity	DTLF, DTLE	Services	Activate the <DETAIL> trigger for the current field or, if the trigger is empty, the entity.
END_LINE	^127^189	Field	none	Navigation	Move the cursor to the end of the current line.
ERASE	^127^008	Form	ERAS	Database I/O	Delete all data in the form, both in the form and in the database.
FIELD	^255^010	Field	NFLD or PFLD (navigation only)	Navigation or text editing	More cursor as NEXT_FIELD or PREV_FIELD (depending on direction mode), or apply a composite function like REMOVE to a field.

Structure editor functions.

part 1 of 5

Function	Numeric value	Level	Associated trigger	Classification	Purpose
FIND_TEXT	^127^150	Field	none	Text editing	Search for previously specified string or profile. (See also PROFILE.)
FIRST	^255^067	Field, entity	none	Navigation	Composite function, for use with objects like WORD and OCCURRENCE (but not with FIELD).
FIRST_OCC	^127^037	Entity	none, but will activate OGF for first occurrence	Navigation	Move cursor to first promptable field of first occurrence in current entity; activate OGF for that occurrence.
FIRST_TEXT	^127^129	Field	none	Navigation	Move cursor to beginning of text.
FONT	^127^151	Field	none	Text editing	Choose UNIFACE character set.
FRAME	^127^089	Field	none	Text editing	Define a frame (run Define Frame form).
HELP	^127^092	Field, entity	HLPF, HLPE	Services	Activate the <HELP> trigger for the current field or, if the trigger is empty, the entity.
HOME	^127^022	Form	none	Navigation	Move cursor to top of form window.
INSERT	^255^071	Field, entity	AIO, if the object is an occurrence	Text editing, data entry	Insert object specified (like WORD) in a composite function. When the object is OCCURRENCE, same as INS_OCC; otherwise, insert contents of buffer for the object.
INS_CHAR	^127^184	Field	none	Text editing	Insert character in the INS_CHAR buffer.
INS_FIELD	^127^181	Field	none	Text editing	Insert contents of the INS_FIELD buffer.
INS_FILE	^127^180	Field	none	Data entry	Insert file into current field (this is the same as the #11e10aa statement).
INS_LINE	^127^182	Field	none	Text editing	Insert the contents of the INS_LINE buffer.
INS_OCC	^127^043	Entity	AIO	Data entry	Insert a new occurrence before the current occurrence, that is, at position \$cazccc.
INS_OVER	^127^146	Application	none	Mode toggle	Toggle Insert/Overstrike mode.
INS_SELECT	^127^195	Field	none	Text editing	Insert contents of the INS_SELECT buffer, or, if this is empty, the INS_FIELD buffer.
INS_TEXT	^127^177	Field	none	Text editing	Insert contents of the INS_TEXT buffer.
INS_WORD	^127^183	Field	none	Text editing	Insert contents of the INS_WORD buffer.
ITALIC	^127^148	Field	none	Text editing	Toggle italic character attribute.
KEY_HELP	^127^072	Application	none	Services	Keyboard layout help.
LAST	^255^068	Field, entity	none	Navigation	Composite function, used with objects like WORD and OCCURRENCE (but not with FIELD).
LAST_OCC	^127^038	Entity	none, but will activate OGF for last occurrence	Navigation	Move cursor to first promptable field of last occurrence in current entity; activate OGF for that occurrence.
LAST_TEXT	^127^128	Field	none	Navigation	Move cursor to end of text.

Structure editor functions.

part 2 of 5

Function	Numeric value	Level	Associated trigger	Classification	Purpose
LINE	^255^004	Field	none	Navigation or text editing	Move cursor as NEXT_LINE or PREV_LINE (depending on direction mode), or apply a composite function like REMOVE to a line.
MENU	^127^101	Field, entity	MNUF, MNUE	Services	Activate the <MENU> trigger for the current field or, if the trigger is empty, the entity.
MESSAGE	^127^093	Application	none	Services	Display the message frame.
NEXT	^255^065	Field, entity	none	Mode toggle or navigation	Set the direction mode to Next, or, if specified with an object, a composite navigation function.
NEXT_CHAR	^127^142	Field	none	Navigation	Move cursor to next character.
NEXT_FIELD	^127^046	Field	NFLD	Navigation	Activate the <NEXT FIELD> trigger for the current field, or, if the trigger is empty, move cursor to next promptable field.
NEXT_LINE	^127^136	Field	none	Navigation	Move cursor to beginning of next line.
NEXT_OCC	^127^039	Entity	none, but will activate OGF for occurrence	Navigation	Move cursor to first promptable field of next occurrence in current entity, activate OGF for that occurrence.
NEXT_TEXT	^127^163	Field	none	Navigation	Move cursor to beginning of the next text section that is not currently visible, below.
NEXT_WORD	^127^140	Field	none	Navigation	Move cursor to beginning of next word.
OCCURRENCE	^255^011	Entity	none for navigation, AIO for add or insert, RMO for remove	Navigation or data entry	Give focus to next or previous occurrence (depending on direction mode), or apply a composite function like REMOVE to an object.
OCC_WINDOW	^255^015	Entity	none	Navigation	Scroll the displayed occurrences up (in Next mode) or down (in Previous mode) by as many occurrences as are painted on the form for that entity.
PAGE_DOWN	^127^198	Field	none	Navigation	Enhanced editor: scroll the text in the field down one page, leave cursor in same relative place.
PAGE_UP	^127^197	Field	none	Navigation	Enhanced editor: scroll the text in the field up one page, leave cursor in same relative place.
PREV	^255^066	Field, entity	none	Mode toggle or navigation	Set the direction mode to Previous, or, if specified with an object, a composite navigation function.
PREV_CHAR	^127^143	Field	none	Navigation	Move cursor to previous character.

Structure editor functions.

part 3 of 5

Function	Numeric value	Level	Associated trigger	Classification	Purpose
PREV_FIELD	^127^047	Field	PFLD	Navigation	Activate the <PREVIOUS FIELD> trigger for the current field, or, if the trigger is empty, move cursor to previous promptable field.
PREV_LINE	^127^137	Field	none	Navigation	Move cursor to beginning of previous line.
PREV_OCC	^127^040	Entity	none, but will activate OGF for occurrence	Navigation	Move cursor to first promptable field of previous occurrence in current entity, activate OGF for that occurrence.
PREV_TEXT	^127^162	Field	none	Navigation	Move cursor to beginning of the next text section that is not currently visible, above.
PREV_WORD	^127^141	Field	none	Navigation	Move cursor to beginning of next word.
PRINT	^127^098	Form	PRNT	Services	Activate the <PRINT> trigger, or, if trigger is empty, run the Print form.
PRINT_ATTRIBUTES	^127^099	Form	none	Services	Run the Print Job Model form.
PROFILE	^127^087	Field	none	Text editing	Define string or profile to search for with FIND_TEXT function.
PULLDOWN	^127^086	Form, application	PULS, PULA	Services	Activate the <PULLDOWN> trigger for the current form, or, if the trigger is empty, the application.
QSELECT	^127^190	Field	none	Text editing	Enhanced editor: as SELECT but without information message.
QRESET_SELECT	^127^191	Field	none	Text editing	Enhanced editor: as RESET_SELECT but without information message.
QUICK_ZOOM	^127^096	Field	none	Services	Zoom current field to maximum zoom size in one step.
QUIT	^127^010	Form	QUIT	Session control	Activate <QUIT> trigger (intended to end edit session for current form on a negative note, ignore modifications).
REFRESH	^127^067	Application	none	Services	Refresh the screen.
REMOVE	^255^073	Field, entity	RMO, if the object is an occurrence	Text editing, data entry	Remove the object specified (like WORD) in a composite function; all removed objects except for occurrences are written to INSERT buffers.
REM_CHAR	^127^172	Field	none	Text editing	Delete character to right of cursor and write to INS_CHAR buffer.
REM_FIELD	^127^166	Field	none	Text editing	Remove the current selection (or the entire field if nothing is selected) to INS_FIELD and INS_SELECT buffers.
REM_FILE	^127^192	Field	none	Text editing	Write contents of field to a file (this is the same as the \$11.edump statement).
REM_LINE	^127^167	Field	none	Text editing	Remove the current line (from right of cursor) to INS_LINE buffer.

Structure editor functions.

part 4 of 5

Function	Numeric value	Level	Associated trigger	Classification	Purpose
REM_OCC	^127^045	Entity	RMO	Text editing	Remove the current occurrence.
REM_SEL_CHAR	^127^174	Field	none	Text editing	Enhanced editor: Remove selected characters (using Delete).
REM_SELECT	^127^194	Field	none	Text editing	Remove selected text to INS_SELECT and INS_FIELD buffers.
REM_WORD	^127^169	Field	none	Text editing	Remove current word (from right of cursor) to INS_WORD buffer.
RESET_SELECT	^127^196	Field	none	Text editing	Turn off Select mode.
RETRIEVE	^127^005	Form	RETR	Database I/O	Activate <RETRIEVE> trigger.
RETRIEVE_SEQ	^127^003	Form	RETS	Database I/O	Activate <RETRIEVE SEQUENTIAL> trigger.
RUB_CHAR	^127^173	Field	none	Text editing	Backspace (delete character to left of cursor).
RUB_SEL_CHAR	^127^175	Field	none	Text editing	Remove selected characters (using Backspace).
RULER	^127^081	Field	none	Text editing	Ruler definition (run Ruler form).
SAVE	^127^179	Field	none	Text editing	Write selected text to INS_SELECT buffer.
SELECT	^127^193	Field	none	Text editing	Turn on Select mode.
SQL	^127^097	Application	none	Services	Run the SQL form.
STORE	^127^011	Form	STOR	Database I/O	Activate the <STORE> trigger.
SWITCH_KEY	^127^100	Application	SWIT	Services	Activate the <SWITCH KEYBOARD> trigger.
TEXT	^255^009	Field	none	Navigation or text editing	Move cursor as NEXT_TEXT or PREV_TEXT (depending on direction mode), or apply a composite function like REMOVE to text.
TEXT_WINDOW	^255^014	Field	none	Navigation	Scroll text up or down, depending on direction mode.
TOP_OF_FORM	^127^020	Form	none	Navigation	Move cursor to the top of the form.
UNDERLINE	^127^149	Field	none	Text editing	Toggle underline character attribute.
USER_KEY	^127^091	Form, application	UKYS, UKYA	Services	Activate the <USER KEY> trigger for the current form, or, if the trigger is empty, the application.
VIEW	^127^073	Application	none	Text editing	Toggle view mode on or off.
WORD	^255^003	Field	none	Navigation or text editing	Move cursor as NEXT_WORD or PREV_WORD (depending on direction mode), or apply a composite function like REMOVE a word.
ZOOM	^127^095	Field	none	Services	Zoom the current field.

Structure editor functions.

part 5 of 5

Chapter 10 Interface definitions

This chapter summarizes the data types, packing codes, variable length techniques and shorthand codes you can use in field interface definitions. See the *Templates* chapter in the ► *Designers' Guide* for more details.

Data types, default packing codes and layouts

The following table shows the default packing codes and layouts for each UNIFACE data type:

UNIFACE data type	Explanation	Default packing code	Default layout
S*	String	C40	
SS**	Special String	C40	
R	Raw data	R	
I	Image (images and glyphs)	R	
N	Numeric	F	
F	Floating decimal point	F	
D	Date	D	DDMMYYYY
T	Time	T	HH:NN:SS
E	Date and Time (combined)	E	DDMMYYYY HHNNSS
LD	Linear Date	D	DDMMYYYY
LT	Linear Time	T	HH:NN:SS
LE	Linear Date and Time (combined)	E	DDMMYYYY HHNNSS
B	Boolean	B	T/F, t/f

Data types, default packing codes and layouts.

Table notes:

* Strings allow UNIFACE Fonts 0 and 1.

** Special strings allow all UNIFACE Fonts, plus 0 and 1.

Packing codes

The following table shows valid packing codes for UNIFACE:

Packing code	Explanation
C1-C*	Character (number if type 'N', only C1-C32 are allowed; numbers are stored as sign left, right-aligned, decimal point included).
VC1-VC*	Variable length character string.
SC1-SC*	Segmented character string.
U1-U*	TRX character.
VU1-VU*	TRX length variable character string.
SU1-SU*	TRX segmented character string.
R1-R*	Binary (raw).
SR1-SR*	Segmented binary (raw).
VR1-VR*	Variable length binary (raw).
I1	One-byte integer.
I2	Two-byte integer.
I3	Three-byte integer.
I4	Four-byte integer.
I8	Eight-byte integer.
J1-J32	ASCII Number string.
M1	Money: eight-byte integer, scaling 4.
M2	Money: double precision D-float.
M4	SYBASE Money format, scaling 4.
M6	SYBASE Small Money format.
N1-N32	Number, stored without decimal point.
O1-O32	Zoned numeric encoding of DEC OpenVMS VAX trailing numeric.
P1-P8	Packed decimal, +/- at beginning of field.
Q1-Q8	Packed decimal, +/- at end of field.
F	Optimum DBMS Floating Point default.
F4	Single precision F-float.
F8	Double precision D-float.
D	Optimum DBMS Date default.
D1	ASCII Date DD-MMM-YYYY.

Valid packing codes for UNIFACE.

part 1 of 3

Packing code	Explanation
D2	ASCII Date YYYYMMDD.
D3	ASCII Date DDDMMYYYY.
D4	ASCII Date YYYYMMDD.
D5	ASCII Date DDDMMYY.
D6	Binary Date YYMD.
D7	Binary Date DMY.
D8	Binary Date YMD.
D9	Binary Date DM.
D10	Binary Date YYYYMMDD.
D11	Binary Date DDDMMYY.
D12	ASCII Date MM/DD/YYYY.
D13	ASCII Date YYYY-MM-DD.
E	Optimum DBMS combined Datetime default.
E1	SYBASE linear four-byte Date and four-byte Time.
E2	RMS linear Datetime.
E3	ASCII Date DDDMMYYYY Time HH:NN:SS.
E4	ASCII Date DDDMMYYYY Time HHNNSS.
E5	Ingres Date DD-MMM-YYYY Time HH:NN:SS.
E6	ORACLE internal Datetime format.
E7	SYBASE ASCII Date MM/DD/YYYY HH:NN:SS.TT.
E8	ASCII Datetime YYYYMMDDHHMMSS (like D2+T2).
E9	SYBASE binary Small Datetime.
E10	ASCII Datetime YYYY-MM-DD HH:MM:SS.
E11	Rdb date YYYY-MM-DD Time HH:MM:SS.
E12	Rdb date YYYY-MM-DD Time HH:MM:SS.S.
E13	Rdb date YYYY-MM-DD Time HH:MM:SS.SS.
T	Optimum DBMS Time default.
T1	ASCII Time HH:NN:SS.
T2	ASCII Time HHHMMSS.
T3	ASCII Date DDDMMYYYY Time HHHMMSS.
T4	Time HH:MM:SS.
T5	Time HH:MM:SS.S.
T6	Time HH:MM:SS.SS.

Valid packing codes for UNIFACE.

part 2 of 3

Packing code	Explanation
B	Optimum DBMS Boolean default.
B1	ASCII Boolean 0/1.
B2	ASCII Boolean F/T.
B3	ASCII Boolean N/Y.
B4	Binary Boolean 0/1.
B5	Binary Boolean 0/-1.

Valid packing codes for UNIFACE.

part 3 of 3

Valid combinations of data type and packing code

The following table shows valid combinations of data types and packing codes. Each valid combination is indicated by a bullet (*). Each empty cell indicates that the combination is not valid:

UNIFACE packing code	UNIFACE data type									
	SS	R	N	F	LD	LE	LT	B	I	
C1-C*	*	*	*		*	*	*	*	*	
VC1-VC*	*									
SC1-SC*	*									
U1-U*	*	*								
VU1-VU*	*	*								
SU1-SU*	*	*								
R1-R*		*							*	
VR1-VR*		*								
SR1-SR*		*								
I1-I4			*							
N1-N32			*							
M1-M4			*							
O1-O32			*							
P1-P8			*							

Data type and packing code combinations.

part 1 of 2

UNIFACE packing code	UNIFACE data type									
	SS	R	N	F	LD	LE	LT	B	I	
Q1-Q8			*							
F			*	*						
F4			*	*						
F8			*	*						
D					*					
D1-D13					*					
E					*	*	*			
E1-E13					*	*	*			
T							*			
T1-T6							*			
B1-B4								*		

Data type and packing code combinations.

part 2 of 2

Variable length techniques

Use a variable length definition to define more than one variable length field in an entity. To do this use one or both of the following techniques:

- String identification.
- Length identification.

String identification

String identification uses ASCII strings to mark the field, the first subfield and subsequent subfield occurrences, if defined.

To specify string identification:

- Choose String Identifier from the Identification Type drop-down list.
- Enter the actual string or, if non-printing ASCII, the decimal value of the ASCII string to be used as a string identifier in the Field Identifier field (for example, *28).

Length identification

Length identification inserts between one- and four-bytes into the data string in the field, where the length of the field is maintained in binary. UNIFACE keeps the length identification up-to-date.

To specify length identification choose a length identifier from the Identification Type drop-down list. The available options in this list are shown in the following table (where *x* is a value from 1 through 4):

Option	Meaning
x-byte binary length	Use a binary length identifier which is <i>x</i> bytes long.
String Id. and x-byte binary length	Use a string identifier (see above) followed by a binary length identifier which is <i>x</i> bytes long.
x-byte binary length and String Id.	Use a binary length identifier which is <i>x</i> bytes long followed by a string identifier.

Options available from the Identification Type drop-down list.

Shorthand codes for interface definitions

- Enter shorthand codes in the Interface field on the Define Field form.
- Click on the More button to define a shorthand definition.

The syntax for shorthand codes is as follows:

PLs{n}{(Lt)}\vTy\FI\SI\SS

Where *v* is a constant which indicates where the variable part of the definition begins. This syntax is explained in the following table:

Code	Meaning
P	Packing code type. For example, C, VC.
Ls	Length of each subfield in the field. If no subfields are defined Ls is the total field length (no subfields = 1).
.n	Decimal point and scaling, optional
Lt	Total length of field if subfields are defined, optional
Ty	Variable length identifier type.
FI	Field identifier.
1SI	First subfield identifier.
SS	Subfield separator.

Syntax for interface definitions.

Chapter 11 Syntax definitions

This chapter summarizes the formats, allowed characters and shorthand codes you can use in syntax definitions. See the *Templates* chapter in the ► *Designers' Guide* for further information.

Entry format

Enter your syntax string in the Format field of the Define Field Syntax Template form. The following table shows the syntax codes you can use in syntax strings:

Syntax codes	Explanation
#	One digit (0-9).
#*	0- <i>n</i> digits.
&	One letter (A-Z, a-z).
&*	0- <i>n</i> letters.
@	One letter, digit or underscore (_).
@*	0- <i>n</i> letters, digits or underscores.
?	One ASCII character.
?*	0- <i>n</i> ASCII characters.
*	0- <i>n</i> ASCII characters (same as ?*).
A-Z	That uppercase letter (A, B, C and so on).
a-z	That letter in either case (A, a, B, b, and so on).
x	Any ASCII character except the syntax code characters (#, *, &, @, ?, (,), % and ^).
%x	Any ASCII character, with no special meaning.
(any)	The syntax string <i>any</i> is optional. A syntax check is only done if data is present.

Syntax string codes.

Examples of entry formats

The following table gives examples of entry formats:

Entry format	Allowed	Not allowed
#*	1000 [nothing] 12 12314567	1,000 0000.0 34 456 10A
?*%*	Any text with an asterisk (*) as the last character.	Text without an asterisk as the last character.
#*.##	1000.00 0.50 .50 10.53	1,000.00 0.5 0.510 1000,50
Mr. Smith	MR. SMITH Mr. Smith MR. Smith Mr. SMITH	mr. smith Mr.Smith MR. Smith Mr. Jones
(###) ###-####	404 396-3040 396-3040	(404) 396-3040 41 396-3040 396 3040
% (###%) ###-####	(404) 396-3040	404 396-3040 396-3040 (404)396-3040
@*	Smith1 Smith Jones 12345X2	Smith? Mr. Smith 12345*2
# ##-&	1 23-a	123-a 1 23-a 1 23 - a
#*?##	123,45 0.30 a30 121/33	12345 .3 ,300 123.300
%#	#	Anything else!
(J) (N) (.)	J N .	Anything else!

Examples of entry formats.

Characters Allowed field

The following table shows the possible options you maybe be available in the Characters Allowed field:

Option	Allowed characters
Digits only	0-9
Numbers only	0-9, . + -
ASCII only	UNIFACE Font 0
ISO Latin-1	UNIFACE Fonts 0 and 1
Full character set	UNIFACE Fonts 0 through 7

Options for Characters Allowed field.

The data type for a field affects what options are available in the Characters Allowed field. The following table shows the options available for each data type. Each valid option is indicated by a bullet (*). Each default option is indicated by a double bullet (**).

Characters allowed	UNIFACE data type											
	S	SS	R	N	F	D	LD	T	LT	E	LE	B
Digits only	*	*	*	*	*	*	*	*	*	*	*	*
Numbers only	*	*	*	**	**							*
ASCII only	*	*	*			**	*	**	*	*	*	*
ISO Latin-1	**	*	*				**		**	**	**	**
Full character set	**	**										

Characters allowed per data type.

Data clean-up

The following table shows the data clean-up options for a syntax definition. (To see these options click the More button on the Define Field Syntax Template form):

Field	Action
Leading Spaces	Delete spaces before the first non-space character.
Leading Zeros	Delete zeros before the first non-zero character.
Leading Control	Delete non-printing control characters before the first non-control character in the data, except for carriage returns, form feeds, line feeds, tabs and vertical tabs.
Control	Delete carriage returns, form feeds, line feeds, tabs and vertical tabs, wherever they are in the data.
Text Control	Delete non-printing control characters except for carriage returns, form feeds, line feeds, tabs and vertical tabs, wherever they are in the data.
Trailing Control	Delete non-printing control characters after the last non-control character in the data, except for carriage returns, form feeds, line feeds, tabs and vertical tabs.

Data clean-up options.

Shorthand codes

The following rules apply when entering shorthand codes for syntax definitions:

- The maximum number of characters allowed = 31.
- Separate codes with a comma (,).

The following table shows the valid shorthand codes:

Code	Description
ASC	UNIFACE font 0 only.
BRM	Check that brackets match.
DIG	Digits only.

Shorthand codes for field syntax definitions.

part 1 of 3

Code	Description
DIM	Dim field. No edit and no prompt.
DLC	Delete leading control characters.
DLS	Delete leading spaces.
DLZ	Delete leading zeros.
DTC	Delete trailing control characters.
ENT(<i>syntax</i>)	Entry format.
FUL	Full character set allowed.
HID	Hide field. No display, no edit and no prompt.
JMP	Auto jump.
LEN(<i>n-m</i>)	Length of field or subfield: <i>n</i> = minimum, <i>m</i> = maximum.
LOW	All lowercase.
MAN	Mandatory field (minimum length of one).
MOD(<i>n</i>)	Use checkdigit modulo number <i>n</i> .
MUL	UNIFACE fonts 0 and 1 only.
NBLD	Bold not allowed.
NCR	Carriage returns not allowed. Only one-line field allowed.
NDCC	Do not delete any control characters.
NDCX	Do not delete text control characters.
NDI	Do not display the contents of this field.
NED	No edit allowed in this field.
NITA	Italics not allowed.
NPR	Do not prompt this field.
NUM	Numbers only.
NUND	Underlining not allowed.
OVS	Overstrike.
PRO(<i>characters</i>)	Profile allowed.
RCS	Replace contiguous spaces with one space.
REP(<i>n-m</i>)	Repetition of subfield: <i>n</i> = minimum, <i>m</i> = maximum.
UPC	All uppercase.
YBLD	Bold allowed.
YDCC	Delete all control characters.
YDCX	Delete all text control characters.

Shorthand codes for field syntax definitions.

part 2 of 3

Code	Description
YITA	Italics allowed.
YUND	Underlining allowed.

Shorthand codes for field syntax definitions.

part 3 of 3

Chapter 12 Layout definitions

This chapter summarizes the display formats you can use in field layout definitions. See the *Templates* chapter in the ► *Designers' Guide* for further information.

Shorthand codes

Separate shorthand codes with a comma (.). The following table shows a list of the shorthand codes you can use when defining field layout definitions:

Code	Description
BLI	Blink.
BOR	Borderline.
BRI	Bright.
CTR	Center alignment.
DEC	Decimal alignment.
DIS(<i>format</i>)	Display format.
INV	Inverse video.
LFT	Left alignment.
NAV	No active field video.
NBR	Not bright.
NBL	No blink.
NIN	Not inverse video.
NUN	No underline.
RGT	Right alignment.
SEP(<i>c</i>)	Use subfield separator <i>c</i> .
UND	Underline.
WID(<i>n</i>)	Line width of <i>n</i> characters.

Shorthand codes for field layout definitions.

String

For string fields with data type String or Special String the valid display format codes are as follows:

Display format	What is displayed
?	Character from data element.
%?	One question mark.
%%	One percent symbol.
Any ASCII character	That ASCII character as a constant.

String display format codes.

Examples for string fields

The following table shows examples of display format codes for string fields:

Format	Input	Displayed
Mr. ??????	Plato Stephenson	Mr. Plato Mr. Steph
Mr. ???????	Plato Stephenson	Mr. Plato? Mr. Steph?
Mr. ??????%	Plato Stephenson	Mr. Plato% Mr. Steph%

Examples of format codes for string fields.

Numeric and float

For numeric fields with the data type Numeric or Float the valid display format codes are shown in the following table:

Display format	What is displayed
9	Digit or leading/trailing zero.
z	Digit, suppressed zeros (after decimal) if leading or trailing.
B	Spaces for suppressed zeros, minus (-) and plus (+) signs.

Numeric display format codes.

part 1 of 2

Display format	What is displayed
+	+ to left or right if value is positive (>0).
-	- to left or right if value is negative (<0).
P	Fixed decimal point.
K	Fixed decimal comma.
.	Layout decimal point.
,	Layout decimal comma.

Numeric display format codes.

part 2 of 2

Examples for numeric fields

The following table gives examples of display formats for numeric fields:

Display format	Input	Displayed
99999	12345	12345
	123	00123
	00123	00123
	123456	error: "too much data"
	-1234	error: "illegal numeric"
	123.45	12345 (no point defined)
zzzzz	12345	12345
	123	123
	00123	123
	123456	error: "too much data"
	-1234	error: "illegal numeric"
	123.45	12345 (no point defined)
Bzzzzz	123	123
	01234	1234
-zzzzz	123	123
	-123	-123
-zzzzzB	123	123
	-123	- 123
zzzzz-	123	123
	-123	123-

Examples of display format codes for numeric fields.

part 1 of 2

Display format	Input	Displayed
+zzzzz	123	+123
	-123	123
+-zzzzz	123	+123
	-123	-1234
-Bzzzz99	-123	- 123
	1234	1234
B-zzz99	-123	-123
	1234	1234
999P99	123	123.00
	123.45	123.45
	12.3	012.30
	1234.5	error: "too much data"
	123.456	error: "too much data"
zzz9P9zzz	123	123.0
	.8970	0.897
	012.120	12.12
zzz.zz.zzz	12345678	123.45.678
	12345	12.345
	1.234	1.234
	123.45.67	123.45.67

Examples of display format codes for numeric fields. part 2 of 2

Date

The following table shows the valid display format codes for fields with data type Date:

Display format	Explanation
d	Day number in one or two digits.
dd	Day number in two digits.
zd	Day number in two digits or one space and one digit.
aa	Two-letter abbreviation for day name.

Date display format codes. part 1 of 2

Display format	Explanation
AA	As aa, but uppercase.
Aa	As aa but initial letter is capitalized.
aa*	Full day name, lowercase.
AA*	As aa*, but uppercase.
Aa*	As aa*, but initial letter is capitalized.
AAA	Three-letter abbreviation for day name.
m	Month number in one or two digits.
mm	Month number in two digits.
zm	Month number in two digits or one space and one digit.
mmm	Three-letter abbreviation for month, lowercase.
MMM	As mmm, but uppercase.
mmm*	Full month name, lowercase.
MMM*	As mmm*, but uppercase.
Mmm*	As mmm*, but initial letter is capitalized.
w	Week number in one or two digits.
ww	Week number in two digits.
zw	Week number in two digits or one space and one digit.
yyyy	Calendar year in four digits.
yy	Calendar year in two digits.
xxxx	Fiscal year in four digits.
xx	Fiscal year in two digits.
Lcode	Number of days, months or years as a linear value, using one of the above codes.

Date display format codes. part 2 of 2

Examples for linear date fields

Display format	Value	Displayed
Lzd.yyyy	25 December, 1990	359.1990
Ldd.mm.yyyy	25 December, 1990	25.12.1990
Ldd.mm.yyyy	25 days and 11 months	25.11.0

Examples of display formats for linear date fields.

Examples for non-linear date fields

Display format	Displayed (1)	Displayed (2)
Mmm* d, yyyy	March 16, 1995	June 2, 1995
AA, MMM d	WEN, MAR 16	THU, JUN 2
dd/mm/yy	16/03/95	02/06/95
mm/dd/yy	03/16/95	06/02/95
d/m/yy	16/3/95	2/6/95
zd/zm/yy	16/ 3/95	2/ 6/95

Examples of display formats for date fields.

Time

The following table shows the valid display format codes for fields with data type Time:

Display format	Explanation
h	Hours in one or two digits.
hh	Hours in two digits.
zh	Hours in two digits or one space and one digit.
n	Minutes in one or two digits.
nn	Minutes in two digits.
zn	Minutes in two digits or one space and one digit.
s	Seconds in one or two digits.
ss	Seconds in two digits.
zs	Seconds in two digits or one space and one digit.
lh	Number of hours as linear value.
ln	Number of minutes as linear value.
ls	Number of seconds as linear value.
t	'Ticks' (1/100 seconds).

Time display format codes.

Examples for linear time fields

Display format	Value	Displayed
Lzdd.zh.zn.zs	27 days, 3 hours, 31 minutes	27.3.31.0
Lzdd.zh.zn.zs	71 minutes, 29 seconds	0.1.11.29

Examples of display formats for linear time fields.

Examples for non-linear time fields

Display format	Displayed
hh:nn	16:15 or 09:05
h:nn	16:15 or 9:05
hh:nn.ss	16:15.2 or 09:05.0
h:nn.s	16:15.2 or 9:05.3
zh:zm.zs	16:15.2 or 9: 5. 3

Examples of display formats for time fields.

Example for combined date and time fields

Display format	Displayed
dd MMM yyyy hh:nn:ss	2 APR 1991 14:15:39

Example of display format for date and time fields.

Boolean

The display formats you can specify for Boolean fields are shown in the following table:

Display format	Displayed
y/n	y if true; n if false
j/n	j if true; n if false
ja/nee	ja if true; nee if false

Display formats for Boolean fields.

If no display format for Boolean fields is specified, the value displayed is based on the packing code used for the field, as shown in the following table:

Packing code	Displayed
B1	1 if true; 0 if false
B2	T if true; F if false
B3	Y if true; N if false
B4	1 if true; 0 if false
None	T if true; F if false

Values displayed for Boolean fields based on the packing code used.

Chapter 13 Handling data in Proc

This chapter summarizes information that is found in the *Proc language and Handling data in Proc* chapters in the ► *Proc Language Reference Manual*. The following topics are covered:

- Date and time arithmetic.
- Date and time constants.
- Explicit type conversion.
- Extracting values from date and time data.
- Extracting values from numeric data.
- Extracting values from string data.
- Implicit type conversion.
- Operators.
- Substitution in string constants.
- Syntax string constants.

Date and time arithmetic

The following table shows examples of arithmetic operations with Date, Time and Datetime values:

Statement	Result
<code>;DATE_FLD is Date</code>	
<code>;DATM_FLD is Datetime</code>	
<code>;END_TIME is Datetime</code>	
<code>;START_TIME is Datetime</code>	
<code>;TIME_FLD is Time</code>	
<code>\$1 = END_TIME - START_TIME</code>	Elapsed time.
<code>\$2 = DATE_FLD</code>	Get a Date value.
<code>\$3 = TIME_FLD</code>	Get a Time value.
<code>\$4 = \$2 + \$3</code>	Make combined Datetime.
<code>\$5 = \$4 + \$1</code>	Add elapsed time from \$1 to \$4.
<code>\$5 = \$5 + 1s</code>	Add 1 second to Datetime in \$5.
<code>\$5 = \$5 + 1n</code>	Add 1 minute.
<code>\$5 = \$5 + 1n1s</code>	Add 1 minute and 1 second.
<code>DATE_FLD = DATE_FLD + 4d</code>	Add 4 days.
<code>DATM_FLD = DATM_FLD + 12h</code>	Add half a day (12 hours).
<code>\$1 = DATM_FLD + 7d12h</code>	Add one week and half a day.

Examples of arithmetic with date and time values.

Date and time constants

The following table shows the codes that can be used to create date and time constants:

Code	Meaning	Examples	Value (fraction of one day)
d	Day	1d	1 day
		3d	3 days
h	Hour	1h	1/24
		2h	1/12
		1d6h	1-1/4
n	Minute	1n	1/1,440
s	Second	1s	1/86,400
		1n2s	1-1/720
t	Tick	1t	1/8,640,000
		5t	1/1,728,000

Date and time constant codes.

Explicit type conversion

The functions shown in the following table can be used to explicitly convert data from one data type to another. These functions are summarized in chapter 15 *Proc functions*.

Source	Target	Function
String	Number	\$number
String	Date	\$date
String	Time	\$clock
String	Datetime	\$datim
String	Syntax string	\$syntax
Number*	Time	\$clock

Proc functions for explicit type conversion.

Table notes:

* A Number is converted to a String before conversion.

Extracting values from date and time data

The format for extracting values from Date, Time and Datetime values is:

source [*parameter*]

The date and time extraction parameters are described in the following table:

Parameter	Data extracted
clock	The time part of a Datetime.
H	Hour (24-hour clock).
N	Minutes.
S	Seconds.
T	Ticks (1/100 of a second).
date	The date part of a Datetime.
D	Day number.
M	Month number.
Y	Year (four digits).
X	Fiscal year (four digits).
W	Week number.*
mmm*	Month name spelled out, lowercase.
Mmm*	Month name spelled out, initial capital.
mmm	Three-letter month abbreviation, lowercase.
Mmm	Three-letter month abbreviation, initial capital.
A	Day of week (Monday = 1).
aa*	Day name spelled out, lowercase.
Aa*	Day name spelled out, initial capital.
aa	Two-letter abbreviation for day name, lowercase.
Aa	Two-letter abbreviation for day name, initial capital.
aaa	Three-letter abbreviation for day name, lowercase.
Aaa	Three-letter abbreviation for day name, initial capital.

Date and time extraction parameters.

Table notes:

* UNIFACE complies with the ISO 2015 standard for week numbering:

- Monday is day 1 in the week.
- Sunday is day 7 in the week.
- The rule for determining week 1 works as follows:
 - Week 1 begins on a Monday.
 - January 1 falls in week 1 if it is a Monday, Tuesday, Wednesday or Thursday.
 - January 1 falls in week 53 of the previous year if it is a Friday, Saturday or Sunday.

The following table shows examples of how to use the date and time extraction facilities of the Proc language:

Statement	Meaning	Result
delydate="08-nov-1961 10:30:00"		
delydate[A]	Day of week (numeric)	3
delydate[aa*]	Day spelled out	wednesday
delydate[H]	Hour part of time	10
\$1 = "08-nov-1961 22:00:00"		
\$2 = \$datim(\$1) [clock]	Convert \$1 to a Datetime and extract time	22:00:00
date2 = "1-jan-1993"		
\$1 = date2[W]	Week number	53
\$2 = date2[X]	Fiscal year	1992
\$3 = date2[Y]	Calendar year	1993
\$4 = date2[Mmm*]	Month spelled out	January

Examples using date and time extraction.

Extracting values from numeric data

The format for extracting values from Numeric and Floating Decimal Point values is:

source [parameter]

The numeric extraction parameters are described in the following table:

Parameter	Data extracted
[trunc]	Integer part of a value.
[fraction]	Fractional part of a value.
[round]	Value rounded to the units position.
[round, 0]	
[round, n]	If $n < 0$, round to n positions left of the decimal point. If $n > 0$, round to n positions right of the decimal point.

Numeric extraction parameters.

The following table shows examples of how to use the numeric extraction facilities of the Proc language:

Statement	Meaning	Result
\$25 = 123.76		
\$1 = \$25[fraction]	Extract the fractional value	0.76
\$2 = \$25[trunc]	Extract the truncated value	123
\$3 = \$25[trunc] + .11	Extract the truncated value and add 0.11	123.11
\$25 = 1897.7652		
\$1 = \$25[round]	Round the value in \$25	1898
\$25 = 1897.7652		
\$25 = \$25[round, -2]	Round the value in \$25	1900.0000
25 = 1897.7652		
\$25 = \$25[round, 2]	Round the value in \$25	1898.7700

Examples using numeric extraction.

Extracting values from string data

The format for extracting values from String and Special String values is:

source [*start* {:*num* | ,*end*}]

The syntax of string extraction is explained in the following table:

Parameter	Data extracted
<i>start</i>	Position number from which to start extracting.
<i>num</i>	The number of positions to extract from <i>start</i> .
<i>end</i>	Position number at which to stop extracting.

String extraction parameters.

The following table shows examples of how to use the string extraction facilities of the Proc language:

Statement	Meaning	Result
<code>NAME = "HOLLERITH"</code>		HOLLERITH
<code>\$1 = NAME[4,8]</code>	Extract positions 4 through 8	LERITH
<code>\$1 = NAME[1:3]</code>	Extract positions 1 through 3	HOL
<code>\$10 = 2</code>		
<code>\$1 = NAME[\$10:4]</code>	Extract positions 2 through 5	OLLE
<code>\$1 = NAME[3]</code>	Extract positions 3 through 9	LLERITH
<code>length NAME</code>	Get length of NAME	9
<code>\$1 = NAME[\$result]</code>	Extract last character of NAME	H
<code>scan NAME, 'LL?'</code>	Get position of string matching 'LL?'	3
<code>\$10 = \$result+2</code>	Get position of char following 'LL'	5
<code>\$1 = NAME[\$10:1]</code>	Extract character following 'LL'	E
<code>FIELD = "AMSTER123DAM"</code>	FIELD contains letters and numbers	
<code>scan FIELD, '#'</code>	Get position of first digit in FIELD	7
<code>FIELD = FIELD[\$result]</code>	Remove leading non-digits in FIELD	123DAM
<code>\$1 = \$number(FIELD)</code>	Convert to number	123
<code>\$1 = "Amsterdam123jim"</code>		
<code>scan \$1, '#'</code>	Get position of first digit in \$1	10
<code>if (\$result > 0)</code>	\$1 contains numeric data	
<code> \$3 = \$1[\$result]</code>	Remove leading non-digits in \$1	123jim
<code> \$2 = \$number(\$3)</code>	Convert to number	123
<code>else</code>		
<code> \$2 = ""</code>	No numeric data in \$1	
<code>endif</code>		

Examples using string extraction.

Implicit type conversion

The following table shows how assignments between mixed data types behave:

Source	Target	Result
String	Numeric	String converted to a Numeric.
String	Date	String converted to a Date using a format of ccyymmdd.*
String	Time	String converted to a Time using a format of hhnnssst.*
String	Datetime	String converted to a Datetime using a format of ccyymmddhhnnssst.*
String	Boolean	<ul style="list-style-type: none"> • If String starts with 0, F or N, converted to False. • If String does not start with 0, F or N, converted to True.
Numeric	Boolean	<ul style="list-style-type: none"> • If Numeric is 0, converted to False. • If Numeric is not 0, converted to True.
Date, Time, Datetime	Numeric	Date, Time or Datetime converted to a linear Datetime number of days.
Date, Time, Datetime	Boolean	<ul style="list-style-type: none"> • If Date, Time or Datetime is null, converted to False. • If Date, Time or Datetime is not null, converted to True.

Implicit type conversion.

Table notes:

* UNIFACE expects the source String to be formatted like the default date, time or datetime format defined in the language setup.

Operators

The following table shows the arithmetic, relational and logical operators recognized by the Proc language:

Type	Priority	Operator	Description
Arithmetic	6	*	Multiplication
		/	Division
		%	Modulus
Relational	5	+	Addition
		-	Subtraction
		<	Less than
		<=	Less than or equal to
Logical	3	!=	Not equal to
		=	Equal to
		==	Equal to
		>=	Greater than or equal to
		>	Greater than
		!	Logical NOT
Proc operators.	2	&	Logical AND
		1	

Substitution in string constants

The following rules apply to substitution in string constants:

- Two percent symbols (%%) followed by the name of a field, variable or function are replaced by the current value of that field, variable or function.
- To prevent ambiguity in interpreting a field, variable or function name in a string, follow the name with three percent symbols (%%%). For example:

```
message "%$GLOBVAR%%XYZ"
```
- Use two percent symbols followed by a double quotation mark ("%") to include a double quotation mark in the resulting string.
- A space following two percent symbols is automatically removed by the Proc compiler; the two percent symbols appear unchanged.
- Use two percent symbols followed by a caret ("%^") to include a carriage return in the resulting string.

The following table shows examples of string substitution:

Example	Result
<pre>; the current entity is ENT1 \$1 = 1111 \$\$GVAR1 = 999 FIELDA.ENT1 = "AAAA"</pre>	
<pre>"Values %%FIELDA %%\$1 %%\$entname"</pre>	Values AAAA 1111 ENT1
<pre>"FIELDA contains %%FIELDA."</pre>	FIELDA contains AAAA
<pre>"FIELDA contains %%FIELDA%%."</pre>	FIELDA contains AAAA.
<pre>"FIELDA contains %%"%%FIELDA%%"</pre>	FIELDA contains "AAAA"
<pre>"%%FIELDA.001"</pre>	Compile error: 1000 - Field 'FIELDA.001' not found.
<pre>"%%FIELDA%%%.001"</pre>	AAAA.001
<pre>"%%\$1%%001"</pre>	1111001
<pre>"%%\$\$GVAR1%%xyz"</pre>	999xyz
<pre>"%%\$USER"</pre>	KAYE
<pre>"%% \$USER"</pre>	%%\$USER
<pre>"%% \$USER"</pre>	%% \$USER
<pre>"% % \$USER"</pre>	% % \$USER
<pre>"Roses are red%^Violets are blue"</pre>	Roses are red Violets are blue

Examples of string substitution.

Syntax string constants

A syntax string is a group of characters and pattern-matching codes enclosed in single quotation marks ('). The codes allowed in a syntax string are shown in the following table:

Syntax code	Explanation
#	One digit (0-9).
#*	0- <i>n</i> digits.
&	One letter (A-Z, a-z).
&*	0- <i>n</i> letters.
@	One letter, digit or underscore (_).
@*	0- <i>n</i> letters, digits or underscores.
?	One ASCII character.
?*	0- <i>n</i> ASCII characters.
*	0- <i>n</i> ASCII characters (same as ?*).
A-Z	That uppercase letter (A, B, C and so on).
a-z	That letter in either case (A, a, B, b, and so on).
x	Any ASCII character except the syntax code characters (#, *, &, @, ?, (, ., % and ^).
%x	Any ASCII character, with no special meaning.
(any)	The syntax string <i>any</i> is optional. A syntax check is only done if data is present.
%%^	Carriage return or line feed.

Syntax codes for pattern matching.

The following table shows examples using syntax strings:

Example	Result
<code>if ('#' = "123")</code>	False
<code>if ('##*' = "123")</code>	True
<code>\$1 = 123</code>	
<code>if ('##*' = "%%"\$1")</code>	True
<code>if ('&###' = "1234")</code>	False
<code>if ('@###' = "1234")</code>	True
<code>if ('?' = "A")</code>	True
<code>if ('??*' = "A")</code>	True
<code>if ('?' = "ABC")</code>	False
<code>if ('??*' = "ABC")</code>	True

Examples of syntax strings.

Chapter 14 Proc statements

This chapter summarizes the use of the Proc statements that make up the Proc language. See the *Proc statements* chapter in the ► *Proc Language Reference Manual* for complete information.

Name

=
Assign the value of an expression to a destination.

Synopsis

`{compute} destination{/init} = expression | constant`

Return value

None.

Name

`addmonths`
Add the specified number of months to the date.

Synopsis

`addmonths amount, "date"{, "start_date"}`

Return value

The `addmonths` statement does not affect `$status`. The resulting date is stored in `$result`. The data type of `$result` depends on the data type of the `date` argument:

- If `date` is given as a constant string, `$result` is returned as a Datetime field with the time part set to 0.
- If `date` is given as a field, global variable or local variable, the data type in `$result` depends on the data type of `date`.

Name `apexit`
Exit the application immediately.

Synopsis `apexit`

Return value None.

Name `askmess`
Display a message and wait for the user's response.

Synopsis `askmess(/nobeep){/question|/info|/warning|/error}
"message" {"reply_1", ..., "reply_n"}`

Return value When no *replies* are provided, after `askmess`, `$status` is set to:

- 0 if the reply was equivalent to 'No'.
- 1 if the reply was equivalent to 'Yes'.

When *replies* are defined, after `askmess`, `$status` is set to the number of the reply entered by the user. The first reply is 1, the second reply is 2 and so on.

Name `blockdata`
Define a constant block of text.

Synopsis `label: blockdata char
text
...
...
...
char`

Return value None.

Name `break`
Unconditionally exit a `repeat` or `while` loop.

Synopsis `break`

Return value None.

Name `call`
Execute the specified Proc module.

Synopsis `call entry_name`

Return value After `call`, the value of `$status` is the value returned by the Proc module that was called. If no value is returned or if there is no `return` statement in the called module, `$status` contains 0.

SEE ENTRY
Name `clear`
Clear data from the form.

Synopsis `clear(/e "entity"){ source}`

Return value The values commonly returned by `clear` in `$status` are shown in the following table:

Value	Meaning
0	Data was successfully cleared.
-3	Exceptional I/O error (hardware or software).
-16	Network error (unknown).

Values returned by `clear` in `$status`.

Name `close`
Log off from the specified DBMS path or from all paths.

Synopsis `close (" $path ")`

Return value The values commonly returned by `close` in `$status` are shown in the following table:

Value	Meaning
0	The path was successfully closed.
-1	A network path was specified in <i>path</i> .
-3	Exceptional I/O error (hardware or software).
-16	Network error (unknown).

Values returned by `close` in `$status`.

Name `clrmess`
Clear all text from the message frame.

Synopsis `clrmess`

Return value None.

Name `commit`
Commit a transaction to a DBMS or path.

Synopsis `commit { "dbms" | " $path " }`

Return value The values commonly returned by `commit` in `$status` are shown in the following table:

Value	Meaning
0	The data was successfully committed.
-3	Exceptional I/O error (hardware or software).
-16	Network error (unknown).

Values returned by `commit` in `$status`.

Name `compare`
Compare fields of two adjacent occurrences.

Synopsis `compare (/previous | next) (field1 {,field2,...,fieldn}) {from "entity" }`

Return value The `compare` statement sets both `$status` and `$result`. The values that can be returned in `$status` are shown in the following table:

Value	Meaning
0	Success. This can be returned even when there is no next or previous occurrence.
-1	One or more <i>fields</i> could not be accessed. This can occur when <i>entity</i> is contained in a field or variable and the field or variable does not contain the correct entity name (or one that does not exist). In this situation, <code>\$result</code> is always 0.

Values returned by `compare` in `$status`.

The result of the comparison is stored in `$result`. The possible values are shown in the following table:

Value	Meaning
1	Perfect match of all specified fields.
0	Fields do not match. This value is always returned if <code>\$status</code> is -1.
-1	No previous or next occurrence (error situation).

Values returned by compare in `$result`.

Name `creocc`
Create an empty occurrence of the specified entity.

Synopsis `creocc "entity", sequence_number`

Return value The values returned in `$status` by the `creocc` statement are shown in the following table:

Value	Meaning
>0	Sequence number of the created occurrence.
-1	An occurrence could not be created.

Values returned by `creocc` in `$status`.

Name `debug`
Start the interactive debugger.

Synopsis `debug`

Return value None.

Name `delete`
Delete the current occurrence from the database.

Synopsis `delete`

Return value The values commonly returned by `delete` in `$status` are shown in the following table:

Value	Meaning
0	Data was successfully deleted.
-3	Exceptional I/O error (hardware or software).
-5	Update request for non-updatable occurrence. The following message is displayed: 2004 - No modifications allowed on occurrence of this entity.
-6	Exceptional I/O error on write request.
-11	Occurrence currently locked.
-16	Network error (unknown).

Values returned by `delete` in `$status`.

Name `delitem`
Delete an item from a list.

Synopsis `delitem list, n`
`delitem/id list, index`

Return value The possible values of `$status` following `delitem` are shown in the following table:

Value	Meaning
>0	The item number in <code>list</code> .
0	Unsuccessful.

Values returned by `delitem` in `$status`.

Name `discard`
Remove one or more occurrences from the form and the hitlist.

Synopsis `discard {"entity"} {,from_occurrence_number
{,to_occurrence_number}}`

Return value The values returned by `discard` are shown in the following table:

Value	Meaning
>0	The sequence number of the occurrence that is now current after discarding an occurrence.
0	No next occurrence is available. That is, the occurrence is the last occurrence or there is only one occurrence.
-1	The <code>from_occurrence_number</code> was greater than the number of available occurrences.
-2	The entity name does not exist.

Values returned by `discard` in `$status`.

Name `display`
Present the form on the screen as display-only.

Synopsis `display(/menu) {field}`

Return value The values returned by `display` are shown in the following table:

Value	Meaning
0	Success.
-1	The form specified could not be found. The following message is displayed: 0113 - Form paint is empty; cannot edit, display or print.
-1	The field does not exist. The following message is displayed: 0114 - Failed to start edit on field <code>field</code> .
-16	The application is running in batch mode. Use a test on <code>\$batch</code> to avoid this. The following message is displayed: 0016 - Terminal input aborted; not allowed in batch mode.

Values returned by `display` in `$status`.

Name `done`
Exit from the Proc module without changing `$status`.

Synopsis `done`

Return value The value of `$status` remains unchanged.

Name `edit`
Present the form and start the structure editor for user input.

Synopsis `edit{/menu|/nowander} {field}`

Return value The values returned by `edit` are shown in the following table:

Value	Meaning
0	Success.
-1	The <code>edit</code> statement is not in an EXECUTE trigger or there are no promptable fields on the form. The following message is displayed: 0164 - Edit instruction only allowed in EXECUTE trigger
-16	An <code>edit</code> is attempted when in batch mode. Use a test on <code>\$batch</code> to avoid this.

Values returned by `edit` in `$status`.

Name `eject`
Eject a page when printing.

Synopsis `eject`

Return value The values returned by `eject` are shown in the following table:

Value	Meaning
-1	The form was not being printed when the <code>eject</code> statement was encountered (that is, <code>\$printing</code> is 0). The <code>eject</code> statement is ignored.
0	Any other situation.

Values returned by `eject` in `$status`.

Name `end`
Mark the end of a Proc module.

Synopsis `end`

Return value The value of `$status` remains unchanged.

Name `entry`
Label the start of a Proc module.

Synopsis `entry entry_name`

Return value None.

Name `erase`
Activate the DELETE or DELETE UP trigger for all occurrences in the form.

Synopsis `erase{/e {"entity"}}`

Return value The values commonly returned by `erase` in `$status` are shown in the following table:

Value	Meaning
1	<code>erase</code> is not allowed. (For example, the form was activated with <code>run/query</code> .)
0	Data was successfully erased.
-2	Occurrence not found.
-3	Exceptional I/O error (hardware or software).
-5	Update request for non-updatable occurrence.
-6	Exceptional I/O error on write request.
-11	Occurrence currently locked.
-16	Network error (unknown).

Values returned by `erase` in `$status`.

SEE CALL,

Name `exit`
Exit the current form and return to the previous or specified form.

Synopsis `exit ((expression))[, "form"]`

Return value The result of evaluating *expression* is placed in `$status`. If *expression* is omitted, `$status` defaults to 0.

Name `field_syntax`
Set the syntax attributes of the specified field.

Synopsis `field_syntax field, "attribute_1 {, ..., attribute_n}"`

Attribute	Description
NDI	Do not display this field.
NED	No edit allowed in this field.
NPR	Do not prompt this field.

Shorthand codes for field syntax definitions.

Reset the syntax with `field_syntax field, ""`.

Return value None.

Name `field_video`
Set the video attributes of the specified field.

Synopsis `field_video field, "DEF|NON|attribute_1{, ..., attribute_n}"`

- Use **DEF** to set the default video attributes for *field*.
- Use **NON** to set no video attributes for the field.
- Use one or more of the video attributes shown in chapter 4 *Video attributes* (except **BOR**), separated by commas.

Return value None.

Name `filebox`
Display a file selection box.

Synopsis `filebox (/save) {"filter" } [, "default_path"]`

Return value The values returned by `filebox` are shown in the following table:

Value	Meaning
>0	The number of files selected.
0	No file was selected by the user.
<0	An error occurred.

Values returned by `filebox` in `$status`.

If the user selects a file, the `filebox` statement sets `$result` to contain the fully qualified name of the file.

Name `filedump`
Write the contents of the specified field to the specified file.

Synopsis `file[_]dump(/append) (/raw) (/image) field, "{path}file"`

Return value The values returned by `filedump` are shown in the following table:

Value	Meaning
>=0	The number of bytes written to the file.
-1	An error occurred, for example, the file could not be created or the user does not have 'write' permission for an existing file.

Values returned by `filedump` in `$status`.

Name `fileload`
Read the contents of the specified file into the specified field.

Synopsis `file[_]load [/raw] [/image] "[path]file", destination`

Return value The values returned by `fileload` are shown in the following table:

Value	Meaning
>=0	The number of bytes read from the file.
-1	The file cannot be opened.

Values returned by `fileload` in `$status`.

Name `getitem`
Copy an item from a list to a field or variable.

Synopsis `getitem target, list, n`
`getitem/id target, list, index`

Return value The possible values of `$status` following `getitem` are shown in the following table:

Value	Meaning
>0	The sequence number of the list item that was copied.
0	Unsuccessful; <code>target</code> is blank.

Values returned by `getitem` in `$status`.

Name `getlistitems`
Copy items from a list into a field or variable.

Synopsis `getlistitems list, target_value`
`getlistitems/id list{, target_value}{, target_representation}`
`getlistitems/id {/field | /local | /global} list`
`getlistitems/occ list, "entity"`

Return value The possible values of `$status` following `getlistitems` are shown in the following table:

Value	Meaning
>0	The number of items moved.
0	Unsuccessful.

Values returned by `getlistitems` in `$status`.

Name `goto`
Branch unconditionally to the specified label.

Synopsis `goto label`

Return value None.

Name `help`
Display the specified message in a help box and wait for the user's response.

Synopsis `help [/nborder] help_message {, vertical_pos, horizontal_pos`
`{, vertical_size, horizontal_size}}`

Return value The values returned by `help` are shown in the following table:

Value	Meaning
1	The help form was left with ^ACCEPT.
0	The help form was left with ^QUIT.
-1	The help form USYS:USYSTXT could not be found.
-2	The help form is not correct. One of the following messages is displayed: 0019 - Form USYS:USYSTXT has wrong version; you must recompile it 0020 - File USYS:USYSTXT not recognized as application or form.

Values returned by help in \$status.

Name *if*
Define an *if/else/endif* conditional block.

Synopsis *if* (*condition*)
 statements
{else
 statements}
endif

Return value None.

Name *length*
Return the number of characters in the specified string.

Synopsis *length string*

Return value The value of \$status remains unchanged. \$result is set to the number of characters in the string.

Name *lock*
Lock the current occurrence in the database.

Synopsis *lock*

Return value The values commonly returned by *lock* in \$status are shown in the following table.

Value	Meaning
0	The occurrence cannot be modified. (For example, during a run/query.)
-1	There is no active occurrence.
-2	Occurrence not found. Occurrence removed since last retrieve.
-3	The hit for the occurrence does not exist.
-5	There is no hit for the occurrence or the occurrence is read-only (cannot be locked). One of the following messages is displayed: 2008 - Occurrence cannot be modified due to fetch error. 2004 - No modifications allowed on occurrence of this entity.
-10	Occurrence has been modified or removed since it was retrieved; a reload should be executed.
-11	Occurrence currently locked.

Values returned by *lock* in \$status.

Other DBMS driver error codes may be returned in certain circumstances. Refer to the ► *DBMS Specific Guide* module for your DBMS.

Name **lookup**
Count the number of occurrences that match the current profile.

Synopsis **lookup**

Return value The values commonly returned by **lookup** in **\$status** are shown in the following table:

Value	Meaning
>=0	The number of hits that match the profile.
-1	Record not found: end of file encountered.
-2	Occurrence not found. (The table or file has no occurrences.)
-3	Exceptional I/O error (hardware or software).
-4	Open request for table or file failed. The file or table is not painted or does not exist. A subsequent I/O request to the table or file will return -1.
-15	UNIFACE network error.
-16	Network error (unknown).

Values returned by **lookup** in **\$status**.

Name **lowercase**
Convert a string to lowercase.

Synopsis **lowercase source, destination**

Return value None.

Name **macro**
Place structure editor input in the event input buffer.

Synopsis **macro(/exit) "character_sequence"**

Return value **\$status** is always set to 0.

Name **message**
Display the specified string in the message area.

Synopsis **message(/error | /warning | /info | /hint){/nobeeep} "string"**

Return value None.

Name **nodebug**
Stop the interactive debugger.

Synopsis **nodebug**

Return value None.

Name **numgen**
Increment the specified counter.

Synopsis **numgen "counter", increment [, "library"]**

Return value The values commonly returned by **numgen** in **\$status** are shown in the following table:

Value	Meaning
0	A value was successfully generated. In this case, \$result is set to the new number.
-1	The counter went out of range.

Values returned by **numgen** in **\$status**.

Name	numset Initialize the value of the specified counter.
Synopsis	<code>numset "counter", init_value (, "library")</code> The range of <i>init_value</i> is -2147483648 through 2147483647.
Return value	The values commonly returned by <i>numset</i> in <i>\$status</i> are shown in the following table:
Value	Meaning
0	The counter was successfully initialized.
-1	<i>init_value</i> was out of range.
Values returned by <i>numset</i> in <i>\$status</i> .	

Name	open Open the specified DBMS or path.
Synopsis	<code>open "login_parameters", "path{/net}"</code> The login information in <i>login_parameters</i> is specified as: (<i>name</i>) (<i>username</i>) (<i>password</i>) where:
	<ul style="list-style-type: none"> • <i>name</i> is: <ul style="list-style-type: none"> • For a DBMS driver, the database name or the DBMS network server name, when appropriate. • For a network driver, the network node name or the network server name. • <i>username</i> is the logon name for the DBMS or network driver. • <i>password</i> is the password for the <i>username</i> on the driver. <p>Each of the parameters <i>name</i>, <i>username</i> and <i>password</i> may be replaced by a question mark (?) or completely omitted.</p>

Return value	The values commonly returned by <i>open</i> in <i>\$status</i> are shown in the following table:
Value	Meaning
0	The path was successfully opened.
-3	Exceptional I/O error (hardware or software).
-4	Open request for table or file failed. The file or table is not painted or does not exist. A subsequent I/O request to the table or file will return -1.
-16	Network error (unknown).
Values returned by <i>open</i> in <i>\$status</i> .	

Name	perform Call the specified 3GL function.
Synopsis	<code>perform(/noterm) "function"</code>
Return value	<i>\$status</i> is set to the value returned by <i>function</i> or to -1 if <i>function</i> could not be found. It is not a good idea to return -1 in a 3GL function, since this cannot be distinguished from UNIFACE not being able to find <i>function</i> .
Name	pragma Interpret profile characters in the Proc module as 'maybe' characters.
Synopsis	<code>pragma v5profile</code>
Return value	None.

Name `print`
 Activate printing.

Synopsis `print{/ask} {"print_job_model"} {, "print_mode"}`
 Possible values for `print_mode` are shown in the following table:

Value	Meaning
A	All, that is, print the form and all data in the hitlist.
C	As A, but clear the data from memory after printing. This should be used when you are writing reports.
F	Print the entire form and all data currently in it.
S	Print what is on the screen.

Values for the `print_mode` parameter on the `print` statement.

If `print_job_model` is not included, UNIFACE uses the default `PRINTER`; if `print_mode` is not included, UNIFACE uses the default `A`.

Return value The values returned by the `print` statement in `$status` are shown in the following table:

Value	Meaning
0	Success.
-1	UNIFACE could not print, for example: <ul style="list-style-type: none"> • Printing is already being performed (<code>\$printing</code> is 1). • ^QUIT was used in the Print form. • An invalid <code>print_mode</code> was used (not one of A, C, F or S).

Values returned by `print` in `$status`.

The name of the print file created is available in `$result`.

Name `print_break`
 Print the specified break frame.

Synopsis `print_break "frame_name"`

Return value The values returned by the `print_break` statement are shown in the following table:

Value	Meaning
1	The <code>FRAME GETS FOCUS</code> trigger for the specified break frame returned a positive value.
0	The <code>FRAME GETS FOCUS</code> trigger for the specified break frame returned a negative value.
-1	Not printing or inside a header or footer.

Values returned by `print_break` in `$status`.

Name `pulldown`
 Activate or load the specified menu into the menu area.

Synopsis `pulldown{/load} {"menu_bar_name"}`

Return value The values returned by the `pulldown` statement are shown in the following table:

Value	Meaning
-1	The menu does not exist.
0	The <code>OPTION</code> trigger of the selected menu item is empty.
Others	The value returned by the <code>OPTION</code> trigger of the selected menu item.

Values returned by `pulldown` in `$status`.

Name putitem
Add or replace an item in a list.

Synopsis putitem *list, n, source*
putitem/*id list, index, source*

Return value The possible values of \$status following putitem are shown in the following table:

Value	Meaning
>0	The sequence number of the list item that was replaced or added.
0	Unsuccessful.

Values returned by putitem in \$status.

Name putlistitems
Copy data from a specified source to the items of a list.

Synopsis putlistitems *list, source field*
putlistitems/*id list, (source_value) {, source_representation}*
putlistitems/*id {/field | /local | /global} list*
putlistitems/*occ list, "entity"*

Return value The possible values of \$status following putlistitems are shown in the following table:

Value	Meaning
>=0	The number of items copied.
-1	Unsuccessful.

Values returned by putlistitems in \$status.

Name putmess
Append text to the message frame.

Synopsis putmess {"text"}

Return value None.

Name read
Build a hitlist (if it does not exist) and fetch an occurrence from the hitlist.

Synopsis read{/lock} {using *expression1*} %\
{u_where (*expression2*) |where "*expression3*" } \
{order by "*field_1* (desc) {, *field_2* (desc) ... *field_n* (desc) }" }

Return value The values commonly returned by read in \$status are shown in the following table:

Value	Meaning
0	The occurrence was successfully read.
-1	Record not found: end of file encountered.
-2	Occurrence not found. (Usually, the table is empty.)
-3	Exceptional I/O error (hardware or software).
-4	Open request for table or file failed. The file or table is not painted or does not exist. A subsequent I/O request to the table or file will return -1.
-15	UNIFACE network error.
-16	Network error (unknown).

Values returned by read in \$status.

Name refresh
Redraw the screen.

Synopsis refresh

Return value None.

Name release
Release database controls.

Synopsis release [/mod]
release/e [/mod] {"entity"}

Return value If *entity* does not exist, *\$status* is not set and the following message is displayed:

0145 - Entity *entity* not available.

If *entity* does exist, the values commonly returned by **release** in *\$status* are shown in the following table:

Value	Meaning
0	Data successfully released.
-3	Exceptional I/O error (hardware or software).
-16	Network error (unknown).

Values returned by **release** in *\$status*.

Name reload
Reread and lock the current occurrence from the database.

Synopsis reload(/nolock)

Return value If the occurrence exists, the values commonly returned by **reload** in *\$status* are shown in the following table:

Value	Meaning
0	Data successfully reloaded.
-1	Record not found: end of file encountered.
-2	Occurrence not found.
-3	Exceptional I/O error (hardware or software).
-11	Occurrence currently locked.
-16	Network error (unknown).

Values returned by **reload** in *\$status*.

Name remocc
Mark an occurrence of the specified entity for deletion on the next store.

Synopsis remocc "entity", *sequence_number*

where *sequence_number* is the sequence number (in the form) of the occurrence to be removed:

- If *sequence_number* is less than 0, the last occurrence in the form structure is removed.
- If *sequence_number* equals 0, the current occurrence of *entity* is removed (default).
- If *sequence_number* is greater than the number of occurrences of *entity*, *\$status* is set to -1 and no occurrence is removed.

Return value

The values returned by `remocc` in `$status` are shown in the following table:

Value	Meaning
>=0	The sequence number of the occurrence that became current after removing an occurrence.
-1	The occurrence could not be removed: <ul style="list-style-type: none"> • <i>entity</i> does not exist. • <i>entity</i> is the outer entity of a Record form. • <i>sequence_number</i> is greater than the number of occurrences of <i>entity</i>.

Values returned by `remocc` in `$status`.

Name

repeat
Define a `repeat/until` loop.

Synopsis

```
repeat
  statement
  {statements}
until (expression)
```

Return value

None.

Name

reset
Reset the value of the specified Proc function to 0.

Synopsis

```
reset $function
```

Return value

The values returned by `reset` in `$status` are shown in the following table:

Value	Meaning
0	The function was successfully reset.
-1	The function cannot be reset.

Values returned by `reset` in `$status`.

Name

retrieve
Activate the READ trigger for the first outermost entity and all related entities, or for a specific entity.

Synopsis

```
retrieve(/e ("entity"))
```

Return value

The values commonly returned by `retrieve` in `$status` are shown in the following table:

Value	Meaning
0	Data was successfully retrieved.
-1	Record not found: end of file encountered.
-2	Occurrence not found. (No hits were found in the table.)
-3	Exceptional I/O error (hardware or software).
-4	Open request for table or file failed. The file or table is not painted or does not exist. A subsequent I/O request to the table or file will return -1.
-15	UNIFACE network error.
-16	Network error (unknown).

Values returned by `retrieve` in `$status`.

Name `retrieve/o`
 Attempt to retrieve an occurrence of an entity using the current primary key value.

Synopsis `retrieve/o ["entity_name"]`

Return value The values commonly returned by `retrieve/o` in `$status` are shown in the following table:

Value	Meaning
4	The occurrence was found in the form. The current occurrence is removed and the cursor repositioned on the found occurrence.
3	The occurrence was found among the removed occurrences; it was un-removed.
2	The entity is painted as a foreign entity and one hit was found in the database.
1	The entity is painted as a foreign entity with coding in the WRITE UP trigger and the key value was not found during the database lookup. It is assumed that this is a new occurrence.
0	A new occurrence was created.
-1	Record not found: end of file encountered.
-2	The entity is painted as a foreign entity and the key value was not found during the database lookup.
-3	Exceptional I/O error (hardware or software).
-4	Open request for table or file failed. The file or table is not painted or does not exist. A subsequent I/O request to the table or file will return -1.
-7	The key exists in the database but was not found in the hitlist. This occurs when the user tries to enter a duplicate key.
-11	Occurrence currently locked.
-14	The entity is painted as a normal 'down' entity and multiple hits were found during the database lookup (ambiguous key).
-15	The entity is painted as a foreign entity and multiple hits were found during the database lookup.
-16	Network error (unknown).

Values returned by `retrieve/o` in `$status`.

Name `retrieve/x`
 Retrieve an additional occurrence of the specified entity without discarding the hitlist.

Synopsis `retrieve/x "entity"`

Return value The values commonly returned by `retrieve/x` in `$status` are shown in the following table:

Value	Meaning
5	The entity is painted as a foreign entity and one hit was found in the database.
4	The occurrence was found in the form. The current occurrence is removed and the cursor repositioned on the found occurrence.
3	The occurrence was found among the removed occurrences; it was un-removed.
1	The entity is painted as a foreign entity with coding in the WRITE UP trigger and the key value was not found during the database lookup. It is assumed that this is a new occurrence.
0	The occurrence does not exist.
-3	Exceptional I/O error (hardware or software).
-5	The key exists in the database but was not found in the hitlist. This occurs when the user tries to enter a duplicate key.
-11	Occurrence currently locked.
-14	The entity is painted as a normal 'down' entity and multiple hits were found during the database lookup (ambiguous key).
-15	The entity is painted as a foreign entity and multiple hits were found during the database lookup.
-16	Network error (unknown).

Values returned by `retrieve/x` in `$status`.

Name `return`
Exit from the Proc module, optionally returning a value.

Synopsis `return ((expression))`

Return value If *expression* is present, its value is placed in `$status`. If *expression* is not present, `$status` is set to 0.

Name `rollback`
Back out of the transaction (if supported by DBMS).

Synopsis `rollback {"dbms" | "$path"}`

Return value The values commonly returned by `rollback` in `$status` are shown in the following table:

Value	Meaning
0	Data was successfully rolled back.
-3	Exceptional I/O error (hardware or software).
-16	Network error (unknown).

Values returned by `rollback` in `$status`.

Name `run`
Activate the specified form.

Synopsis `run (/display) | (/query) "form"
{,vertical_pos, horizontal_pos {,vertical_size, horizontal_size}}`

Return value The `run` statement sets `$status` to the value returned by the EXECUTE trigger (of the form that was activated) if it contains a `return` or `exit` statement. The default values returned by `run` (that is, if no `return` or `exit` statements are present) are shown in the following table:

Value	Meaning
-1	<i>form</i> could not be found.
0	<i>form</i> did not contain an <code>edit</code> or <code>display</code> statement in the EXECUTE trigger.
9	The user left <i>form</i> with ^ACCEPT.
10	The user left <i>form</i> with ^QUIT.

Values returned by `run` in `$status`.

Name `scan`
Return the starting position of the specified profile within a field or variable.

Synopsis `scan string, profile`
where:

- string* is a field or variable.
- profile* is either a string or a syntax string.

Return value

The value of `$status` remains unchanged. The values that can be returned in `$result` are shown in the following table:

Value	Meaning
>0	Starting position in <i>string</i> of the first match.
0	<i>profile</i> not found or <i>string</i> is a null string.

Values returned by scan in `$result`.

Name

`selectdb`
Calculate aggregate values for specified fields in the database.

Synopsis

```
selectdb (select_1 {, select_2, ... , select_n}) %\  
  {from entity} %\  
  {using index} %\  
  {u_where (clause)} %\  
  to (destination_1, ... , destination_n)
```

Each *select* phrase is of the form:

function (field) | field

where *function*, if used, is one of the ones shown in the following table. If the form without *function* is used, UNIFACE transports the value of the specified field from the last selected record.

Function	Calculation performed
<code>ave</code>	Sum of all values in the named field in the database divided by <code>count</code> .
<code>count</code>	Number of fields in the database that are filled.
<code>max</code>	Largest value in the named field in the database (null is ignored).
<code>min</code>	Smallest value in the named field in the database (null is ignored).
<code>sum</code>	Sum of all values in the named field in the database.

`selectdb` functions.

The functions that can be used with each UNIFACE data type are shown in the following table:

Data type	ave	count	max	min	sum
String	No	Yes	No	No	No
Raw	No	Yes	No	No	No
Numeric	Yes	Yes	Yes	Yes	Yes
Float	Yes	Yes	Yes	Yes	Yes
Date	No	Yes	Yes	Yes	No
Time	No	Yes	Yes	Yes	No
Datetime	No	Yes	Yes	Yes	No
Boolean	No	Yes	Yes	Yes	No

`selectdb` functions and data types.

Return value

The values commonly returned by `selectdb` in `$status` are shown in the following table:

Value	Meaning
>=0	The number of occurrences that matched <i>clause</i> .*
-1	<i>field</i> does not exist, or <i>function</i> cannot be used with this field type.
-3	Exceptional I/O error (hardware or software).
-15	UNIFACE network error.
-16	Network error (unknown).

Values returned by `selectdb` in `$status`.

Table notes:

* When one of the `selectdb` functions is used and the number of occurrences that matched the *clause* is 0, `$status` is set to a different value on a record level DBMS than on a field level DBMS. The following values are returned in `$status`:

- 0 for a record level DBMS.
- 1 for a field level DBMS. (In this case, the function will always return exactly one row.)

Name `set`
Set the value of the specified Proc function to 1.

Synopsis `set $function`

Return value The values returned by the `set` statement are shown in the following table:

Value	Meaning
1	The function was successfully set.
-1	The function cannot be set.

Values returned by `set` in `$status`.

Name `setocc`
Make a specific occurrence the current occurrence.

Synopsis `setocc "entity", sequence_number`

Return value The values returned in `$status` by `setocc` are shown in the following table.

Value	Meaning
>0	The sequence number of the new current occurrence in <code>entity</code> .
-1	No more occurrences to set to.

Values returned by `setocc` in `$status`.

Name `skip`
Skip the specified number of lines when printing.

Synopsis `skip (expression)`

Return value The values returned by `skip` are shown in the following table:

Value	Meaning
0	Success or <code>skip</code> was ignored.
-1	UNIFACE is not printing.

Values returned by `skip` in `$status`.

Name `sort`
Sort the occurrences in the hitlist for the specified entity.

Synopsis `sort{/e} "entity", "sort_spec_1{, sort_spec_2, ..., sort_spec_n}"`
Each `sort_spec` is:
`field{.entity}{:a(ascending)|:d(descending)}`

Return value None.

Name `sort/list`
Sort the items in a list.

Synopsis `sort/list list[, "D(ascending) U(nique)"]`

Return value The values returned by `sort/list` are shown in the following table:

Value	Meaning
>=0	The number of (remaining) subfields in <code>list</code> .
<0	<code>list</code> could not be located.

Values returned by `sort/list` in `$status`.

Name `spawn`
Pass the specified command to the operating system.

Synopsis `spawn "command"`

Return value The values returned by `spawn` are shown in the following table:

Value	Meaning
0	Success.
<0	Failure.

Values returned by `spawn` in `$status`.

Name `sql`
Pass an SQL statement to the specified DBMS path.

Synopsis `sql "statement" [, "path"]`

Return value The values commonly returned by `sql` in `$status` are shown in the following table:

Value	Meaning
>=0	The number of hits.
-3	Exceptional I/O error (hardware or software). Or, the DBMS reached by <code>path</code> does not support a DML.
-11	Occurrence currently locked.
-16	Network error (unknown).

Values returned by `sql` in `$status`.

Following `sql`, `$result` is set to the value of the first column of the last row (if `statement` contains a 'select').

Name `store`
Activate WRITE, WRITE UP, DELETE or DELETE UP triggers for all occurrences marked as modified.

Synopsis `store(/complete | /truncate)`
`store/e(/complete | /truncate) {"entity"}`

Return value The values commonly returned by `store` in `$status` are shown in the following table:

Value	Meaning
1	No data was stored because no modifications were made to the data since the last <code>retrieve</code> or <code>store</code> statement.
0	Data successfully stored.
-1	Constraint violation. Restricted link violation.
-3	Exceptional I/O error (hardware or software).
-4	Open request for table or file failed. The file or table is not painted or does not exist. A subsequent I/O request to the table or file will return -1.
-5	Update request for non-updatable occurrence.
-6	Exceptional I/O error on write request.
-7	Duplicate key.
-10	Occurrence has been modified or removed since it was retrieved; a reload should be executed.
-11	Occurrence currently locked.
-15	UNIFACE network error.
-16	Network error (unknown).

Values returned by `store` in `$status`.

Name `u_where`
Provide the profile for selection.

Synopsis `u_where (selection_criteria)`

selection_criteria is a logical expression built with the relational and logical operators shown in chapter 13 *Handling data in Proc*. In addition, you can use the following wildcard characters:

Wildcard character	Meaning
*	Match 0- <i>n</i> characters.
?	Match any single character.

Wildcard characters in the `u_where` clause profile.

Return value See `read` or `selectdb`, as appropriate.

Name `uppercase`
Convert a string to uppercase.

Synopsis `uppercase source, destination`

Return value None.

Name `while`
Define a `while/endwhile` loop.

Synopsis `while (expression)`
 `statement`
 `(statements)`
`endwhile`

Or:

`while (expression) statement`

Return value None.

Name `write`
Write the current occurrence to the database.

Synopsis `write`

Return value The values commonly returned by `write` in `$status` are shown in the following table:

Value	Meaning
0	Data was successfully written.
-3	Exceptional I/O error (hardware or software).
-4	Open request for table or file failed. The file or table is not painted or does not exist. A subsequent I/O request to the table or file will return -1.
-5	Update request for non-updatable occurrence.
-6	Exceptional I/O error on write request.
-7	Duplicate key.
-10	Occurrence has been modified or removed since it was retrieved; a reload should be executed.
-11	Occurrence currently locked.
-15	UNIFACE network error.
-16	Network error (unknown).

Values returned by `write` in `$status`.

Chapter 15 Proc functions

This chapter summarizes the functions that are available with the Proc language. See the *UNIFACE functions* chapter in the ► *Proc Language Reference Manual* for complete information.

Name	\$appname Return the name of the application.
Synopsis	\$appname
Return value	The function \$appname returns a string that contains the name of the current start-up shell (in uppercase).

Name	\$batch Return or set the batch mode indicator.
Synopsis	\$batch \$batch = expression
Return value	The function \$batch returns 1 if UNIFACE is in a batch process and 0 if not.

Name \$char
Return the UNIFACE character code for the key that activated some triggers.

Synopsis \$char

Return value The function \$char returns the code for the character or function chosen by the user which activated a trigger. In the <USER KEY> trigger, \$char contains the ^USER_KEY identifier character code. (For more information, see the descriptions of the <USER KEY> triggers in the *Triggers* chapter of the ► *Proc Language Reference Manual*.)

Name \$check
Return or set the checked status of a menu item.

Synopsis \$check
\$check = *expression*



Note: This function can be used with the set and reset commands.

Return value The function \$check returns 1 if the menu item is checked and 0 if it is not checked.

Name \$clock
Return the system time or convert the argument to the Time data type.

Synopsis \$clock {(*source*)}

Return value The function \$clock returns a value that is formatted as HH:MM:SS.

- If *source* is given, \$clock converts *source* into the corresponding time.
- If *source* is omitted, the function returns the system clock time.

Name \$curline
Return the line on which the cursor is positioned in the current field.

Synopsis \$curline

Return value The dp\$curline returns a value that indicates the line on which the cursor is currently positioned.

Name \$curocc
Return the sequence number of the current occurrence in the hitlist.

Synopsis \$curocc {(*entity*)}

Return value The function \$curocc returns the sequence number in the hitlist of the current occurrence. If *entity* does not exist or is not painted on the form, -1 is returned.

The following structure editor functions and statements affect the value of \$curocc:

- ^NEXT_OCC sets \$curocc.
- ^PREV_OCC sets \$curocc.
- retrieve sets \$curocc.
- setocc sets \$curocc.
- ^ADD_OCC modifies \$curocc.
- ^INS_OCC modifies \$curocc.
- ^REM_OCC modifies \$curocc.
- clear resets \$curocc to 1.

Name \$currhits
Return the number of occurrences currently in the hitlist.

Synopsis \$currhits {(*entity*)}

Return value The function \$currhits returns a value which indicates the number of occurrences in the hitlist. If the hitlist has only been partially built, the value is negative. If *entity* does not exist or is not painted on the form, -1 is returned.

Name `$cursorword`
Return the word on which the cursor is positioned in the current field.

Synopsis `$cursorword`

Return value The function `$cursorword` returns the word on which the cursor is currently positioned.

Name `$date`
Return the current date or convert the argument to the Date data type.

Synopsis `$date {(source)}`

Return value The function `$date` returns a value with data type Date.

- If *source* is present, `$date` converts *source* into the corresponding date.
 - If *source* is omitted, the function returns the current system date.
-

Name `$datim`
Return the system date and time or convert the argument to the Datetime data type.

Synopsis `$datim {(source)}`

Return value The function `$datim` returns a value with data type Datetime.

- If *source* is given, `$datim` converts *source* to the corresponding date and time; *source* should be formatted as dd-mmm-yy hh:mm:ss.
 - If *source* is omitted, the function returns the date and time from the system clock. Note that a correct system time value depends on the system clock being correctly set.
-

Name `$dberror`
Return the error code reported by the DBMS.

Synopsis `$dberror`

Return value The function `$dberror` returns a number that is set when the DBMS or network driver encounters an error situation. The value returned is the one given by the DBMS or network to the driver; it is DBMS or network specific.

Name `$dbocc`
Return the sequence number of the current occurrence in the database.

Synopsis `$dbocc {(entity)}`

Return value The values returned by `$dbocc` are shown in the following table:

Value	Description
>0	The sequence number of the current or specified <i>entity</i> in the database.
0	The current occurrence has not been retrieved from the database (it has been entered by the user, and not stored yet).
-1	<i>entity</i> does not exist or is not painted on the form.

Values returned by `$dbocc`.

The following structure editor functions and statements affect the value of `$dbocc`:

- `^NEXT_OCC.`
 - `^PREV_OCC.`
 - `retrieve.`
 - `store.`
 - `clear` (sets `$dbocc` to 0).
-


Name `$direction`
Return the structure editor mode (NEXT or PREVIOUS).

Synopsis `$direction`

Return value The function `$direction` returns 0 if the structure editor is in NEXT mode and 1 if the structure editor is in PREVIOUS mode.

Name `$disable`
Return or set the 'selectable' status of a menu item.

Synopsis `$disable`
`$disable = expression`

 *Note: This function can be used with the `set` and `reset` commands.*

Return value The function `$disable` returns 0 if the menu item is currently not selectable and a non-zero value if the menu item is selectable.

Name `$display`
Return the name of the current display device translation table.

Synopsis `$display`

Return value The value returned by `$display` is the same as the value of the environment variable UDISP. (UDISP defaults to VT100 if it is not set.)
The values returned by default for each GUI are shown in the following table:

GUI	Default value of \$display
Character	USYSTEM
OSF/Motif	X11
OS/2	OS2
Macintosh	MAC
MS-Windows	MSWIN

Default values for \$display.

Name `$empty`
Return the empty frame status for an entity or named area frame.

Synopsis `$empty { (entity | named_area_frame) }`

Return value The values returned by `$empty` are shown in the following table:

Value	Description
2	There are no occurrences of <code>entity</code> or <code>named_area_frame</code> that contain data, and the frame definition of <code>entity</code> or <code>named_area_frame</code> has Suppress on Empty set to 'Y' (Yes).
1	There are no occurrences of <code>entity</code> or <code>named_area_frame</code> that contain data but the frame definition of <code>entity</code> or <code>named_area_frame</code> has Suppress on Empty set to 'N' (No), or has been left blank.
0	The <code>entity</code> or <code>named_area_frame</code> contains at least one occurrence with data.
-1	The <code>entity</code> or <code>named_area_frame</code> does not exist.

Values returned by \$empty.

Name `$entname`
Return the name of the current entity.

Synopsis `$entname`

Return value The function `$entname` returns the name of the current entity (in uppercase); if there is no current entity, `$entname` returns a null value.

Name `$error`
Return the UNIFACE message number for the error.

Synopsis `$error`

Return value The function `$error` returns the message number for the current error. The ► *Messages Manual* shows the default message text for the error message and describes the cause of the error.

Name `$fieldcheck`
Return or set the requirement for field checking.

Synopsis `$fieldcheck(field_name)`
`$fieldcheck(field_name) = expression`



Note: This function can be used with the `set` and `reset` commands.

Return value The function `$fieldcheck` returns 1 if field checking is currently enabled and 0 if not.
In addition, when `$fieldcheck` is used as the target of an assignment:

- `$status` is set to 1, if field checking was successfully enabled.
- `$status` is set to -1, if field checking could not be enabled. This usually means that `field_name` is not present or does not exist; these situations are flagged as a warning at compile time.

Name `$fieldendmod`
Return the modification status of a field when the field is exited.

Synopsis `$fieldendmod`

Return value In the LEAVE FIELD trigger, the function `$fieldendmod` returns 0 if the field has not been modified; it returns 1 if the field has been modified or if `$fieldcheck` has been set for the current field.

The following statements and trigger set `$fieldendmod` to 0:

- EXECUTE.
- clear.
- erase.
- release.
- reload.
- retrieve.
- store.

Name `$fieldmod`
Return the modification status of a field.

Synopsis `$fieldmod({field},{entity})`

Return value The values returned by `$fieldmod` are shown in the following table:

Value	Description
1	Modified.
0	Not modified.
-1	<code>field</code> or <code>entity</code> does not exist (flagged as a warning at compile time).

Values returned by `$fieldmod`.

The following statements and trigger set `$fieldmod` to 0:

- EXECUTE.
- clear.
- erase.
- release.
- reload.
- retrieve.
- store.

Name `$fieldname`
Return the name of the current field.

Synopsis `$fieldname`

Return value The function `$fieldname` returns the name of the current field (in uppercase); if there is no current field, `$fieldname` returns a null value.

Name `$fieldprofile`
Return an indication if a profile character has been entered in the current field.

Synopsis `$fieldprofile (field_name)`

Return value The values returned by `$fieldprofile` are shown in the following table:

Value	Description
1	A profile character has been entered.
0	No profile character has been entered.
-1	<i>field</i> does not exist (flagged as a warning at compile time).

Values returned by `$fieldprofile`.

Name `$fieldproperties`
Return or set the current widget properties of an instance of a field.

Synopsis `$fieldproperties (field)`
`$fieldproperties (field) = "string"`

Return value The function `$fieldproperties` returns the widget properties for the specified *field*.

Name `$fieldvalrep`
Return or set the ValRep for an instance of a field.

Synopsis `$fieldvalrep (field)`
`$fieldvalrep (field) = "string"`

Return value The function `$fieldvalrep` returns the values used by a widget for the specified *field*.

Name `$format`
Return or set data for formatting.

Synopsis `$format`
`$format = "value"`

Return value The function `$format` returns the field data, formatted according to the display (DIS) template of the field.

Name `$formdb`
Return an indication whether data has been retrieved from a database.

Synopsis `$formdb`

Return value The function `$formdb` returns 1 if any entity in the form has been retrieved from a database. The function returns 0 if no entities have been retrieved from a database or if it has been reset to 0 by a Proc statement. The following statements affect the value of `$formdb`:

- `clear` resets `$formdb` to 0.
 - `clear/e` resets `$formdb` to 0 if the only entities retrieved are related to the cleared entity. If unrelated entities in the form have been retrieved from the database, `$formdb` is not reset to 0.
 - `erase` resets `$formdb` to 0.
 - `erase/e` resets `$formdb` to 0 if the only entities retrieved are related to the erased entity. If unrelated entities in the form have been retrieved from the database, `$formdb` is not reset to 0.
 - `release` resets `$formdb` to 0.
-

- **release/e** resets `$formdb` to 0 if the only entities retrieved are related to the released entity. If unrelated entities in the form have been retrieved from the database, `$formdb` is not reset to 0.
- **release/e/mod** resets `$formdb` to 0 if the only entities retrieved are related to the released entity. If unrelated entities in the form have been retrieved from the database, `$formdb` is not reset to 0.
- **release/mod** resets `$formdb` to 0.
- **retrieve** sets `$formdb` to 1. ^RETRIEVE causes the first outermost entity to be retrieved with its related entities. Any unrelated entities are not automatically retrieved. Internally, the entity level flags for database origin are set. This affects the value that `$formdb` becomes when any unrelated entities use Proc statements that modify `$formdb`.
- **retrieve/e** sets `$formdb` to 1. The specified entity is retrieved with its related entities. Any unrelated entities are not automatically retrieved. Internally, the entity level flags for database origin are set. This affects the value `$formdb` becomes when any unrelated entities use Proc statements that modify `$formdb`.
- **store** sets `$formdb` to 1.
- **store/e** sets `$formdb` to 1. Internally, the entity level flags for database origin are set for the entity and related entities stored. This affects the value `$formdb` becomes when any unrelated entities use Proc statements that reset `$formdb`.

Name `$formdbmod`
Return the modification status of database fields in the form.

Synopsis `$formdbmod`

Return value The value of `$formdbmod` is 1 if any fields in the form defined as being part of a database have been modified. If no modifications have been made to database fields, `$formdbmod` returns 0.

The following statements affect the value of `$formdbmod`:

- **clear** resets `$formdbmod` to 0.
- **clear/e** resets `$formdbmod` to 0 if the only database fields modified are in entities related to the cleared entity. If unrelated entities in the form have database fields that have been modified, `$formdbmod` is not reset to 0.
- **erase** resets `$formdbmod` to 0.
- **erase/e** resets `$formdbmod` to 0 if the only database fields modified are in entities related to the erased entity. If unrelated entities in the

form have database fields that have been modified, `$formdbmod` is not reset to 0.

- **release** resets `$formdbmod` to 0.
- **release/e** resets `$formdbmod` to 0 if the only database fields modified are in entities related to the released entity. If unrelated entities in the form have database fields that have been modified, `$formdbmod` is not reset to 0.
- **release/e/mod** sets `$formdbmod` to 1. Internally, the modification status is only set for the specified entity and related entities. Consequently, Proc statements that reset the modification status for unrelated entities do not cause `$formdbmod` to be reset. (Remember `$formdbmod` is evaluated as an inclusive OR for all entities in the form.)
- **release/mod** sets `$formdbmod` to 1.
- **remocc** sets `$formdbmod` to 1 only if the removed occurrence is in the database. If the user has added an occurrence, but not stored it in the database, `$formdbmod` is not altered by **remocc**. The entity level modification flags are set only for the entity, and its related entities.
- **reset** `$formmod` also resets `$formdbmod` to 0. (**set** `$formmod` has no effect on `$formdbmod`.)
- **retrieve** resets `$formdbmod` to 0 if the only database fields modified are in entities related to the retrieved entity. If unrelated entities in the form have database fields that have been modified, `$formdbmod` is not reset to 0. A ^RETRIEVE causes the first outermost entity to be retrieved with its related entities. Any unrelated entities are not automatically retrieved.
- **retrieve/e** resets `$formdbmod` to 0 if the only database fields modified are in entities related to the retrieved entity. If unrelated entities in the form have database fields that have been modified, `$formdbmod` is not reset to 0. Any unrelated entities are not automatically retrieved.
- **store** resets `$formdbmod` to 0.
- **store/e** resets `$formdbmod` to 0 if the only database fields modified are in entities related to the stored entity. If fields in unrelated entities in the form have been modified, `$formdbmod` is not reset to 0.

Name	<code>\$formmod</code> Return the modification status of data in the form.
Synopsis	<code>\$formmod</code> <code>\$formmod = expression</code> <i>Note: This function can be used with the <code>set</code> and <code>reset</code> commands.</i>
Return value	The value of <code>\$formmod</code> is 1 if any field in the form has been modified. If no modifications have been made, <code>\$formmod</code> returns 0. The following statements affect the value of <code>\$formmod</code> : <ul style="list-style-type: none"> • <code>clear</code> resets <code>\$formmod</code> to 0. • <code>clear/e</code> resets <code>\$formmod</code> to 0 if the only fields modified are in entities related to the cleared entity. If unrelated entities in the form have fields that have been modified, <code>\$formmod</code> is not reset to 0. • <code>creocc</code> sets <code>\$formmod</code> to 1. The entity level indicators are only set for the entity and its related entities. • <code>erase</code> resets <code>\$formmod</code> to 0. • <code>erase/e</code> resets <code>\$formmod</code> to 0 if the only fields modified are in entities related to the erased entity. If unrelated entities in the form have fields that have been modified, <code>\$formmod</code> is not reset to 0. • <code>release</code> resets <code>\$formmod</code> to 0. • <code>release/e</code> resets <code>\$formmod</code> to 0 if the only fields modified are in entities related to the released entity. If unrelated entities in the form have fields that have been modified, <code>\$formmod</code> is not reset to 0. • <code>release/e/mod</code> sets <code>\$formmod</code> to 1. • <code>release/mod</code> sets <code>\$formmod</code> to 1. • <code>remocc</code> sets <code>\$formmod</code> to 1. The entity level indicators are only set for the entity and its related entities. • <code>reset</code> resets <code>\$formmod</code> to 0. For consistency, <code>\$formdbmod</code> is also reset. • <code>retrieve</code> resets <code>\$formmod</code> to 0 if the only fields modified are in entities related to the retrieved entity. If unrelated entities in the form have fields that have been modified, <code>\$formmod</code> is not reset to 0. A <code>^RETRIEVE</code> causes the first outermost entity to be retrieved along with its related entities. Any unrelated entities are not automatically retrieved. • <code>retrieve/e</code> resets <code>\$formmod</code> to 0 if the only fields modified are in inner entities related to the retrieved entity or in the retrieved entity itself. If the outer or unrelated entities in the form have fields that have been modified, <code>\$formmod</code> is not reset to 0.

- `set` sets `$formmod` to 1. Unlike `reset`, `set` does not change the value of `$formdbmod`.
- `store` resets `$formmod` to 0.
- `store/e` resets `$formmod` to 0 if the only modified fields are in entities related to the stored entity. If fields in unrelated entities in the form have been modified, `$formmod` is not reset to 0.

Name	<code>\$formname</code> Return the name of the current form.
Synopsis	<code>\$formname</code>
Return value	The function <code>\$formname</code> returns the name of the current form in uppercase. If no form is current, <code>\$formname</code> returns the name of the start-up shell.
Name	<code>\$formtitle</code> Return or set the window title bar of a form.
Synopsis	<code>\$formtitle</code> <code>\$formtitle = "title_string"</code>
Return value	The function <code>\$formtitle</code> returns the title of the form.
Name	<code>\$framedepth</code> Return the depth of the painted frame.
Synopsis	<code>\$framedepth {(frame)}</code>
Return value	The function <code>\$framedepth</code> returns: <ul style="list-style-type: none"> • If <code>frame</code> is given, the number of lines required to print <code>frame</code>. • If <code>frame</code> is omitted, the number of lines used by the current frame.

Name `$gui`
Return the mnemonic for the user interface.

Synopsis `$gui`

Return value The values returned by `$gui` are shown in the following table:

Value	Description
CHR	Character mode
MAC	Macintosh
MSW	MS-Windows
MTF	OSF/Motif
OS2	OS/2

Values returned by `$gui`.

Name `$hide`
Return or set the display status of a menu item.

Synopsis `$hide`
`$hide = expression`



Note: This function can be used with the `set` and `reset` commands.

Return value The function `$hide` returns 1 if the menu item is hidden and 0 if it is displayed.

Name `$hits`
Return the number of occurrences in the hitlist.

Synopsis `$hits(entity)`



Note: From Version 6 the debugger no longer builds the complete hitlist if you examine `$hits`. Instead, it returns the value in `$currhits`.

Return value The function `$hits` returns the total number of occurrences in the hitlist. If `entity` does not exist or is not painted on the form, -1 is returned. The following statements reset the value of `$hits` to 0:

- `clear.`
- `release.`

Name `$ioprint`
Return or set the message level in the message frame.

Synopsis `$ioprint`
`$ioprint = value`

Return value The value returned by `$ioprint` is the sum of the codes for the messages currently selected; see chapter 18 *I/O messages*. If 0 is returned, no messages appear in the message frame.

Name **\$keyboard**
Return or set the current keyboard translation table.

Synopsis **\$keyboard**
\$keyboard = "table"

Return value The value returned by **\$keyboard** is the keyboard table currently in use. The values returned by default for each GUI are shown in the following table:

GUI	Default value of \$keyboard
Character	USYSTEM
OSF/Motif	X11
OS/2	OS2
Macintosh	MAC
MS-Windows	MSWIN

Default values of \$keyboard.

Name **\$language**
Return or set the current language code.

Synopsis **\$language**
\$language = "code"

Return value The function **\$language** returns the country code currently in use.

Name **\$lines**
Return the number of lines remaining on the current page.

Synopsis **\$lines**

Return value When UNIFACE is printing (**\$printing** is 1), **\$lines** returns the number of lines remaining on the page, *excluding* the header and trailer frames.

When UNIFACE is not printing (**\$printing** is 0), the value of **\$lines** is 0 and **\$status** is set to -1.

Name **\$next**
Return the value of the next occurrence of a field.

Synopsis **\$next (field)**

Return value The function **\$next** returns the value of *field* in the next occurrence. A null value is returned when there is no next occurrence.

Name **\$number**
Return the value of the numeric part of a string.

Synopsis **\$number ("string")**

Return value The function **\$number** returns the value of the leading numeric part it encounters in *string*. If *string* contains no numeric text, or if it starts with alphabetic text, **\$number** returns 0.

Name `$occcheck`
Return or set the modification status of an occurrence.

Synopsis `$occcheck(entity)`
`$occcheck(entity) = expression`



*Note: This function can be used with the **set** and **reset** commands.*

Return value The function `$occcheck` returns 1 if occurrence checking is currently enabled and 0 if not.
In addition, when `$occcheck` is used as the target of an assignment:

- `$status` is set to 1, if occurrence checking was successfully enabled.
- `$status` is set to -1, if occurrence checking could not be enabled. This usually means that *entity* is not present or does not exist; these situations are flagged as a warning at compile time.

Name `$occdel`
Return the removal status of an occurrence.

Synopsis `$occdel(entity)`

Return value The values returned by `$occdel` are shown in the following table:

Value	Description
1	Occurrence is marked for removal.
0	Occurrence is not marked for removal.
-1	<i>entity</i> does not exist or is not painted on the form.

Values returned by `$occdel`.

The following statement and trigger set `$occdel` to 1:

- `erase.`
- `<REMOVE OCCURRENCE>`.

Name `$occddepth`
Return the depth of the painted occurrence.

Synopsis `$occddepth`

Return value The function `$occddepth` returns the number of lines an occurrence requires to be painted on the screen.

Name `$occmmod`
Return the modification status of an occurrence.

Synopsis `$occmmod(entity)`

Return value The values returned by `$occmmod` are shown in the following table:

Value	Description
1	Modified.
0	Not modified.
-1	<i>entity</i> does not exist or is not painted on the form.

Values returned by `$occmmod`.

The following statements and trigger set the value of `$occmmod` to 0:

- `EXECUTE.`
- `store.`
- `release.`
- `retrieve.`
- `clear.`
- `reload.`

Name **\$oprsys**
Return a mnemonic for the client operating system used by UNIFACE.

Synopsis **\$oprsys**

Return value The function **\$oprsys** returns a mnemonic that identifies the client operating system. The values returned are shown in the following table:

Value	Description
A	ALPHA/VMS
D	MS-DOS
I	Macintosh
L	INTEL NT
M	MPE/x
O	OS/2
S	STRATUS VOS
U	UNIX
V	OpenVMS VAX
W	MS-Windows
X	ALPHA NT

Values returned by **\$oprsys**.

Name **\$page**
Return the current page number.

Synopsis **\$page**

Return value The function **\$page** returns the page number of the page currently being printed. If UNIFACE is not printing, **\$page** returns 0.

Name **\$password**
Return the password used to log on to the path.

Synopsis **\$password (path)**

Return value The function **\$password** returns the password used to log on to the DBMS given by *path*. If no password was required to log on to the DBMS, a null string ("") is returned.

Name **\$previous**
Return the value of the field in the previous occurrence.

Synopsis **\$previous (field)**

Return value The function **\$previous** returns the value of the previous occurrence of *field*. A null value is returned when there is no previous occurrence.

Name **\$printing**
Return a status indicating whether the current form is printing.

Synopsis **\$printing**

Return value The function **\$printing** returns 1 if UNIFACE is printing, and 0 if UNIFACE is not printing.

Name **\$prompt**
Return or set the position for the cursor when the current Proc module ends.

Synopsis **\$prompt**
\$prompt = field(.entity)

Return value The function **\$prompt** returns a string that contains the name of the field where the cursor will be positioned when control returns to the structure editor.

Name **\$properties**
Return or set the current widget properties of a field.

Synopsis **\$properties (field)**
\$properties (field) = "string"

Return value The function **\$properties** returns the widget properties for the specified *field*.

Name **\$putmess**
Return the contents of the message frame.

Synopsis **\$putmess**

Return value The function **\$putmess** returns the contents of the message frame. If the message frame has been cleared then **\$putmess** returns nothing.

Name **\$relation**
Return the related key field.

Synopsis **\$relation({field[.entity]})**

Return value The function **\$relation** returns the name of the related field.

Name **\$result**
Return the result of certain Proc statements.

Synopsis **\$result**
\$result = expression

Return value The function **\$result** is set by many Proc statements. Refer to the documentation for the individual Proc statements in chapter 14 *Proc statements* for the values of **\$result**.

Name **\$rettype**
Return the retrieval mode of the outermost entity.

Synopsis **\$rettype**

Return value The function **\$rettype** returns a value only in the READ or <ADD/INSERT OCCURRENCE> trigger. The values returned are shown in the following table. (Note that some values are returned only in the <ADD/INSERT OCCURRENCE> trigger.)

Value	Returned in <ADD/INSERT OCCURRENCE>?	Returned in READ ?	Description
65	Y	N	Add occurrence.
73	Y	N	Insert occurrence.
78	Y	Y	Next occurrence.
82	Y	Y	Retrieve.
110	Y	Y	Retrieve sequential.

Values returned by **\$rettype**.

The following structure editor functions and statement set **\$rettype**:

- ^ADD_OCC.
- ^INS_OCC.
- read.

Name **\$selblk**
Return or set the contents of the select buffer.

Synopsis **\$selblk**
\$selblk = field

Return value The function **\$selblk** returns the current contents of the structure editor select buffer.

Name **\$status**
Return the current condition code.

Synopsis **\$status**

Return value The function **\$status** is always an integer value. If a decimal value is assigned to **\$status**, UNIFACE rounds it to the nearest integer. In general:

- A negative value in **\$status** indicates an error.
- A positive value indicates a warning or information.
- 0 indicates a successful operation.

Name **\$storetype**
Return the type of update for the current occurrence.

Synopsis **\$storetype(entity)**

Return value The values returned by **\$storetype** are shown in the following table:

Value	Description
1	The occurrence will be inserted in the database.
0	The occurrence will be updated in the database.
-1	<i>entity</i> does not exist or is not painted on the form.

Values returned by **\$storetype**.

The following structure editor function and statements affect the value of **\$storetype**:

- **store** and **retrieve** set **\$storetype** to 0.
 - **release/mod** and **^ADD_OCC** set **\$storetype** to 1.
-

Name **\$syntax**
Convert a string to a syntax string.

Synopsis **\$syntax(string)**

Return value The function **\$syntax** returns a syntax string. (See chapter 13 *Handling data in Proc* for more information about syntax strings.)

Name **\$text**
Return the text of a message or help text.

Synopsis **\$text(idstring)**

Return value The function **\$text** returns the text associated with the message *idstring*.

Name **\$time**
Return the system time.

Synopsis **\$time**

Return value The function **\$time** returns the system time, accurate to one second.

Name **\$totdbocc**
Return the number of occurrences of the entity that have been retrieved from a database.

Synopsis **\$totdbocc(entity)**

Return value The function **\$totdbocc** returns the total number of occurrences of *entity* currently fetched from the database. If *entity* does not exist or is not painted on the form, -1 is returned.

The following structure editor functions and statements affect the value of `$totdbocc`:

- `^NEXT_OCC` sets `$totdbocc` to the number retrieved.
- `^PREV_OCC` sets `$totdbocc` to the number retrieved.
- `retrieve` sets `totdbocc` to the number retrieved.
- `store` sets `$totdbocc` to the number stored.
- `clear` sets `$totdbocc` to 0.

Name `$totlines`
Return the total number of lines available on the page for printing.

Synopsis `$totlines`

Return value If UNIFACE is printing (that is, `$printing` is 1), `$totlines` returns the total number of lines on the page available for printing. It does not include any lines required to print header or trailer frames.
If UNIFACE is not printing (that is, `$printing` is 0), the value of `$totlines` is set to 0 and `$status` is set to -1.

Name `$totocc`
Return the number of occurrences of an entity in the form.

Synopsis `$totocc(entity)`

Return value The function `$totocc` returns the number of occurrences in the form. If `entity` does not exist or is not painted on the form, -1 is returned. When a form is empty, `$totocc` always returns 1; this reflects the empty occurrence in the form.

The following structure editor functions and statements affect the value of `$totocc`:

- `^NEXT_OCC` sets `$totocc` to the total number of occurrences of an entity.
- `^PREV_OCC` sets `$totocc` to the total number of occurrences of an entity.
- `read` sets `$totocc` to the total number of occurrences of an entity.
- `retrieve` sets `$totocc` to the total number of occurrences of an entity.
- `^ADD_OCC` increments `$totocc` by 1.

- `^INS_OCC` increments `$totocc` by 1.
- `^REM_OCC` decrements `$totocc` by 1.
- `clear` sets `$totocc` to 1.

Name `$user`
Return the user name used to log on to the path.

Synopsis `$user(path)`

Return value The function `$user` returns the current user name.

Name `$valrep`
Return or set the ValRep used by a widget for a field.

Synopsis `$valrep(field)`
`$valrep(field) = "string"`

Return value The function `$valrep` returns the values used by a widget for the specified `field`.

Name `$variation`
Return or set the variation code of the library.

Synopsis `$variation`
`$variation = "string"`

Return value The function `$variation` returns the name of the current library.

Chapter 16 Debugger commands

The following table summarizes the debugger commands, showing examples of each. The commands are explained in greater detail in the ► *Proc Language Reference Manual*. The shortest form that is recognized for each command is shown in capital letters (for example, **S**hOW indicates that you can type **sh**, **sho** or **show**).

Command	Example	Meaning
<i>Process control:</i>		
Step	step	Execute one Proc statement.
	s 5	Execute five Proc statements.
Line	l 10	Execute ten Proc statements, treating any called modules as a unit.
Nop	nop	Skip the current Proc statement without executing it.
Go	g	Leave debug mode until a breakpoint or next debug instruction.
DONe	done	Exit the Proc module.
QUIT	quit	Exit the application.
<i>Process tracing:</i>		
XTRACE	xtrace	Copy all executed Proc statements to the message frame. Example output: {DTLF} Y32:1 run "bookings" {EXEC} Y1:2 retrieve {READ} Y27:1 read {READ} Y27:2 done<end of module> {EXEC} Y1:3 edit booknr.booking.travel ...
	xtrace off	Stop copying executed Proc statements to the message frame.

Summary of debugger commands.

part 1 of 3

Command	Example	Meaning
FILTer	<code>filt pdis</code> <code>filt off</code> <code>filt</code>	Turn off debugging in the PREDISPLAY trigger. Debug in all triggers. Display the name of the currently 'filtered' trigger.
Breakpoints:		
Break	<code>break y2 4</code> <code>b gp_store 15</code> <code>break</code>	Set a breakpoint at line 4 of Proc module <code>y2</code> . Set a breakpoint at line 15 of Proc module <code>gp_store</code> . Clear all breakpoints.
CALL	<code>call on</code> <code>call off</code>	Set a breakpoint at each <code>call</code> statement. Clear breakpoints set by <code>call on</code> .
RETurn	<code>return on</code> <code>return off</code>	Set a breakpoint at each <code>return</code> and <code>done</code> statement. Clear breakpoints set by <code>return on</code> .
SHow	<code>show</code>	Display the current breakpoint settings, for example: <code>break on: censtore:15 call on</code>
Examine data:		
Examine	<code>ex \$1</code> <code>\$saveit\$</code> <code>e</code> <code>e \$fieldendmod</code> <code>\$entname</code> <code>ex cusnr.customer</code> <code>e city</code> <code>\$result = 10</code> <code>\$\$name = "Uniface"</code>	Display the contents of <code>\$1</code> . Display the contents of local variable <code>\$saveit\$</code> . Display the contents of the next general variable. Display the value of <code>\$fieldendmod</code> . Display the value of <code>\$entname</code> . Display the contents of <code>cusnr.customer</code> . Display the contents of <code>city</code> . Assign the value 10 to <code>\$result</code> . Assign the string to global variable <code>\$\$name</code> .
Miscellaneous:		
DUMP	<code>dump</code> <code>dump gp_store</code> <code>dump/all</code>	Dump the Proc statements of the current module. Dump the Proc statements of the Proc module <code>gp_store</code> . Dump all Proc modules of the current form.
MESsage	<code>mes 1024</code>	Display the text of message 1024 in the message frame.
PUTmess	<code>putmess off</code> <code>putmess on</code>	Send the message frame input to the screen. Send the message frame input to the message frame.
CLRmess	<code>clrmess off</code> <code>clrmess on</code>	Do not clear the message frame. Clear messages from the message frame normally.

Summary of debugger commands.

part 2 of 3

Command	Example	Meaning
IOprint	<code>io 63</code>	Send I/O messages with codes totalling 63 ($63 = 1 + 2 + 4 + 8 + 16 + 32$) to the message frame. (See chapter 18 <i>I/O messages</i> .)
HELP	<code>help</code>	Show help information for the debugger.
FRAME	<code>frame</code>	Display the message frame.
RECall	<code>rec</code>	Redisplay the next Proc statement.

Summary of debugger commands.

part 3 of 3

Chapter 17 Trigger mnemonics

When the debugger displays the location of the next statement, it includes an abbreviated name for the trigger containing that statement. The abbreviations used are shown below. The cross-reference Repository Reports also use these abbreviations.

Abbreviation	Trigger	Level
ACPT	<ACCEPT>	Form
AIO	<ADD/INSERT OCCURRENCE>	Entity
APPL	APPLICATION EXECUTE	Application
ASYN	ASYNCHRONOUS INTERRUPT	Application
ASYS	ASYNCHRONOUS INTERRUPT	Form
CLR	<CLEAR>	Form
DECR	DECRYPT	Field
DELE	DELETE	Entity
DFMT	DEFORMAT	Field
DLUP	DELETE UP	Entity
DTLE	<DETAIL>	Entity
DTLF	<DETAIL>	Field
ENCR	ENCRYPT	Field
ERAS	<ERASE>	Form
ERRE	ON ERROR	Entity
ERRF	ON ERROR	Field
EXEC	EXECUTE	Form
FGF	FIELD GETS FOCUS	Field
FMT	FORMAT	Field
HLPE	<HELP>	Entity
HLPF	<HELP>	Field

Trigger name abbreviations used by the debugger.

part 1 of 2

Abbreviation	Trigger	Level
LFLD	LEAVE FIELD	Field
LMK	LEAVE MODIFIED KEY	Entity
LMO	LEAVE MODIFIED OCCURRENCE	Entity
LOCK	LOCK	Entity
LPO	LEAVE PRINTED OCCURRENCE	Entity
MNUA	<MENU>	Application
MNUF	<MENU>	Entity
MNUF	<MENU>	Field
MNUS	<MENU>	Form
NFLD	<NEXT FIELD>	Field
OGF	FRAME GETS FOCUS	Header, trailer, break frame
OGF	OCCURRENCE GETS FOCUS	Entity
OPTN	OPTION	Menu item
PDIS	PREDISPLAY	Menu item
PFLD	<PREVIOUS FIELD>	Field
PRNT	<PRINT>	Form
PULA	<PULLDOWN>	Application
PULS	<PULLDOWN>	Form
QUIT	<QUIT>	Form
READ	READ	Entity
RETR	<RETRIEVE>	Form
RETS	<RETRIEVE SEQUENTIAL>	Form
RMO	<REMOVE OCCURRENCE>	Entity
SMOD	START MODIFICATION	Field
STOR	<STORE>	Form
SWIT	<SWITCH KEYBOARD>	Application
UKYA	<USER KEY>	Application
UKYS	<USER KEY>	Form
VALC	VALUE CHANGED	Field
WRIT	WRITE	Entity
WRUP	WRITE UP	Entity

Trigger name abbreviations used by the debugger.

part 2 of 2

Chapter 18 I/O messages

This chapter summarizes information that appears in the *Command line switches* chapter in the ► *UNIFACE Reference Manual*.

You can determine the I/O messages that appear in the message frame at the following levels:

1. With the debug statement `ioprint`. (See the *Debugging Procs* chapter in the ► *Proc Language Reference Manual*.)
2. With the `$ioprint` function in Proc. (See the *Proc functions* chapter in the ► *Proc Language Reference Manual*.)
3. With the `/pri` command line switch. (See the *Command line switches* chapter in the ► *UNIFACE Reference Manual*.)
4. With Objects→I/O Messages on the Define Start-up Shell form. (See the *Applications* chapter in the ► *Developers' Guide*.)

For all but the last of these, you indicate the desired messages by providing a value that is the sum of the corresponding codes from the following table:

Code	Description
1	Store sequence messages.
2	Driver function calls.
4	Return values from Fetch and Select driver function calls.
8	Description block from Open driver function call.
16	Descriptions from <code>where</code> and <code>order by</code> clauses.
32	Generated SQL (if available).
64	All system calls UNIFACE sends to operating system.
128	All calls to UOBJ plus the data sent.

Codes for selecting I/O messages in the message frame.



The following sections describe the messages generated by each code that can be selected.

Note: The exact format of each message is determined by the DBMS driver in use, but the information is similar.

Code 1—Store sequence messages

These messages show whether the entity is being updated or inserted (Store) or deleted (Delete), or whether the named referential integrity constraint is being applied to a 'many' entity.

Type	Message
<i>'One' entity:</i>	
Store	Store: <i>table/filename</i> dbocc: <i>X</i> occ: <i>Y</i>
Delete	Delete: <i>table/filename</i> dbocc: <i>X</i> occ: <i>Y</i>
<i>Referential integrity constraints for 'many' entity:</i>	
NULLIFY	Nulwid: <i>many_table</i> dbocc: <i>X</i> , occ: <i>Y</i>
CASCADING	Delwid: <i>many_table</i> dbocc: <i>X</i> , occ: <i>Y</i>
RESTRICTED	Readal: <i>many_table</i> dbocc: <i>X</i> , occ: <i>Y</i>

Formats of Store sequence messages.

- The number following dbocc shows the sequence number of the occurrence as it was retrieved from the database. If dbocc is 0, the occurrence was inserted.
- The number following occ shows the sequence number of the occurrence in the form.

Code 2—Driver function calls

These messages provide different types of information depending on the driver function call requested. The information provided by each function call is shown below. See the ► *DBMS Specific Guide* and the ► *DBMS Driver Cookbook* for further information about the actions of the driver function calls.

Driver function call	Message and definition
Logoff	A, pseudo: X on driver: dbms <i>X</i> is a driver code for the DBMS. <i>dbms</i> is the DBMS driver in use.
Commit	C, pseudo: X on driver: dbms <i>dbms</i> is the DBMS driver in use.
Delete	D, mode: 0, on file/table: table/filename
Fetch	F, mode: M, on file/table: table/filename index: X S <i>M</i> is 0 = Fetch record or row. 1 = Fetch and lock record or row. 2 = Position record or row and lock. No data fetched; used for update and delete of overflow segments. <i>X</i> is index access number ('0' physical address: rowid or RFA). <i>S</i> is index selection operator: LIKE, <, <=, =, >=, or >.
Logon	L, pseudo: X on driver: dbms <i>X</i> is a driver code for the DBMS. <i>dbms</i> is the DBMS driver in use.
Open	O, mode: M, on file/table: table/filename <i>M</i> is 0 = Open the file or table and do not create. 1 = Open the file or table and create if it does not exist. 2 = Special call for the Create Table utility. 4 = Generate SQL to create referential integrity constraints. 5 = Generate SQL to drop referential integrity constraints. 6 = Generate SQL to check for referential integrity violations.
SQL	Q, mode: M, pseudo: X on driver: dbms <i>M</i> is 0 = Send an SQL statement to the DBMS. If the statement selects data, return the last row of the result. 2 = Return the first row of selected data. 3 = Return a subsequent row. <i>X</i> is a driver code for the DBMS. <i>dbms</i> is the DBMS driver in use.

Formats of driver function call messages (code 2).

part 1 of 2

Driver function call	Message and definition
Rollback	R, pseudo: X on driver: dbms X is a driver code for the DBMS. dbms is the DBMS driver in use.
Select	S, mode: M, on file/table: table/filename index: X S M is 0 = Select records or rows and create hitlist. 1 = Select records or rows and return number of hits, but do not add to hitlist. 2 = Restricted look up function for existence check. 3 = Select new records or rows and append to the hitlist. 4 = Used for selectdb. X is preferred index access number ('0': rowid or RFA). S is preferred index selection operator: LIKE, <, <=, =, =>, or >.
Update existing occurrence	U, mode: 0, on file/table: table/filename length: L L is length of I/O buffer.
Insert new occurrence	W, mode: 0, on file/table: table/filename length: L L is length of I/O buffer.
Close	Z, mode: 0, on file/table: table/filename
Wildcard delete/nullify	*, mode: M, on file/table: table/filename index: X S M is 0 = Cascading delete. 1 = Nullify delete. X is index access number ('0' physical address: rowid or RFA). S is index selection operator: LIKE, <, <=, =, =>, or >.

Formats of driver function call messages (code 2).

part 2 of 2

Code 4—Return values from Fetch and Select driver function calls

These messages show either the number of hits selected, or the length of the I/O buffer fetched. If there is enough room, this information is included on the same line as the driver function call messages; otherwise, it appears on another line.

Code 8—Description block from Open driver function call

This shows the values used by the driver when opening the file or table. This information is used primarily for debugging DBMS drivers. See the ► *DBMS Driver Cookbook* for more information.

Code 16—Descriptions from where and order by clauses

These show which **where** and **order by** clauses are generated by UNIFACE. The information is formatted as follows:

where -> field: *fieldname* = value **order by** -> *fieldname* ascending/descending

Each **where** clause appears on a separate line. The clauses are logically joined as 'AND' constructions.

Code 32—Generated SQL

These messages show the SQL instructions generated by UNIFACE to execute I/O. This information is only available with SQL-based DBMSs. See the ► *DBMS Specific Guide* module for your DBMS to see whether this is supported.

Code 64—All system calls UNIFACE sends to operating system

These messages show all the system calls, or 'subprocesses' which UNIFACE starts at the operating system level. These are all the spools, spawns and so on. An example of the use of this message level is to track what happens to print files, because the spawned command appears in the message frame.

UNIFACE also reports any file which cannot be opened at operating system level.

Code 128—All calls to UOBJ plus the data sent

These messages record all the calls to UOBJ and all the data returned. This quickly builds a very large message frame.

Code 2, 8 and 16—Location of object

When any of these codes is included in the message level, an extra message line occurs that identifies the target object and shows where it was located:

Object: type = code, lang = language, lib = library, id = name; location

The codes which indicate the object type are shown in the following table:

Code	Object type
P	Global Proc or global variable.
T	Keyboard translation table.
D	Device translation table.
C	Panel.
M	Message, help text or language setup.
U	Menu bar or menu item.
I	Glyph.

Object type codes.

Note: Beginning in V6.1.e, this message occurs only when code 128 is selected in the message level.



Chapter 19 Widgets

This chapter summarizes the information about widget categories and standard widgets. For more details of defining and using:

- Generic standard widgets, see the ► *Developers' Guide*.
- GUI-specific standard widgets—including details of valid values for each widget property—see the appropriate module or modules in the ► *Environment Specific Guide*.
- Custom widgets, see the ► *Widget Cookbook*.

Important points

Each widget has a physical name and a logical name. The names of all physical widgets supplied with UNIFACE begin with the letter 'U', so avoid giving widgets logical names beginning with 'U'.

Unifields and Form Text are functionally equivalent to—but not the same as—widgets. Therefore, they are not included in this chapter. For details of unifields and Form Text, see the ► *Developers' Guide*.

Categorizing widgets

The following table summarizes the main categories of widget:

Category	Description
Physical	Either a widget supplied with UNIFACE or a custom widget (that is, an independently built physical widget created in 3GL, following the instructions in the ► <i>Widget Cookbook</i>). Physical widgets are the building blocks from which all logical widgets are derived; you cannot modify them or paint them on forms. Each physical widget has a unique physical name, by which it is referenced.
Logical	Any widget derived from a physical widget, which you can associate with fields and paint on UNIFACE forms. Each logical widget has a unique logical name, by which it is referenced. Logical widgets supplied with UNIFACE are referred to as standard widgets.

Widget categories supported by UNIFACE.

Properties

The following sections summarize the properties and other useful information for all standard widgets, in alphabetical order by widget type. Tables show the properties supported in each environment, their physical names, and whether you can change those properties dynamically (that is, using Proc code). In the tables, each bullet (*) means that the corresponding property is supported or that it can be changed dynamically.

Type Check box

Physical name UCHECKBOX

Mnemonic CHK

Properties

Property	Name	Platform		MSW	OS2	MTF	MAC	CHR
		Supported	Dynamic	Supported	Dynamic	Supported	Dynamic	Supported
3-D Effect	3D	*				*	*	
Decoration	Decoration							*
Frame	Frame							
Label Font	LabelFont	*	*	*	*	*	*	*
Tri-state	TriState	*	*	*	*	*	*	*
Use Detail Trigger	V52	*	*	*	*	*	*	*
Widget Font	Font	*						

Properties of standard check boxes.

Table notes:

* The 3-D Effect property in Motif is supported by the X resource `shadowThickness`. Therefore, to set the 3-D Effect property in Motif you must use the name of the physical widget.

Triggers

Check boxes can activate the following triggers:

Trigger	Platform	MSW	OS2	MTF	MAC	CHR
<DETAIL> (See Notes)		*	*	*	*	*
VALUE CHANGED		*	*	*	*	*

Triggers that can be activated by standard check boxes.

Notes

The <DETAIL> trigger can be activated by a standard check box only if Use Detail Trigger is checked *on* on the Define Form Field Properties form.

Check box

UNIFACE V6.1

Code all actions associated with a Boolean (two-state) check box in the appropriate <DETAIL> trigger.

Code all actions associated with a Tri-state check box in the appropriate VALUE CHANGED trigger.

Use Proc code to test for the following values, and take the appropriate action:

Value	Description
0	Unchecked, meaning False.
1	Checked, meaning True.
""	Tri-state only: Null, meaning Undetermined.

Valid check box values.

UNIFACE V6.1

Command button

Type Command button

Physical name UCMDBUTTON

Mnemonic CMD

Properties

Property	Name	Platform		MSW	OS2	MTF	MAC	CHR
		Supported	Dynamic	Supported	Dynamic	Supported	Dynamic	Supported
Auto Label	AutoLabel	•	•	•	•		•	•
Decoration	Decoration							•
Horizontal Align	HAlign	•	•	•	•	•	•	
Horizontal Scale	HScale	•	•	•	•		•	•
Label Font	LabelFont	•	•	•	•		•	•
Preserve Aspect	PreserveAspect	•	•	•	•		•	•
Role	Role	•	•	•	•		•	•
Tool Tip	ToolTip	•	•	•	•			
Vertical Align	VAlign	•	•	•	•		•	•
Vertical Scale	VScale	•	•	•	•		•	•
Widget Font	Font	•	•	•	•	•	•	•

Properties of standard command buttons.

Triggers

Command buttons can activate the following trigger:

Trigger	Platform	MSW	OS2	MTF	MAC	CHR
<DETAIL>		•	•	•	•	•

The trigger that can be activated by standard command buttons.

Notes

For information about loading images into command buttons, see chapter 20 *Loading images*.

Type Drag and drop

Physical name UDRAGDROP

Mnemonic Not applicable.

Properties

Property	Name	Platform		MSW	OS2	MTF	MAC	CHR
		Supported	Dynamic	Supported	Dynamic	Supported	Dynamic	Supported
3-D Effect	3D	•	•					
Frame	Frame	•	•					
Dynamic Label	Label	•	•					
Label Font	LabelFont	•	•					
Source	Source	•	•					
Source Feedback	SourceFeedBack	•	•					
Target	Target	•	•					
Target Feedback	TargetFeedBack	•	•					
Target Highlight	TargetHILite	•	•					

Properties of standard drag and drop widgets.

Triggers

Drag and drop widgets can activate the following triggers:

Trigger	Platform	MSW	OS2	MTF	MAC	CHR
<DETAIL>		•				
VALUE CHANGED		•				

Triggers that can be activated by standard drag and drop widgets.

Notes

To drag an icon, you must select the icon; you cannot initiate a drag by selecting the label associated with the icon.

Type Drop-down list

Physical name UDROPDOWNLIST

Mnemonic DRP

Properties

Property	Name	Platform		MSW	OS2	MTF	MAC	CHR
		Supported	Dynamic	Supported	Dynamic	Supported	Dynamic	Supported
3-D Effect	3D	•				•	•	
Dynamic	Dynamic	•						
Force Fit	ForceFit	•	•	•				
Frame	Frame					•	•	
Label Font	LabelFont	•		•	•		•	•
Number of Choices	Entries	•	•	•		•	**	
Sort Alphabetically	Sort	•		•	•	•	•	•
Widget Font	Font	•	•	•	•	•	•	•

Properties of standard drop-down lists.

Table notes:

- * The 3-D Effect property in Motif is supported by the X resource `shadowThickness`. Therefore, to set the 3-D Effect property in Motif you must use the name of the physical widget.
- ** To set the Number of Choices you must use the ValRep list for the drop-down list.

Triggers

Drop-down lists can activate the following triggers:

Trigger	Platform	MSW	OS2	MTF	MAC	CHR
<DETAIL>						•
VALUE CHANGED *		•	•	•	•	

Triggers that can be activated by standard drop-down lists.

Table notes:

- * The VALUE CHANGED trigger is activated even if the value has not changed; use Proc to test the value and take the appropriate action.

Type Edit box

Physical name UEDITBOX

Mnemonic EDT

Properties

Property	Name	Platform		MSW		OS2		MTF		MAC		CHR	
		Supported	Dynamic	Supported	Dynamic	Supported	Dynamic	Supported	Dynamic	Supported	Dynamic	Supported	Dynamic
3-D Effect	3D	*						*	*				
Attach to Window Border	Attach	*	*	*	*	*	*	*	*	*	*	*	*
Auto Select	AutoSelect	*	*	*	*	*	*	*	*	*	*	*	*
Automatic Word Wrap	WordWrap	*		*		*		*		*		*	
Double Click	DbiCik	*	*	*	*	*	*	*	*	*	*	*	*
Dynamic	Dynamic	*	*										
Frame	Frame	*		*		*		*		*		*	
Horizontal Scroll Bar	HScroll	*		*		*		*		*		*	**
Label Font	LabelFont	*	*	*	*					*	*	*	*
Multiline	MultiLine	*		*		*		*		*		*	
No Display Character	NoDisplay	*	*	*	*	*	*	*	*	*	*	*	*
Separate Workspace	Huge	*											
Vertical Scroll Bar	VScroll	*		*		*		*		*		*	
Widget Font	Font	*	*	*	*	*	*	*	*	*	*	*	*

Properties of standard edit boxes.

Table notes:

* The 3-D Effect property in Motif is supported by the X resource `shadowThickness`. Therefore, to set the 3-D Effect property in Motif you must use the name of the physical widget.

** In character mode, edit boxes are not supported but are mapped to unifielfds.

Triggers

Edit boxes can activate the following triggers:

Trigger	Platform	MSW	OS2	MTF	MAC	CHR
START MODIFICATION		•	•	•	•	•
VALUE CHANGED		•	•	•	•	•

Triggers that can be activated by standard edit boxes.

Table notes:

* In character mode, edit boxes are not supported but are mapped to unifielts.

Type	Label
Physical name	ULABEL
Mnemonic	Not applicable.
Properties	Labels have no widget properties and, therefore, no exclusive definition forms. To change the fonts used in your labels, see the Notes on this page.
Triggers	Labels cannot activate triggers.
Notes	<p>Labels are not true widgets because you cannot give them logical names and you can paint them on forms without associating them with fields.</p> <p>In character mode, labels are not supported but are mapped to background text.</p> <p>To change the font for a label that is associated with a widget, use the Define Properties form for the widget (select the widget and choose MENU->Field->Widget Properties).</p> <p>To change the font for a label that is not associated with a widget, select the label and choose MENU->Label->Label Font.</p>

Type List box

Physical name ULISTBOX

Mnemonic LST

Properties

Property	Name	Platform		MSW	OS2	MTF	MAC	CHR
		Supported	Dynamic	Supported	Dynamic	Supported	Dynamic	Supported
3-D Effect	3D	•				•	•	
Attach to Window Border	Attach	•	•	•	•		•	•
Force Fit	ForceFit	•		•			•	
Frame	Frame	•		•		•	•	•
Label Font	LabelFont	•	•	•	•		•	•
Multi Select	MultiSelect					•		
Sort Alphabetically	Sort	•		•		•	•	•
Widget Font	Font	•	•	•	•	•	•	•

Properties of standard list boxes.

Table notes:

* The 3-D Effect property in Motif is supported by the X resource `shadowThickness`. Therefore, to set the 3-D Effect property in Motif you must use the name of the physical widget.

List box

UNIFACE V6.1

Triggers

List boxes can activate the following triggers:

Trigger \ Platform	MSW	OS2	MTF	MAC	CHR
<DETAIL>					•
VALUE CHANGED *	•	•	•	•	

Triggers that can be activated by standard list boxes.

Table notes:

* The VALUE CHANGED trigger is activated even if the value has not changed; use Proc to test the value and take the appropriate action.

UNIFACE V6.1

Picture

Type

Picture

Physical name

UPICTURE

Mnemonic

PIC

Properties

Property	Name	Platform		MSW	OS2	MTF	MAC	CHR
		Supported	Dynamic	Supported	Dynamic	Supported	Dynamic	Supported
3-D Effect	3D	•	•					
Attach to Window Border	Attach	•	•	•	•		•	•
Frame	Frame	•	•	•	•		•	•
Horizontal Alignment	HAlign	•	•	•	•	•	•	•
Horizontal Image Scaling	HScale	•	•	•	•		•	•
Horizontal Scroll Bar	HScroll	•	•	•	•		•	•
Label Font	LabelFont	•	•	•	•	•	•	•
Monochrome Bitmap	Monochrome	•	•				•	•
Preserve Aspect	PreserveAspect	•	•	•	•		•	•
Switch Palette	SwitchPalette	•	•					
Vertical Alignment	VAlign	•	•	•	•		•	•
Vertical Image Scaling	VScale	•	•	•	•		•	•
Vertical Scroll Bar	VScroll	•	•	•	•		•	•

Properties of standard picture widgets.

Table notes:

* In character mode, picture names are not supported but are mapped to text fields.

Triggers

Picture widgets can activate the following triggers:

Trigger \ Platform	MSW	OS2	MTF	MAC	CHR
<DETAIL>
VALUE CHANGED

Triggers that can be activated by standard picture widgets.

Table notes:

* In character mode, picture widgets are not supported but are mapped to text fields.

Notes

For information about loading images, see chapter 20 *Loading images*.

Type

Radio group

Physical name

URADIOGROUP

Mnemonic

RAD

Properties

Property	Name	Platform		MSW	OS2	MTF	MAC	CHR
		Supported	Dynamic	Supported	Dynamic	Supported	Dynamic	Supported
3-D Effect	3D
Alignment	Align
Decoration	Decoration
Frame	Frame
Label Font	LabelFont
Number of Columns	Columns
Number of Rows	Rows
Widget Font	Font

Properties of standard radio groups.

Table notes:

* The 3-D Effect property in Motif is supported by the X resource `shadowThickness`. Therefore, to set the 3-D Effect property in Motif you must use the name of the physical widget.

Triggers

Radio groups can activate the following triggers:

Trigger \ Platform	MSW	OS2	MTF	MAC	CHR
<DETAIL>
VALUE CHANGED

Triggers that can be activated by standard radio groups.

Type Spin button

Physical name USPINBUTTON

Mnemonic SPN

Properties

Property	Name	Platform		MSW	OS2	MTF	MAC	CHR
		Supported	Dynamic	Supported	Dynamic	Supported	Dynamic	Supported
3-D Effect	3D	•				•	•	
Acceleration	Accel	•	•	•				
Auto Select	AutoSelect	•	•	•	•		•	•
Delay	Delay	•	•					
Double Click	DbiClk	•	•				•	•
Frame	Frame						•	•
Increment Value	Increment	•	•	•	•	•	•	•
Label Font	LabelFont	•	•	•	•	•	•	•
Orientation	Orientation	•	•					
Repeat	Repeat	•	•					
Widget Font	Font	•	•	•	•	•	•	•
Cyclic	Cyclic						•	•

Properties of standard spin buttons.

Table notes:

* The 3-D Effect property in Motif is supported by the X resource `shadowThickness`. Therefore, to set the 3-D Effect property in Motif you must use the name of the physical widget.

** In character mode, spin buttons are not supported but are mapped to text fields.

Spin button

UNIFACE V6.1

Triggers

Spin buttons can activate the following trigger:

Trigger \ Platform	MSW	OS2	MTF	MAC	CHR
VALUE CHANGED	•	•	•	•	•

The trigger that can be activated by standard spin buttons.

Table notes:

* In character mode, spin buttons are not supported but are mapped to text fields.

UNIFACE V6.1

Visual Basic Extension (VBX)

Type Visual Basic Extension (VBX)

Physical name vvbxx

Mnemonic vbxx

Properties

Property \ Name	Platform		MSW	OS2	MTF	MAC	CHR	
	Supported	Dynamic	Supported	Dynamic	Supported	Dynamic	Supported	Dynamic
Accept Event(s)	AcceptEvent	•	•					
Change Event(s)	ChangeEvent	•	•					
Ignore Event(s)	IgnoreEvent	•	•					
Value Property	ValueProp	•	•					
VBX Name	Vbx	•	•					

Properties of standard VBX widgets.

Triggers

VBX widgets can activate the following triggers:

Trigger \ Platform	MSW	OS2	MTF	MAC	CHR
<DETAIL>	•				
VALUE CHANGED	•				

Triggers that can be activated by standard VBX widgets.

Notes

VBX widgets are not dependent on Visual Basic.

Chapter 20 Loading images

This chapter summarizes the information about loading images, which is included in the ► *Developers' Guide*.

Identifying images using data types

UNIFACE provides the following data types for image fields:

Data type	Description
I	Generic image data type. If the field is a BLOB field in the database, no special steps are needed to load the data. Otherwise the image is identified by the field data.
!#	Image is a BLOB in the database.
!^	Image is a glyph in UOBJ.
!@	Image is stored in a file.

Image data types supplied with UNIFACE.

Identifying images using field data

If a field has the generic image data type I, you must specify whether the image source is a file, a glyph in UOBJ or uobj.dol, or a BLOB in the database. Use the following notation:

Notation	Meaning
^ <i>Glyph</i>	Load the glyph named <i>Glyph</i> from UOBJ or uobj.dol.
@ <i>FileName</i>	Load the image from the file <i>FileName</i> .
# <i>FieldName</i>	Load the image from the BLOB field <i>FieldName</i> .

Identifying images using field data.

Image loaders

UNIFACE provides loaders for the following image formats:

Format	Description
BAF	MS-Windows or OS/2 Array file.
BMP	MS-Windows or OS/2 Bitmap file.
DIB	MS-Windows or OS/2 Dibmap file.
GEM	Digital Research (Novell).
GIF	CompuServe Inc.
GL	Grasp GL image.
ICO	MS-Windows or OS/2 Icon.
IFF	Electronic Arts (IFF/LBM).
MAC	Macintosh MacPaint Monochrome.
PBM	Poskanzer Portable.
PCX	Zsoft.
PIC	Pictor/PcPaint.
PICT	QuickDraw Pict (Macintosh).
SGI	Silicon Graphics.
SUN	Sun Raster.
TGA	Truevision Targa.
TIF	Aldus/Microsoft Tiff.
UNI	UNIFACE internal storage format (used with glyphs).
WMF	MS-Windows MetaFile.
XBM	X-Windows Bitmap (monochrome).
XPM	X-Windows Picture Bitmap (color).
XWD	X-Windows Dump.

Image loader formats supported by UNIFACE.

Chapter 21 Search order for global objects

This chapter summarizes the way UNIFACE searches for global objects at run time. For more information about global objects and how UNIFACE searches for them, see the *Global objects and libraries* chapter in the ► *Designers' Guide*.

Locating global objects at run time

The following table shows the order in which UNIFACE searches libraries for global objects at run time. A '1' indicates the first library searched, a '2' the second and so on.

Library	Object						
	Global Procs	Global variables	Menus and menu bars	Messages	Glyphs	Glyphs	Help texts
Form library	1	3	3				
Application library	2	2					
SYSTEM_LIBRARY	3	1					
\$VARIATION			1	1	1	1	1
USYS			2	2	2	2	2

The order in which UNIFACE searches for global objects at run time.



Note: When a menu is called from another menu, UNIFACE looks in the library in which the parent menu was found.

Language variations

Panels, messages, help texts, menus and menu bars can all have language variations. When UNIFACE searches for these global objects at run time it searches first through all the libraries in the search path for objects with the same language as that specified in `$language`. If it cannot find the object in that language variation, it then repeats the search for objects with language variation USA.

UNIFACE V6.1

1. **Function keys and Super keys**
2. **Profile characters**
3. **Naming objects**
4. **Video attributes**
5. **Command line switches**
6. **Assignment files**
7. **Assignment settings**
8. **Models of the Repository**
9. **Structure editor functions**
10. **Interface definitions**
11. **Syntax definitions**
12. **Layout definitions**
13. **Handling data in Proc**
14. **Proc statements**
15. **Proc functions**
16. **Debugger commands**
17. **Trigger mnemonics**
18. **I/O messages**
19. **Widgets**
20. **Loading images**
21. **Search order for global objects**



unifAce