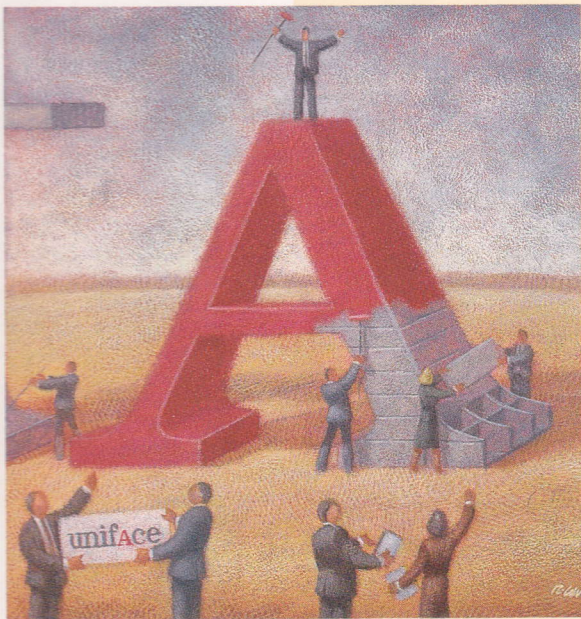


Quick Reference Guide



uniface
Advanced Software Technology

Quick Reference Guide

UNIFACE V5.2

101075201

Revision 1

21 September 1992

QR

UNIFACE V5.2

Quick Reference Guide

Revision 1

© Copyright Uniface B.V., Amsterdam, The Netherlands.

Rights to the contents of this document

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

Trademarks

UNIFACE™, PolyServer™ and Universal Presentation Interface™ are trademarks of Uniface B.V. This document contains many references to third party hardware and software products and manufacturers; these are all registered trademarks.

Authors

This document was written by Jim Gabriel and Jim Crew. It was compiled by Anna Hayward with the invaluable assistance of Leda Baker.

Publishing equipment

This document was produced with FrameMaker publishing software on Data General AViiON machines.

Reactions

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

Uniface B.V.
Technical Publications
Hogehilweg 16
1101 CD Amsterdam
The Netherlands

Tel. +31 (0)20 6976644
Fax. +31 (0)20 6912912

Table of Contents

1	Proc statements.....	1-1
1.1	Which Procs in which triggers.....	1-32
1.1.1	Rules for areas of restricted usage.....	1-32
2	Special functions.....	2-1
3	Extracting values from the data	3-1
3.1	Strings.....	3-1
3.1.1	Extracting values from strings	3-1
3.1.2	Rules for string extraction	3-2
3.1.3	Converting to strings.....	3-3
3.2	Date and time	3-4
3.2.1	Information about date and time	3-4
3.3	Units of measurement for use in Procs	3-5
3.3.1	Codes for date and time arithmetic	3-5
3.3.2	Examples of how UNIFACE treats date and time values.....	3-6
3.3.3	Limit values.....	3-6
3.3.4	Normalization of time and date values.....	3-7
3.3.5	Extracting values from date and time data	3-9

3.3.6	Week numbering	3-11	4.2.18	nop	4-7
3.3.7	Converting to a date or time value.....	3-11	4.2.19	putmess off	4-7
3.3.8	Converting to dates	3-11	4.2.20	putmess on.....	4-7
3.3.9	Converting to times.....	3-13	4.2.21	quit.....	4-8
3.3.10	Converting to a date and time.....	3-14	4.2.22	return on.....	4-8
3.3.11	Converting to time from a number.....	3-14	4.2.23	return off.....	4-8
3.4	Numbers	3-15	4.2.24	show, sh	4-8
3.4.1	Extracting values from numeric data	3-15	4.2.25	trace on, tr on.....	4-9
3.4.2	Rounding.....	3-15	4.2.26	trace off, tr off	4-9
3.4.3	Converting to a number	3-16	4.2.27	trace 1, tr 1.....	4-9
4	Debugging Procs.....	4-1	4.2.28	trace 0, tr 0.....	4-9
4.1	Command line.....	4-1	4.2.29	step, s.....	4-9
4.2	Commands	4-3	4.2.30	step {number}, s {number}	4-9
4.2.1	break {module} {line}, b {module} {line}.....	4-4	4.2.31	xtrace	4-10
4.2.2	break, b.....	4-4	4.2.32	Examining contents of \$registers.....	4-10
4.2.3	call on.....	4-4	4.3	Trigger mnemonics.....	4-10
4.2.4	call off.....	4-4	5	Naming conventions, reserved words and wildcards.....	5-1
4.2.5	clmess off.....	4-4	5.1	Naming conventions.....	5-1
4.2.6	clmess on	4-4	5.1.1	External schema, conceptual schema, application, field	5-1
4.2.7	done	4-5	5.1.2	Frames on paint tableau.....	5-2
4.2.8	dump.....	4-5	5.1.3	Global models.....	5-2
4.2.9	dump module	4-5	5.2	Reserved words	5-2
4.2.10	examine {number}, ex {number}.....	4-5	5.2.1	IDF application dictionary names.....	5-3
4.2.11	examine, ex	4-5	5.2.2	UNIFACE Reporter application dictionary names.....	5-3
4.2.12	ex {field_name}{entity_name}	4-6	5.3	Wildcards.....	5-4
4.2.13	ex number = {value}	4-6	6	Interface definition.....	6-1
4.2.14	ex {\$function(name)}	4-6	6.1	Data types (UNIFACE).....	6-1
4.2.15	go, g	4-6	6.2	UNIFACE packing codes	6-2
4.2.16	ioprint {number}, io {number}	4-6			
4.2.17	line {number}, l {number}	4-7			

6.3	Allowed combinations.....	6-4	10.4	Hewlett Packard HP-HIL.....	10-5
6.4	Variable length techniques.....	6-5	10.5	IBM PC AT 83/84 key.....	10-6
7	Syntax checks.....	7-1	10.6	IBM AT 101/102 key enhanced.....	10-7
7.1	Entry format.....	7-1	10.7	IBM RS6000 console.....	10-8
7.1.1	Entry format examples.....	7-2	10.8	IBM 6150 RT PC console.....	10-9
7.2	Display/Edit/Prompt.....	7-3	10.9	OS/2.....	10-10
7.3	Characters allowed in a field.....	7-4	10.10	SCO-Unix.....	10-11
7.4	Shorthand codes for Field syntax model.....	7-4	10.11	SCO-Xenix.....	10-12
8	Display format.....	8-1	10.12	Siemens 97801.....	10-13
8.1	String.....	8-1	10.13	Stratus v102.....	10-14
8.2	Numeric (and float).....	8-2	10.14	Sun-3.....	10-15
8.3	Date.....	8-4	10.15	Sun-4 SPARC station.....	10-16
8.4	Time.....	8-6	10.16	Function key combinations (all keyboards).....	10-17
8.5	Combined date and time.....	8-7	10.17	'Super' key combinations.....	10-18
8.6	Shorthand codes for Field layout model.....	8-7	11	IDF command switches.....	11-1
9	Video and color.....	9-1	11.1	Switches.....	11-2
9.1	Video attributes.....	9-1	11.2	Sub-switches.....	11-3
9.2	Color definition.....	9-2	12	Assignments.....	12-1
10	Keyboard layouts.....	10-1	12.1	Priorities.....	12-1
10.1	Bull TWS 2103.....	10-2	12.2	Syntax.....	12-2
10.2	Data General FKB4700.....	10-3	12.3	Entity assignments.....	12-3
10.3	DEC VT100/200.....	10-4	12.4	Path assignments.....	12-4
			12.4.1	Path to DBMS or network driver.....	12-4
			12.4.2	Path to path.....	12-4
			12.5	Wildcard assignments.....	12-5
			12.6	UNIFACE system settings and options.....	12-7

Name	askmess - display a message and wait for the user response.
Synopsis	askmess[/no beep] "message" {,"reply_1", ..., "reply_n" {,terminator_1, terminator_2}} or askmess terminator_1, terminator_2
Return Value	\$status is set to the value returned by the askmess statement. This is: 0 Is returned in \$status for 'N' (if no replies given). 1 For 'Y' or 'J' (if no replies given). reply number If replies are given. The reply entered by the user is indicated in \$status; the first reply as 1, the second reply as 2, etc. <0 If the user uses terminator_2 to end reply, instead of terminator_1.

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 4GL Proc module.

Synopsis call entry_name

Return Value The \$status register is set to the value returned by the called module. If no value is returned, or there is no return statement in the module, 0 is returned.

Name clear - clear the data (entered by the user) in the external schema or named entity.

Synopsis clear{/e "entity"}{source}

Return Value \$status is set to 0 if the data was successfully cleared. An error is returned by the DBMS driver if the driver could not clear the data. The following common errors can be returned:

-3 Hardware or software error.
-16 Network error.

Name	close - close the database specified, or all databases.
Synopsis	close { <i>\$path</i> }
Return Value	\$status is set to 0 if the DBMS or all DBMSs were successfully closed. An error is returned by the DBMS driver if the driver could not close a DBMS. The following common errors can be returned:
	-3 Hardware or software error.
	-16 Network error.

Name	clrmess - clear the message frame of text.
Synopsis	clrmess
Return Value	None.

Name	commit - commit a transaction to the database.
Synopsis	commit { <i>dbms</i> <i>\$path</i> }
Return Value	\$status is set to 0 for success, a negative value indicates the DBMS driver returned an error code. The following codes are commonly returned by DBMS drivers:
	-3 Hardware or software error.

-16 Network error.

Name	compare - compare fields of two adjacent occurrences.
Synopsis	compare {/next /previous} (<i>field1</i> { <i>field2</i> ,... <i>fieldn</i> }) from " <i>entity</i> "
Return Value	The compare statement sets both \$status and \$result. The following values may be returned in \$status:
	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 register, and the field or register does not contain the correct entity name (or one that does not exist). In this situation, \$result is always 0.
	The result of the comparison is stored in \$result. The possible values are:
	1 Perfect match of all specified fields.
	0 Fields do not match. This value is always returned if \$status is -1.
	-1 No previous or next occurrence (error situation).

Name	compute - evaluate an expression.
Synopsis	{compute} <i>destination</i> {/init} = <i>expression</i> <i>constant</i>
Return Value	None.

Name creocc - create an empty occurrence of the specified entity.

Synopsis creocc "entity", sequence_number

- If *sequence_number* is less than 0, an occurrence is added (appended) after the last occurrence in the external structure.
- If *sequence_number* equals 0, an empty occurrence of *entity* is created, using the current sequence number. The new occurrence is inserted before the old active occurrence, so the effect is to increase all subsequent occurrence sequence numbers by 1.
- If *sequence_number* is greater than the current number of occurrences plus one of *entity*, \$status is set to -1 and no occurrence is created.

Return Value \$status is set by the creocc statement. It can be one of the following two values:

sequence_number Of the created occurrence.
-1 If an occurrence could not be created.

Name debug - start the interactive debugger.

Synopsis debug

Return Value None.

Name delete - delete an occurrence from the database.

Synopsis delete

Return Value

\$status is set by this statement:

1 If the form is being prototyped.
0 Function completed successfully.
-3 Hardware or software error.
-5 If the user is not allowed to modify this occurrence, and message 2004 - No modifications allowed on occurrence of this entity is displayed.
-6 Exceptional I/O error on write request.
-11 Occurrence currently locked.
-16 Network error.

Name

display - present the external schema on the screen as read-only (cannot be modified).

Synopsis

display{/menu} {field}

Return Value

\$status is set by this statement:

0 On success.
-1 If the form specified could not be found, and the message 0113 - Form paint is empty; cannot edit, display, or print is displayed.
-1 If the field does not exist, and message 0114 - Failed to start edit on field *field* is displayed.
-16 If the application is running in batch mode, and the message 0016 - Terminal input aborted; not allowed in batch mode is displayed. Use a test on \$batch to avoid this.

Name done - exit from a Proc.

Synopsis done

Return Value \$status remains unchanged.

Name edit - display the external schema and start the structure editor for user input.

Synopsis edit[*/menu /nowander*] (*field*)

Return Value \$status is set to the following values:

- 0 On success.
- 1 If the edit statement is not in an EXECUTE trigger, and the message 0164 - Edit instruction only allowed in EXECUTE trigger is displayed. This value is also returned if there are no prompting fields on the form (they are all defined as no prompt fields).
- 16 If an edit is attempted when in batch mode. Use a test on \$batch to avoid this.

Name eject - eject a page during printing.

Synopsis eject

Return Value \$status is set to the following values:

- 1 If the external schema is not being printed when you issue the eject statement (that is, \$printing is 0), \$status is set and no further action is taken for this statement.
- 0 Any other situation.

Name else - execute statements when the if condition is not satisfied.

Synopsis else {*Proc_statement*}
endif
or
else
Proc_statements
endif

Return Value None.

Name end - mark the ending of a Proc.

Synopsis end

Return Value \$status remains unchanged.

Name endif - mark the end of an if/else block.

Synopsis endif

Return Value None.

Name endwhile - mark the end of a while loop.

Synopsis endwhile

Return Value None.

Name entry - label the start of a 4GL Proc module.

Synopsis entry *entry_name*

Return Value None.

Name erase - activate entity level DELETE or DELETE UP trigger for all occurrences in the external schema.

Synopsis erase(/e "*entity*")

Return Value \$status is set by the erase statement:

- | | |
|-----|---|
| 1 | Erase is not allowed (for example, the external schema was activated with run/query). |
| 0 | For success. |
| -2 | Occurrence not found. |
| -3 | Hardware or software error. |
| -5 | Update request for an occurrence that cannot be updated. |
| -6 | Exceptional I/O error on write request. |
| -11 | Occurrence currently locked. |
| -16 | Network error. |

Name exit - immediately exit the current external schema and return to the previous or specified external schema.

Synopsis exit {{{*expression*}}}, "*external_schema*"})

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

Name `field_syntax` - dynamically set the syntax attributes for a field.

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

Return Value None.

Name `field_video` - set the video attributes of the specified field for the current occurrence.

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

Return Value None.

Name `file_dump` - write the contents of the specified field to the specified file.

Synopsis `file_dump{/append} field, "(path)/file"`

Return Value None.

Name `file_load` - read the contents of the specified file into the specified field.

Synopsis `file_load "(path)/file", destination`

Return Value `Sstatus` is set by the `file_load` statement. Possible error situations are:

-1	The file cannot be opened.
-3	Exceptional I/O error.
-11	File locked.
-16	Network error.

Name `goto` - unconditional branch 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 response.

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

Return Value `Sstatus` is set by the `help` statement.

- 1 With ^ACCEPT.
 0 With ^QUIT.
 -1 If the help file USYS:USYSTXT could not be found.
 -1 If the field does not exist.
 -2 If the help file contains information from a different version (message 0019 - Form *formname* has wrong version; you must recompile it is displayed), or cannot be interpreted (the file is not a help file, message 0020 - File *formname* not recognized as application or form is displayed).

Name if - start of an if/else block.

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

Return Value None.

Name /init - initialize a field without changing the status of *soccmmod*, *\$formmod* or *\$fieldmod*.

Synopsis *field/init = value*

Return Value None.

Name length - return the number of characters in the specified text field.

Synopsis length *string*

Return Value \$result is set to the number of characters in the string.

Name lock - lock the occurrence in the database.

Synopsis lock

Return Value \$status is set by the lock statement.

- 1 Is always returned when the form is being prototyped.
 0 Is returned when the occurrence cannot be modified (for example, during a run/query).
 -1 Is returned if there is no active occurrence.
 -2 Is returned if the occurrence has been removed since it was retrieved.
 -3 Is returned if the hit for the occurrence does not exist.
 -5 Is returned when there is no hit for the occurrence and message 2008 - Occurrence cannot be modified due to fetch error is displayed.
 -5 Is also returned when the occurrence is read-only (cannot be locked), and message 2004 - No modifications allowed on occurrence of this entity is displayed.
 -10 Is returned to indicate the occurrence has been modified or removed since it was retrieved, and a reload should be executed.
 -11 Is returned if the occurrence is already locked.

Other DBMS driver error codes may be returned in certain circumstances; refer to the *Specific DBMS Information Manual*.

Name lookup - find the number of occurrences that match the profile.

Synopsis lookup

Return Value The number of hits that match the profile is returned in `$$status`. If an error occurs, the following values can be returned:

-3 Exceptional I/O error.
-16 DBMS network error.

Name macro - define a structure editor keystroke macro.

Synopsis macro{`/exit`} "*character_sequence*"

Return Value `$$status` is always set to 0.

Name message - write the string to the screen.

Synopsis message{/no beep} "*string*"

Return Value None.

Name nodebug - end interactive debugging.

Synopsis nodebug

Return Value None.

Name numgen - generate a unique number using the specified counter as a base.

Synopsis numgen "*counter*", *increment* {, "*library*"}

Return Value This statement returns a negative value for failure, the value coming from the DBMS driver (used to access the counter) in most cases. `$$result` is set to the new number if the function is successful.

Name	numset - set the specified counter to a new value.
Synopsis	numset "counter", init_value {, "library"}
Return Value	\$status is set to 0 on success, -1 otherwise.
Name	open - open a database for access.
Synopsis	open "parameters", "path"
Return Value	If the open operation fails for any reason, \$status is set to a negative number. This is usually the driver return code. Possible values include: 0 Function completed successfully. -3 Hardware or software error. -4 Open request for table failed (most common error). -16 Network error.
Name	perform - call the specified 3GL function.
Synopsis	perform(/noterm) "function"
Return Value	\$status is set to the value returned by <i>function</i> , or -1 if <i>function</i> could not be found. Consequently, do not return -1 in a 3GL function, as this is indistinguishable from UNIFACE not being able to find <i>function</i> .

\$status may contain garbage if the 3GL function does not return a value.

Name	print - activate printing, optionally using a print model.
Synopsis	print(/ask) {"printer_model"} {,"print_option"}
Return Value	\$status is set by the print statement. 0 On success. -1 Printing is already being performed (\$printing is 1). -1 ^QUIT was used in the Print Attribute form. -1 An invalid <i>print_option</i> was used (not one of A, C, F or S). -1 UNIFACE could not print. 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	\$status is set by the print_break statement. -1 When not printing or inside a header or footer. 0 If the Proc code in the OCCURRENCE BECOMES ACTIVE trigger for the break frame returns a negative value. 1 If the Proc code in the OCCURRENCE BECOMES ACTIVE trigger returns a positive value.

Name pull-down - activate or load the specified pull-down menu into the application pull-down menu area.

Synopsis pull-down{/load} {"menu_bar_name"}

Return Value \$status is set by the pull-down and pull-down/load statements:

-1 If the pull-down menu does not exist.
 0 If the OPTION trigger of the selected pull-down menu item is empty.
 Otherwise \$status is set to the value returned by the Proc code in the OPTION trigger of the selected pull-down menu item.

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 a record from the hitlist.

Synopsis read{/lock} {fu_where (expression1)} | {where "expression2"} %\n {order by "field {desc} {,..."}

Return Value \$status is set to the value returned by the DBMS driver. This is 0 for success, and a negative value for failure. The following values can be returned:

0 Success.
 -2 Occurrence not found.
 -3 Hardware or software error.
 -16 Network error.

Name refresh - redraw the screen.

Synopsis refresh

Return Value None.

Name release - release the database controls and clear the message frame.

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

Return Value If *entity* does not exist, message 0145 - Entity *entity* not available is displayed, but \$status is not set. If *entity* does exist, the usual set of DBMS driver codes are returned. These include:

0 Function completed successfully.
 -3 Hardware or software error.
 -16 Network error.

Name	reload - reread and lock the current occurrence from the database.
Synopsis	reload
Return Value	If the occurrence exists, the usual set of DBMS driver error codes can be returned to \$status. These include:
0	Function completed successfully.
-2	Occurrence not found.
-3	Hardware or software error.
-11	Occurrence currently locked.
-16	Network error.

Name	remocc - mark an occurrence of the specified entity for deletion on the next store.
Synopsis	remocc "entity", sequence_number
Return Value	\$status is set by the remocc statement. It can be set to one of the following two values:
sequence_number	Of the removed occurrence.
-1	If the occurrence could not be removed.

Name	repeat - mark the start of a repeat/until block.
Synopsis	repeat statement (statements) until (expression)

Return Value None.

Name reset - reset the specified \$function to 0.

Synopsis reset \$function

Return Value \$status is set to -1 if the function cannot be modified by reset. If the function can be modified by reset, \$status is set to the new value of the function (0). The only modifiable functions are \$formmod, \$fieldcheck and \$occheck.

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

Synopsis retrieve{/e {"entity"}|{/o {"entity"}} {"wildcard_character"}

Return Value \$status is set by the retrieve statement. Common values returned include:

- 10 No data to retrieve.
- 4 The occurrence was found in the external schema. 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 Success.
- 1 Unexpected 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.
- 4 Open request for the file or table failed.
- 7 The key exists in the database and was not found in the hitlist (duplicate key). This is also returned by `retrieve/o` when the entity is painted as a normal down entity, and multiple hits were found during the database lookup (ambiguous 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). This is also returned by `retrieve/o` when the entity is painted as a foreign entity and multiple hits were found during the database lookup.
- 16 DBMS network error.

Name return - exit from the Proc module, optionally returning a value.

Synopsis return { (*expression*) }

Return Value \$status is set to the value of *expression*, if one is given. If no expression is given, \$status is set to 0.

Name rollback - back out of the transaction (if supported by DBMS).

Synopsis rollback {*dbms*} {*\$path*}

Return Value \$status is set to the value returned by the DBMS driver. This is 0 for success, and a negative value for failure. The following values are returned:

- | | |
|-----|----------------------------------|
| 0 | Function completed successfully. |
| -3 | Hardware or software error. |
| -16 | Network error. |

Name run - activate the specified external schema.

Synopsis run [/display] { /query } "schema" { {,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 run external schema) if it contains a return or exit statement. The default (that is, if no return or exit statements are present) is one of the following values:

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

Name scan - inspect the field or register, returning the starting position of the text that matches the specified profile.

Synopsis scan *string*, '*profile*'
or
scan *string*, '*profile*'

Return Value The position of the first character of the string is returned in \$result:

\$result > 0	Starting position of the match.
\$result = 0	Profile not found.
\$result = 0	Source is a null string.

Name selectdb - calculate the aggregate values for specified fields in the database.

Synopsis selectdb (*(function (field), ... , function (field))*) from "*entity*"
{*u_where clause*} to *destinations*

where *function* is one of ave, count, max, min, sum.

Return Value \$status is set by the selectdb statement.

>=0	The number of occurrences that matched <i>clause</i> .
-1	If a field does not exist.
-1	Function cannot be used with this type of field.
-3	Hardware or software error.
-16	Network error.

Name set - set the specified \$function to 1.

Synopsis set *\$function*

Return Value \$status is set to -1 if the function cannot be modified by set. If the function can be modified by set, \$status is set to the new value of the function (1). The only modifiable functions are \$formmod, \$fieldcheck and \$occcheck.

Name setocc - make a specific occurrence the current occurrence.

Synopsis setocc "*entity*", *sequence_number*

Return Value \$status is set by the setocc statement. One of the following values are returned:

<i>sequence_number</i>	Of the new occurrence.
-1	If the occurrence could not be set to.
-3	No more occurrences to set to.

Name	skip - line feed the specified number of lines when printing.
Synopsis	skip (<i>expression</i>)
Return Value	\$status is set to -1 if UNIFACE is not printing. \$status is set to 0 on success, or if the statement is ignored.

Name	spawn - execute the specified command, using the operating system.
Synopsis	spawn " <i>command</i> "
Return Value	\$status is -1 if a null command (" ") has been given as an argument. Otherwise, spawn returns either an operating system code or the <i>command</i> return code.

Name	sql - pass a SQL statement to the specified DBMS.
Synopsis	sql " <i>statement</i> ", " <i>path</i> "
Return Value	\$status is set to the number of hits, \$result is set to the value of the first column of the last row (if the statement contains a select). A negative value indicates a DBMS driver error code. Common values include: -3 Hardware or software error, or the DBMS given by <i>path</i> does not support a DML.

-11	Occurrence currently locked.
-16	Network error.

Name	store - activate WRITE, WRITE UP, DELETE or DELETE UP triggers for all occurrences marked as modified.
-------------	--

Synopsis	store/e (" <i>entity</i> ")
-----------------	-----------------------------

Return Value	The usual set of DBMS driver error codes can be returned. These include:
1	No store performed because no modifications were made to the data since the last retrieve or store statement.
0	Function completed successfully.
-3	Hardware or software error.
-4	Open request for the file or table failed.
-5	Update request for an occurrence that cannot be updated.
-6	Exceptional I/O error on write request.
-7	Duplicate key.
-10	Record modified (perform a reload).
-11	Occurrence currently locked.
-15	UNIFACE network error.
-16	DBMS network error.

Name until - mark the end of a repeat/until block.

Synopsis until *expression*

Return Value None.

Name u_where - provide the profile for selection.

Synopsis u_where (*clause*)

Return Value See read or selectdb, as appropriate.

Name where - DBMS-specific profile clause for the read statement.

Synopsis where "*specific_clause*"

Return Value See read and \$berror.

Name while - mark the start of a while/endwhile block.

Synopsis while (*expression*)

```

statement
(statement
...
...
statement)
endwhile
or
while (expression) statement

```

Return Value None.

Name write - write the current occurrence to the database.

Synopsis write

Return Value The usual set of DBMS driver error codes can be returned. These include:

0	Function completed successfully.
-3	Hardware or software error.
-4	Open request for the file or table failed.
-5	Update request for an occurrence that cannot be updated.
-6	Exceptional I/O error on write request.
-7	Duplicate key.
-10	Record modified (perform a reload).
-11	Occurrence currently locked.
-15	UNIFACE network error.
-16	DBMS network error.

1.1 Which Procs in which triggers

In figure 1-1 you see which Procs are allowed in which triggers. The following symbols are used:

Symbol	Meaning
✓	This trigger is the best place for the statement.
-	You should not use the statement in the trigger.
?	You can use the statement in the trigger, but there is usually a better place for it.
*	You can use the statement in the trigger, but correct usage depends heavily on what you want to do with it; caution is recommended.

table 1-1 Meaning of symbols used in figure 1-1.

1.1.1 Rules for areas of restricted usage

Rule 1

Use only when the following condition holds:

- Proc operates on occurrence of inner entity.

Rule 2

Use only when one of the following conditions holds:

- Proc operates on occurrence of inner entity.
- Proc operates on current occurrence of current entity.

Rule 3

Use only when the following condition holds:

- Proc operates on field in current occurrence of current entity.

Rule 4

Use only when one of the following conditions holds:

- Proc operates on field in current occurrence of current entity.
- Proc operates on field in occurrence of inner entity.

Procs and their classification	ES start		ES database requests			ES general	Occurrence database requests	Occurrence manipulation			Occurrence database I/O	Field general				
	edit	display	retrieve	store	erase	release clear	retrieve/o	setocc	creocc	remocc	read	write	delete	lock	reload	(assignments) file_dump file_load
APPL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MNUA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UKYA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SWIT	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
EXEC	✓	✓	*	*	*	*	-	*	*	*	-	-	-	-	-	✓
ACPT	-	-	?	*	?	-	-	-	✓	*	-	-	-	-	-	✓
QUIT	-	-	-	?	-	-	-	-	-	-	-	-	-	-	-	✓
RETR	-	-	✓	-	-	?	?	-	✓	*	-	-	-	-	-	✓
STOR	-	-	-	✓	-	*	-	-	✓	*	-	-	-	-	-	✓
ERAS	-	-	?	-	✓	-	-	-	✓	?	?	-	-	-	-	✓
CLR	-	-	-	?	-	✓	-	-	*	*	-	-	-	-	-	✓
MNUS	-	-	-	-	-	✓	-	-	*	*	-	-	-	-	-	✓
UKYS	-	-	?	?	?	?	?	-	?	?	?	-	-	-	-	✓
OBA	-	-	-	-	-	-	-	-	1	1	1	-	-	-	-	✓
LMK	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	✓
LMO	-	-	-	-	-	-	-	-	1	1	1	-	-	-	-	✓
AIO	-	-	-	-	-	-	-	-	2	2	?	-	-	-	-	✓
RMO	-	-	-	-	-	-	-	-	?	?	?	-	-	-	-	✓
DTLE	-	-	?	?	?	-	*	-	1	1	1	-	-	-	-	✓
HLPE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓
MNUE	-	-	-	-	-	-	-	-	1	1	1	-	-	-	-	✓
READ	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	3
WRIT	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	3
DELE	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	3
LOCK	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	3
WRUP	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	3
DLUP	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	3
SMOD	-	-	-	-	-	-	-	-	1	1	1	-	-	-	-	4
LFLD	-	-	-	-	-	-	-	-	1	1	1	-	-	-	-	4
NFLD	-	-	-	-	-	-	-	-	-	1	1	-	-	-	-	✓
PFLD	-	-	-	-	-	-	-	-	-	1	1	-	-	-	-	✓
DTLF	-	-	?	?	?	-	*	-	1	1	1	-	-	-	-	4
HLPF	-	-	-	-	-	-	-	-	1	1	1	-	-	-	-	4
MNUF	-	-	-	-	-	-	-	-	1	1	1	-	-	-	-	4

figure 1-1 Usage of Proc statements in triggers.

Chapter 2 Special functions

Name `$applname` - return the name of the application.

Synopsis `{register = | field_name =} $applname`

Return Value The `$applname` function returns the name of the current application (in uppercase). The *register* should be defined as a string or special string register, unless it is a \$register, in which case the type conversion will be done automatically.

Name `$batch` - batch mode indicator.

Synopsis `{field = | register =} $batch`

Return Value `$batch` returns the following values:

0	UNIFACE is not in a batch process.
1	UNIFACE is in a batch process.

Name \$char - return the UNIFACE character code for the key that activated the <USER KEY> or START MODIFICATION trigger.

Synopsis *{field = | register =} \$char*

Return Value The code for the character chosen by the user, which activated a START MODIFICATION or <USER KEY> trigger.

Name \$clock - return the system time or convert the argument to the time data type.

Synopsis *{field = | register =} \$clock {source}*

Return Value The value returned is formatted as HH:MM:SS. If *source* is omitted, the function returns the system clock time. Be aware that a correct system time value depends on the system clock for the machine being correctly set. If *source* is given, \$clock converts the *source* into the corresponding time.

Name \$currehits - return the number of occurrences in the hitlist.

Synopsis *{field = | register =} \$currehits {entity}*

Return Value The number of occurrences in the hitlist. This value is negative if the hitlist has only been partially built. If *entity* does not exist, -1 is returned.

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

Synopsis *{field = | register =} \$curocc {entity}*

Return Value \$curocc returns the sequence number in the hitlist of the current occurrence, or -1 if *entity* does not exist. The following statements and triggers effect the value of \$curocc:

<NEXT>< OCCURRENCE>	sets \$curocc.
<PREVIOUS>< OCCURRENCE>	sets \$curocc.
retrieve	sets \$curocc.
setocc	sets \$curocc.
<ADD><OCCURRENCE>	modifies \$curocc.
<INSERT><OCCURRENCE>	modifies \$curocc.
<REMOVE>< OCCURRENCE>	modifies \$curocc.
clear	resets \$curocc to 1.

Name \$date - return the current date or convert a date string into the date data type.

Synopsis *{field = | register =} \$date {source}*

Return Value The value of *source* is returned as a date data type. If *source* is omitted, \$date returns the current system date.

Name `$datim` - return the system date and time, or convert the argument to the date and time data type.

Synopsis `{field = | register =} $datim {source}`

Return Value The current system date and time if `source` is omitted. If `source` is given, `$datim` converts the source to date and time format. `source` should be formatted as dd-mmm-yy hh:mm:ss. Be aware that a correct system time value depends on the system clock for the machine being correctly set.

Name `$dberror` - return the specific DBMS error code.

Synopsis `{field = | register =} $dberror`

Return Value The value `$dberror` returns is set when the DBMS or network driver encounters an error situation. The value returned is that given by the DBMS or network to the driver, and is DBMS or network specific.

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

Synopsis `{field = | register =} $dbocc { (entity) }`

Return Value The following values can be returned by `$dbocc`:
`sequence_number` in the database of the current or specified `entity`.

0 If the current occurrence has not been retrieved from the database (it has been entered by the user, and not stored yet).

-1 If `entity` does not exist.

-1 If `entity` is not painted on the external schema.

The following statements and triggers modify `$dbocc`:

`<NEXT>< OCCURRENCE>` sets `$dbocc`.

`<PREVIOUS>< OCCURRENCE>` sets `$dbocc`.

`retrieve` sets `$dbocc`.

`store` sets `$dbocc`.

`clear` sets `$dbocc` to 0.

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

Synopsis `{field = | register =} $direction`

Return Value One of the following values is returned:

0 NEXT mode.

1 PREVIOUS mode.

Name `$display` - return the name of the current display device table.

Synopsis `{field = | register =} $display`

Return Value The value returned by `$display` is the same as the value of the environment variable `UDISP`. This defaults to `VT100` if it is not set.

Name `Empty` - test whether the specified entity or named area frame is empty.

Synopsis `{field = | register =} Empty {(entity) | (named_area_frame)}`

Return Value The `Empty` function returns the following values:

- | | |
|----|--|
| 2 | There are no occurrences of <i>entity</i> or <i>named_area_frame</i> containing data, and the frame definition of <i>entity</i> or <i>named_area_frame</i> has Supp. on Empty set to 'Y' (Yes). |
| 1 | There are no occurrences of <i>entity</i> or <i>named_area_frame</i> containing data but the frame definition of <i>entity</i> or <i>named_area_frame</i> has Supp. on Empty set to 'N' (No), or has been left blank. |
| 0 | The <i>entity</i> or <i>named_area_frame</i> contains at least one occurrence with data in. |
| -1 | The <i>entity</i> or <i>named_area_frame</i> does not exist (usually due to a spelling mistake by you). |

Name `Sentname` - return the name of the current entity.

Synopsis `{field = | register =} Sentname`

Return Value `Sentname` returns the name of the current entity. This name is always in uppercase.

Name `Error` - return the UNIFACE error message number.

Synopsis `{field = | register =} Error`

Return Value The error code returned by `Error` is only valid in the entity or field level ON ERROR trigger. The following codes are trapped by the current version of UNIFACE. The text accompanying the error codes is supplied by default; you can generate your own by trapping the errors in the ON ERROR trigger. For the entity level ON ERROR trigger, the values in table 2-1 apply:

Error code	Default message
0102	Not enough occurrences in entity <i>entity</i> .
0103	Too many occurrences in entity <i>entity</i> .
0118	More occurrences are not allowed.
0139	Entity <i>entity</i> still has restricted links to <i>entity</i> .
0148	First occurrence.
0149	Last occurrence.
2004	No modifications allowed on occurrence of this entity.
2009	Occurrence locked.
2012	Occurrence in external schema does not match database occurrence.
2013	Occurrence no longer exists.

table 2-1 ON ERROR codes and default messages (entity level).

For the field level ON ERROR trigger, the values in table 2-2 apply:

Error code	Default message
0105	Not allowed to change primary/candidate key field.
0120	Error on field " <i>field</i> "; subfield too large.
0121	Error on field " <i>field</i> "; subfield too small.
0122	Error on field " <i>field</i> "; incorrect check digit.
0123	Error on field " <i>field</i> "; illegal format for numeric field.

table 2-2 continues

Error code	Default message
0124	Error on field " <i>field</i> "; illegal format for date field.
0125	Error on field " <i>field</i> "; illegal format for time field.
0126	Error on field " <i>field</i> "; illegal syntax format.
0127	Error on field " <i>field</i> "; illegal entry format.
0128	Error on field " <i>field</i> "; subfield too large to check.
0129	Error on field " <i>field</i> "; subfield(s) are required.
0130	Error on field " <i>field</i> "; too many subfields specified.
0131	Error on field " <i>field</i> "; font not allowed.
0133	Error on field " <i>field</i> "; ruler/frames not allowed.
0134	Error on field " <i>field</i> "; italic not allowed.
0135	Error on field " <i>field</i> "; underline not allowed.
0136	Error on field " <i>field</i> "; bold not allowed.
0137	Error on field " <i>field</i> "; open/close brackets do not match.
0138	Error on field " <i>field</i> "; illegal format for floating field.
0150	Requested number of "&" and "!" operators not supported.

table 2-2 ON ERROR codes and default messages (field level).

Name	<code>\$fieldcheck</code> - require field checking when the user passes through a field.
Synopsis	set <code>\$fieldcheck</code> (<i>field_name</i>)
Return Value	If field checking was successfully enabled, <code>\$status</code> is set to 1. If field checking cannot be enabled, <code>\$status</code> is set to -1. Inability to perform field checking is usually due to giving a <i>field_name</i> that is not present, or does not exist. This is flagged as a warning at compile time.

Name	<code>\$fieldendmod</code> - return the modification status of a field when the field is left.
-------------	--

Synopsis	{ <i>field</i> = <i>register</i> =} <code>\$fieldendmod</code>
-----------------	--

Return Value	The value returned is only valid in the LEAVE FIELD trigger. It is always 1 if the programmer has used a set <code>\$fieldcheck</code> for the current field. The value can be one of:
---------------------	--

0	Not modified.
1	Modified.

Name	<code>\$fieldmod</code> - return the modification status of a field.
-------------	--

Synopsis	{ <i>field</i> = <i>register</i> =} <code>\$fieldmod</code> {(<i>field</i> {. <i>entity</i> })}
-----------------	--

Return Value	The following values are returned by <code>\$fieldmod</code> :
---------------------	--

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

`$fieldmod` is modified by the following triggers and statements:

<EXECUTE>	<code>\$fieldmod</code> is set to 0.
clear	<code>\$fieldmod</code> is set to 0.
erase	<code>\$fieldmod</code> is set to 0.
release	<code>\$fieldmod</code> is set to 0.
reload	<code>\$fieldmod</code> is set to 0.
retrieve	<code>\$fieldmod</code> is set to 0.

store \$fieldmod is set to 0.

Name \$fieldname - return the name of the current field.

Synopsis {field = | register =} \$fieldname

Return Value This function is only valid in a field level trigger. It returns the name (in uppercase) of the current field.

Name \$formdb - test if any occurrences has been retrieved from a database.

Synopsis {field = | register =} \$formdb

Return Value The \$formdb function returns 1 if any entity in the external schema has been retrieved from a database. Only when no entities have been retrieved from a database (or when \$formdb has been reset to 0) will \$formdb be 0. The following Proc statements affect the value returned by \$formdb:

clear \$formdb is reset to 0.

clear/e \$formdb is reset to 0 if the only entities retrieved are related to the cleared entity. If unrelated entities in the external schema have been retrieved from the database, \$formdb is not reset to 0.

erase \$formdb is reset to 0.

erase/e \$formdb is reset to 0 if the only entities retrieved are related to the erased entity. If unrelated entities in the external schema have been retrieved from the database, \$formdb is not reset to 0.

release \$formdb is reset to 0.

release/e \$formdb is reset to 0 if the only entities retrieved are related to the released entity. If unrelated entities in the external schema have been retrieved from the database, \$formdb is not reset to 0.

release/e/mod \$formdb is reset to 0 if the only entities retrieved are related to the released entity. If unrelated entities in the external schema have been retrieved from the database, \$formdb is not reset to 0.

release/mod \$formdb is reset to 0.

retrieve \$formdb is set to 1 by the retrieve statement. ^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 \$formdb is set to 1 by the retrieve statement. 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 \$formdb is set to 1.

store/e \$formdb is set 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 - test if any database field has been modified.

Synopsis {field = | register =} \$formdbmod

Return Value

The value of \$formdbmod is 1 if any fields in the external schema defined as being part of a database have been modified. If no modifications have been made to database fields, \$formdbmod returns 0. The following Proc statements affect the value \$formdbmod returns:

clear	\$formdbmod is reset to 0.
clear/e	\$formdbmod is reset to 0 if the only database fields modified are in entities related to the cleared entity. If unrelated entities in the external schema have database fields that have been modified, \$formdbmod is not reset to 0.
erase	\$formdbmod is reset to 0.
erase/e	\$formdbmod is reset to 0 if the only database fields modified are in entities related to the erased entity. If unrelated entities in the external schema have database fields that have been modified, \$formdbmod is not reset to 0.
release	\$formdbmod is reset to 0.
release/e	\$formdbmod is reset to 0 if the only database fields modified are in entities related to the released entity. If unrelated entities in the external schema have database fields that have been modified, \$formdbmod is not reset to 0.
release/e/mod	\$formdbmod is set 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 external schema).
release/mod	\$formdbmod is set to 1.
remocc	\$formdbmod is set 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	\$formdbmod cannot be reset, but a reset \$formdbmod causes \$formdbmod to be reset.
retrieve	\$formdbmod is reset to 0 if the only database fields that have been modified are in entities related to the

retrieved entity. If unrelated entities in the external schema 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	\$formdbmod is reset to 0 if the only database fields that have been modified are in entities related to the retrieved entity. If unrelated entities in the external schema have database fields that have been modified, \$formdbmod is not reset to 0. Any unrelated entities are not automatically retrieved.
set	\$formdbmod cannot be set. Unlike reset, setting \$formdb has no effect on \$formdbmod.
store	\$formdbmod is reset to 0.
store/e	\$formdbmod is reset to 0 if the only modified database fields are in entities related to the stored entity. If fields in unrelated entities in the external schema have been modified, \$formdbmod is not reset to 0.

Name \$formmod - test if the form has been modified.

Synopsis {field = | register =} \$formmod

Return Value

The value of \$formmod is 1 if any field in the external schema has been modified. If no modifications have been made, \$formmod returns 0. The following Proc statements affect the value \$formmod returns:

clear	\$formmod is reset to 0.
clear/e	\$formmod is reset to 0 if the only fields modified are in entities related to the cleared entity. If unrelated entities in the external schema have fields that have been modified, \$formmod is not reset to 0.
creocc	\$formmod is set to 1. The entity level indicators are only set for the entity and its related entities.

erase \$formmod is reset to 0.

erase/e \$formmod is reset to 0 if the only fields modified are in entities related to the erased entity. If unrelated entities in the external schema have fields that have been modified, \$formmod is not reset to 0.

examine \$formmod is set to 1, and displayed, if any fields of entities in the external schema have been modified. If no fields of entities in the external schema have been modified, \$formmod is reset to 0 and displayed.

release \$formmod is reset to 0.

release/e \$formmod is reset to 0 if the only fields modified are in entities related to the released entity. If unrelated entities in the external schema have fields that have been modified, \$formmod is not reset to 0.

release/e/mod \$formmod is set to 1.

release/mod \$formmod is set to 1.

remocc \$formmod is set to 1. The entity level indicators are only set for the entity and its related entities.

reset \$formmod is reset to 0. For consistency, \$formdbmod is also reset.

retrieve \$formmod is reset to 0 if the only fields that have been modified are in entities related to the retrieved entity. If unrelated entities in the external schema have fields that have been modified, \$formmod is not reset to 0. A ^RETRIEVE causes the first outermost entity to be retrieved along with its related entities. Any unrelated entities are not automatically retrieved.

retrieve/e \$formmod is reset to 0 if the only fields that have been modified are in entities related to the retrieved entity. If unrelated entities in the external schema have fields that have been modified, \$formmod is not reset to 0.

set \$formmod is set to 1. Unlike reset, set does not change the value of \$formdbmod.

store \$formmod is reset to 0.

store/e \$formmod is reset to 0 if the only modified fields are in entities related to the stored entity. If fields in unrelated entities in the external schema have been modified, \$formmod is not reset to 0.

Name \$formname - return the name of the form (external schema).

Synopsis {field = | register =} \$formname

Return Value \$formname returns the (uppercase) name of the current external schema. If no external schema is current, \$formname returns the name of the application.

Name \$framedepth - depth of the painted frame.

Synopsis {field = | register =} \$framedepth {(frame)}

Return Value The value returned is the number of lines on the screen required to paint frame, or if frame is omitted, the number of lines used by the current frame.

Name \$gui - return the mnemonic for the user interface UNIFACE is using.

Synopsis {field = | register =} \$gui

Return Value The mnemonic identifying the current user interface. Valid values include "MTF" for Motif, "OLO" for open look and "CHR" for a character based interface.

Name	\$hits - return the number of occurrences in the hitlist.
Synopsis	{field = register =} \$hits{entity}
Return Value	\$hits returns the total number of occurrences in the hitlist. It is initialized by building the hitlist, which can be time-consuming. The following statements affect the value of \$hits: clear \$hits is reset to 0. release \$hits is reset to 0.

Name	\$ioprint - return the type of message in the message frame.
Synopsis	{field = register =} \$ioprint
Return Value	The following values are returned by \$ioprint: 0 No information. 1 Store sequence messages. 2 One-line I/O messages. 4 Return values from fetch and select statements. 8 Open description block. 16 where and order by description. 32 Generated SQL (if available). 64 System messages such as the command spawned by a spawn statement or an operating system error message. 128 Calls to UOBJECT and data I/O messages.

Name	\$keyboard - set or return the current keyboard translation table.
-------------	--

Synopsis	{field = register =} \$keyboard (= "table")
-----------------	---

Return Value	The value returned is the keyboard model currently in use.
---------------------	--

Name	\$language - set or return the current language code.
-------------	---

Synopsis	{field = register =} \$language (= "code")
-----------------	--

Return Value	\$language returns the country code currently in use.
---------------------	---

Name	\$lines - return the number of lines left on the current (printed) page.
-------------	--

Synopsis	{field = register =} \$lines
-----------------	--------------------------------

Return Value	When printing (\$printing is 1), \$lines returns the number of lines remaining on the page, not including the header or 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 `{field = | register =} $next (field)`

Return Value The `$next` function returns the value of *field* in the next occurrence. A NULL value will be returned when there is no next occurrence. This can be tested for with:

```
if ($next(field) = '')
```

Name `$number` - return the value of the numeric part of a string.

Synopsis `{field = | register =} $number ("string")`

Return Value `$number` returns the value of the leading numeric part it encounters of *string*. If *string* contains no numeric text, or starts with alphabetic text, `$number` returns 0.

Name `$occcheck` - require modification checks for an occurrence.

Synopsis `set $occcheck (entity)`

Return Value If the set `$occcheck` was successful `$status` is set to 1. If the set `$occcheck` failed, `$status` is set to -1. This can be due to *entity* not existing or not being painted on the external schema (this is flagged as a warning at compile-time).

Name `$occdel` - return the removal status of an occurrence (if it has or has not been removed by the user).

Synopsis `{field = | register =} $occdel{ (entity) }`

Return Value This function is only valid in the DELETE trigger. The value returned is one of the following:

0	Occurrence is not marked for removal.
1	Occurrence is marked for removal.
-1	<i>entity</i> does not exist (flagged as a warning at compile time).

`$occdel` is modified by the following statements and triggers:

<code>erase</code>	<code>\$occdel</code> is set to 1.
<code><REMOVE OCCURRENCE></code>	<code>\$occdel</code> is set to 1.

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

Synopsis `{field = | register =} $occddepth`

Return Value The number of lines an occurrence requires to be painted on the screen.

Name `$occmmod` - return the modification status of an occurrence.

Synopsis `{field = | register =} $occmmod{ (entity) }`

Return Value

The following values are returned by `$occm`:

0	Not modified.
1	Modified.
-1	If <i>entity</i> does not exist or is not painted on the external schema.

The value of `$occm` is modified by the following trigger and statements:

EXECUTE	<code>\$occm</code> is set to 0.
store	<code>\$occm</code> is set to 0.
release	<code>\$occm</code> is set to 0.
retrieve	<code>\$occm</code> is set to 0.
clear	<code>\$occm</code> is set to 0.
reload	<code>\$occm</code> is set to 0.

Name

`$page` - return the current (printed) page number.

Synopsis

`{field = | register =} $page`

Return Value

`$page` returns the page number of the page currently being printed. If no page is being printed, `$page` returns 0.

Name

`$password` - return the password used to log on to the database via the specified path.

Synopsis

`{field = | register =} $password (path)`

Return Value

`$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, 0 is returned.

Name

`$previous` - return the value of the field in the previous occurrence.

Synopsis

`{field = | register =} $previous (field)`

Return Value

The `$previous` function returns the value of the previous occurrence of *field*. A NULL value is returned when there is no previous occurrence. This can be tested for with:

```
if ($previous(field) = '')
```

Name

`$printing` - test if printing.

Synopsis

`{field = | register =} $printing`

Return Value

`$printing` returns the following values:

0	UNIFACE is not printing.
1	UNIFACE is printing.

Name

`$prompt` - position the cursor at the specified field when the current Proc ends.

Synopsis

`$prompt = field(. entity)`

Name `$storetype` - return the type of update in a WRITE trigger (insert or update).

Synopsis `{field = | register =} $storetype({entity})`

Return Value The following values are returned:

- 0 Will be updated.
- 1 Will be inserted.

The value of `$storetype` is set to 1 by the store and release/mod statements, and by the ^ADD ^OCCURRENCE function. It is set to 0 by the retrieve statement.

Name `$syntax` - check if the string matches the specified pattern.

Synopsis `{field = | register =} $syntax (string)`

Return Value In a comparison, `$status` is TRUE (non-zero) if the string it is being compared against matches the pattern given as an argument. It is best to use `$syntax` in an if expression. For example:

```
if (@$fieldname = $syntax("New*"))
```

This matches all text entered in the current field that starts with 'New'.

Name `$text` - access text stored in the central message database (UOBJECT).

Synopsis `{field = | register =} $text (idstring)`

Return Value `$status` is set by the `$text` function.

- 1 If the field does not exist.
- 2 If the help file contains information from a different version (message 0019 - Form *formname* has wrong version; you must recompile it is displayed), or cannot be interpreted (the file is not a help file, message 0020 - File *formname* not recognized as application or form is displayed).

Name `$time` - return the system time (pre-version 5.0), use `$clock` instead.

Synopsis `{field = | register =} $time`

Return Value `$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 `{field = | register =} $totdbocc({entity})`

Return Value The total number of occurrences of *entity* currently fetched from the database. The following triggers and statements affect the value of `$totdbocc`:

- ^NEXT ^OCCURRENCE `$totdbocc` is set to the number retrieved.
- ^PREVIOUS ^OCCURRENCE `$totdbocc` is set to the number retrieved.

retrieve \$stodbocc is set to the number retrieved.
store \$stodbocc is set to the number stored.
clear \$stodbocc is set to 0.

Name \$stotlines - return the total number of lines available on the page for printing.

Synopsis *{field = | register =}* \$stotlines

Return Value When UNIFACE is printing (*\$printing = 1*), \$stotlines returns the total number of lines available for printing, excluding the number of lines required for any headers or trailers. If UNIFACE is not printing when \$stotlines is used (that is, *\$printing = 0*), UNIFACE returns a value of -1 to \$status. The value of \$stotlines is 0 when UNIFACE is not printing.

Name \$stotocc - return the number of occurrences of an entity in the external schema.

Synopsis *{field = | register =}* \$stotocc(*entity*)

Return Value \$stotocc returns the number of occurrences in the external schema. When the external schema is empty, \$stotocc always returns 1. The following triggers and statements modify the value of \$stotocc:

^NEXT ^OCCURRENCE \$stotocc is set to the total number of occurrences of an entity.
^PREVIOUS ^OCCURRENCE \$stotocc is set to the total number of occurrences of an entity.

read \$stotocc is set to the total number of occurrences of an entity.

retrieve \$stotocc is set to the total number of occurrences of an entity.

^ADD ^OCCURRENCE \$stotocc is incremented by 1.

^INSERT ^OCCURRENCE \$stotocc is incremented by 1.

^REMOVE ^OCCURRENCE \$stotocc is decremented by 1.

clear \$stotocc is set to -1.

Name \$user - return the user name.

Synopsis *{field = | register =}* \$user(*{path}*)

Return Value \$user returns the current user name.

Name \$variation - return or set the variation code.

Synopsis *{field = | register =}* \$variation [= "string"]

Return Value \$variation returns the current variation code.

Name \$workfilesize - return the size of the virtual memory swap file.

Synopsis *{field = | register =}* \$workfilesize

Return Value `$workfilesize` returns the current size of the complete virtual page swap file, both on disk and in the primary page swap area in real memory (that is, the sum of `$wflsize` and `$wf2size`); in blocks of 512 bytes.

Name `$wflsize` - return the size of the real memory swap space.

Synopsis `{field = | register =} $wflsize`

Return Value `$wflsize` returns the current size of the primary page swap area in real memory, in blocks of 512 bytes.

Name `$wf2size` - return the size of the disk based swap space.

Synopsis `{field = | register =} $wf2size`

Return Value `$wf2size` returns the current size of the page swap file on disk, in blocks of 512 bytes.

Chapter 3 Extracting values from the data

This chapter describes how you can extract values from data, and how to convert from one type of data to another. Quite often, the programmer needs to extract only part of the data, for example, the last five characters in a string, the day of the week of a date, or the fractional part of a number. Another common requirement is to convert values of one type into another. Examples of this include strings to times, strings to numbers, and numbers to strings.

In most situations, UNIFACE is intelligent enough to automatically convert to the appropriate data type. There are some situations, though, where it is necessary to explicitly tell UNIFACE how to convert values. This chapter shows how to do this.

3.1 Strings

The following data types can be used to store strings:

- S - Strings. These are only the ASCII printable character sets.
- SS - Special strings. These allow the use of all the fonts provided by UNIFACE.

3.1.1 Extracting values from strings

When you are working with data stored as a string, you can extract substrings from the total string. This extraction is done by specifying the offset within the string, indicating from which position you want to extract the substring. The format for this is:

```
{destination = } source [start { : num | , end }]
```

The syntax of string extraction is explained in table 3-1:

Parameter	Explanation
<i>start</i>	Position number from which to start extracting.
<i>end</i>	Position number at which to stop extracting.
<i>num</i>	The amount of positions to extract from <i>start</i> .
, (comma)	Follows <i>start</i> if the next parameter is <i>end</i> .
: (colon)	Follows <i>start</i> if the next parameter is <i>num</i> .

table 3-1 String extraction codes.

Examples

The following examples show how to use the string extraction facilities of Proc. The example manipulates the data in the field NAME, which contains the string HOLLERITH. The example also illustrates the use of indirection with a \$register (\$I0). This \$register contains the value 2, which is used as offset into the string.

```
S1 = NAME[4,8] ;extract positions 4 to 8 : (LERTH)
S1 = NAME[1:3] ;extract positions 1 to 3 : (HOL)
S1 = NAME[$I0:4] ;extract positions 2 to 5 : (OLLE)
S1 = NAME[3] ;extract positions 3 to end : (LLERITH)
```

*Note: Using string extraction on a string containing UNIFACE frame markers will **not** copy the frame markers. You have to copy the whole string with an assignment statement.*



3.1.2 Rules for string extraction

The following rules are applied when string extraction is performed.

- *destination* is always set to empty (that is, an empty string is returned) if any of the following are true:
 - *start*, *end* or *num* is less than 1.
 - *start* is greater than *end*.
 - *start* is greater than available number of characters.
- *start*, *end* and *num* may not be an arithmetic expression; [$\$result + 1$] is therefore an illegal construction.
- *start*, *end* and *num* may be a constant, a \$register, \$status or \$result.
- If *end* or *num* has a value greater than the available character positions, the characters are extracted to the end of the string.

- If string extraction is used on a non-string source (for example, a date or numeric field), the value is first converted to string format according to the default display format.
- The first position in the string is always number 1.
- *start*, *end* and *num* must contain either an integer constant or a \$register.
- If string extraction is applied to a field containing subfields, UNIFACE treats the contents of the complete field as one string for the purposes of the extraction (including the separators).
- Extraction always has a lower priority than indirection.

In addition to string extraction, UNIFACE provides the *length* and *scan* statements. These statements are very useful when used with string extraction. The format for these statements is:

```
length string ; set $result to the length of string.
```

```
scan string, "profile" ; set $result to the position of the start of the ; substring that matches profile.
```

3.1.3 Converting to strings

UNIFACE ensures that automatic type conversion takes place when you assign a value to a string field or register. When applying this conversion, UNIFACE uses the display (DIS) format defined for the field or register.

When a local or global register has a data type of any (\$), or a \$register is used, the register inherits the display (DIS) format of the value assigned to it. For example, if a numeric field has a display format of DIS(z99P99), and a value from this field is assigned to a register, the display format of this numeric field is used for the register. This display format is then used if the value in the register is assigned to a string.

For example, a value in a numeric field is assigned to a \$register. The numeric field has a display format of DIS(99P9P99). After the assignment, the \$register will have a numeric data type, and a display format of DIS(99P9P99). If the value in the \$register is then assigned to a string field, the value in the string field will be formatted as 99P9P99.

3.2 Date and time

There are a wide variety of date and time formats. The data types that can be used for a field or register are:

- D - date.
- LD - linear date.
- T - time.
- LT - linear time.
- E - time and date.
- LE - linear time and date.

3.2.1 Information about date and time

Internally, UNIFACE handles all dates, times and *date/times* as double precision float values. This value is always the number of days since the base date of 1-JAN-0000 00:00:00, using days as the unit of measurement. The integer part of this value represents the day, therefore, and the fraction the time part of the day.

This means that a date value (that is, no time included) is really a combined date/time value, with the time set to null. Similarly, a time value is also a combined date/time value with the date value set to null. Be aware that elapsed time is handled in the same way; that is, elapsed time is also the number of days since 1-JAN-0000 00:00:00.

One of the major advantages of this system is that arithmetic with dates or times is extremely simple. For example, adding the value 1.5 to a date is equivalent to adding one and a half days, because for UNIFACE the value 1.5 means one and a half days.

The examples in table 3-2 show how UNIFACE interprets values:

Value	Interpreted as
0	Date (null) time (null)
0.5	Date (null) 12:00:00
1	1-jan-0000 time (00:00:00)
1.5	1-jan-0000 12:00:00

table 3-2 How UNIFACE interprets values.

Year 0 is a leap year

Bear in mind when performing calculations with dates, particularly those involving elapsed times with the results expressed in months, that year 0 contains 366 days and February 29.

3.3 Units of measurement for use in Procs

Each unit of date and time has its own code which you can use in Procs. UNIFACE converts these codes internally to the equivalent number of days. For example, one second (code: '1s') is 1/8640000 (1.1574 * 10⁻⁶) days, and is handled by UNIFACE as such. Using a common unit of measurement for all parts of the whole makes very complex arithmetic possible.

3.3.1 Codes for date and time arithmetic

The codes available for date and time arithmetic are shown in table 3-3:

Code (as fraction of 1 day)		Meaning/Value
d	Day	1
h	Hour	1/24
n	Minute	1/1440
s	Second	1/86400
t	Tick	1/8840000

table 3-3 Date and time arithmetic codes.

In version 5.0, 'n' was introduced to stand for minutes both here and in the supported display format codes. This was done to avoid confusion with months. UNIFACE still recognizes the code 'm' in display format definitions to mean either minutes or months. The true meaning of 'm' in display format definitions is understood according to the context; that is, the position of this syntax code.

Note: There are no arithmetic codes for months or years because neither of these contains a fixed number of days. If you need to add or subtract

months from a date, use the `addmonths` statement. Weeks are not represented here as they are superfluous. After all, a week is always seven days (7d).

3.3.2 Examples of how UNIFACE treats date and time values

The above codes can be used as numeric constants in Procs. The Proc compiler automatically converts these codes to the relevant floating value. For example, table 3-4 shows how various expressions are interpreted by UNIFACE:

Expression	Value (as a fraction of 1 day)
1t	1/8640000
1s	1/86400
1n	1/1440
1h	1/24
1d12h	1.5
2t	1/4320000
2s	1/720
2h	1/12

table 3-4 Example date and time code values.

Arithmetic with date and time can consist of the full range of operations; that is, addition, subtraction, multiplication and division. Use the codes for arithmetic as described in subsection 3.3.1 *Codes for date and time arithmetic*.

3.3.3 Limit values

The range of values recognized by UNIFACE lies between 1 January 0000 00:00:00.00 and 31 December 9999 23:59:59.99, inclusive. Any calculation which results in or uses a value outside these limits is an unreasonable calculation for UNIFACE and the result is not always predictable. Suffice to say that it will be almost certainly incorrect.

*Caution: Some operations are ridiculous in certain circumstances and should not be used. For example, what would a programmer mean with the expression $(2 * \$date)$? On the other hand, the construction*



*$(4 * elapsed_time)/\$1$ is reasonable, if the `ELAPSED_TIME` field contains an elapsed time which is not too big for this calculation, and if $\$1$ contains a numerical value.*

If you enter a value outside these limits, UNIFACE 'beeps' and refuses to continue until you have corrected the error.

3.3.4 Normalization of time and date values

UNIFACE automatically normalizes all expressions, used to the internal equivalent in numbers of days since 1 January 0000 00:00:00.00. This has the following very important implications:

- The designer does not need to worry about which units of date or time to use when coding Procs because (for example) the expression "864000t" (864000 ticks) is the same as "1d" (one day): UNIFACE treats both these expressions as one day, because there are 864000 ticks in a day.
- The results of some calculations might be a little confusing, particularly when doing either of the following:
 - Calculating the elapsed time between two dates.
 - Expressing the result in months.

For example, although the elapsed time between 1 March 1990 and 1 May 1990 is two months, UNIFACE would return an elapsed time of two months and one day, because the difference is 61 days which, in year 0, (year 0 is a leap year) takes us to 1 March.

Be warned that giving years and months a linear display format can have strange effects (see the above examples!). This is because UNIFACE uses the same rules to work out the linear values as are used for non-linear date/times; that is, all values use the base date of 1 January, 0000.

Leap years

Remember that year 0 is a leap year, which means that year 0 has 366 days, and year 1 has 365 days. Year 2 means a total of 731 days, therefore. If the difference between two dates is more than one year, the first year in the 'counter' stands for 366 days and not 365, as you might expect.

Some months are more equal than others

Month 1 has 31 days, because month 1 is January. If you express an elapsed time as the number of months, UNIFACE counts off 31 days for

the first month, even though your elapsed time might be the difference between 3-jul-1990 and 4-sep-1990.

Similarly, the second month is February, which usually comprises 28 days, so UNIFACE counts off 28 days for the second month, even if your elapsed time is the difference between two dates in the middle of the year. However, if the year is a leap year (and year 0 is a leap year), February has 29 days, so UNIFACE counts off 29 instead of 28 days for the second month; month 2 in this case means a total of 60 days, therefore (31 + 29).

For example: DIS(d.m.y)

The elapsed time between 3-jul-1990 and 4-sep-1990 is an arithmetic operation which results in three days, two months and zero years. The actual number of days between these two dates is 63, which means one month of January (31 days), plus one month of February in a leap year (29 days), because there are zero years, and year 0 is a leap year, plus three days.

A recommendation

Where possible, avoid the use of months in linear date display format definitions. The number of days is usually sufficient.

Examples of linear date values

In the following example, LDATE is a field with the display format DIS(d.m.y). The example sets \$registers \$1 to \$5 to various date values, then sets the LDATE field to various values by subtracting one \$register from another.

```
$1 = Sdate(*1 feb 89*)
$2 = Sdate(*1 mar 89*)
$3 = Sdate(*1 apr 89*)
$4 = Sdate(*1 may 89*)
$5 = Sdate(*1 jan 88*)

ldate = $2 - $1      ;ldate is now 28.0.0 (28 days)
ldate = $3 - $2      ;ldate is now 0.1.0 (31 days)
ldate = $4 - $2      ;ldate is now 1.2.0 (61 days)
ldate = $1 - $5      ;ldate is now 0.1.1 (397 days)
ldate = $1 + 5       ;ldate is now 6.1.1989
```

Examples of date and time arithmetic

The following examples show how to do arithmetic operations with date and time values:

```
$1 = start_datim - end_datim ;elapsed (date_)time
$2 = datefield               ;pick up a date value
$3 = timefield               ;pick up a time value
$4 = $2 + $3                 ;make combined date/time
$5 = $4 + $1                 ;add elapsed date/time in $1 to $5
$5 = $5 + 1s                 ;add 1 second to date/time in $5
$5 = $5 + 1m                 ;add 1 minute
$5 = $5 + 1m1s               ;add 1 minute and 1 second
datefield = datefield + 4d   ;add 4 days
datimfield = datimfield + .5d ;add half a day (12 hours)
$1 = datefield + 7.5d        ;come back next week at 12 noon
$1 = datimfield + 7d12h     ;add one week and half a day
```

Note: Assigning a negative value to a linear date, linear time, or linear date/time value, results in an incorrectly displayed value. If you are evaluating an expression that can result in a negative linear value, assign it to a numeric field, not a linear field.



3.3.5 Extracting values from date and time data

When you are working with data stored as a date, a time or a combined date and time, you can extract information such as the week number from a date, the month name from a date, or the number of minutes in a particular time. This extraction is done with an extraction code. The format for using extraction codes is:

(field | register) = source [code]

The type of data to be extracted is specified by the code; each type of extractable data has its own code. For example, the code for extracting the day number from a date is a capital 'D'. All the possible extraction codes are listed in table 3-5:

Extract	Code
Day number	D
Month number	M
Year (four digits)	Y
Fiscal year (four digits)	X
Week number	W
Day of week (Monday = 1)	A
Hour (24 hour clock)	H

table 3-5 continues

Extract	Code
Minutes	N ¹
Seconds	S
Ticks (1/100 of a second)	T
Three-letter month abbreviation	mmm
Three-letter month abbreviation with initial capital	Mmm
Month name (spelled out)	mmm*
Month name (spelled out) with initial capital	Mmm*
Two-letter abbreviation for day name	aa
Two-letter abbreviation for day name with initial capital	Aa
Three-letter abbreviation for day name	aaa
Three-letter abbreviation for day name with initial capital	Aaa
Day name (spelled out)	aa*
Day name (spelled out) with initial capital	Aa*
The date part	date
The time of day from date/time	clock

table 3-5 Date and time extraction codes.

Examples

The following examples show how you can use the extraction codes to test values and to convert a date into the day name:

trigger: LEAVE FIELD

```
if (delivdate[A] > 5) ; 6 or 7, which is Saturday or Sunday
    message "Deliveries cannot be on a weekend!"
    return (-1)
else
    message "Delivery booked for a %delivdate[aa*]" ; day name spelled out.
endif
```

INCORRECT:

```
$1 = "08-nov-1961 22:00:00"
$2 = $1[clock] ; this is incorrect because $1 is a string, not a time.
; $1 needs to be converted into a date/time data type,
; with the use of $datim.
```

CORRECT:

```
$1 = "08-nov-1961 22:00:00"
$2 = $datim($1)[clock] ; value in $1 is a date/time, so $2 will be set to
; 22:00:00
```

1. An 'N' is used to make clear the difference between minutes and months. You can use an 'M' if the context is clear, but this is only provided for pre-version 5.0 compatibility. Use 'N' instead.

The following example shows how date and time arithmetic can be combined with the extraction facilities provided by UNIFACE:

```
if ($time < datimfield[clock] + 22$5)
    message "This date/time is < 22.05 seconds from now"
endif
```

3.3.6 Week numbering

The rule for week numbering in UNIFACE complies with the ISO 2015 standard for week numbering. This standard can be reduced to the following rules:

- Monday is day 1 in the week.
- Sunday is day 7 in the week.
- The rule for determining which week is 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.

3.3.7 Converting to a date or time value

UNIFACE provides the \$clock, \$date and \$datim functions for converting a string to a date or time. If you want to convert numeric data into a time or date, you should define a numeric register with the appropriate display (DIS), assign the numeric value to this register, then use \$clock, \$date or \$datim. These functions are:

- \$clock - convert string argument to time.
- \$date - convert string argument to date.
- \$datim - convert string argument to date and time.

The functions expect the string argument to use the default date and time formats.

3.3.8 Converting to dates

To convert a string into a date, you should use \$date. For example:

```
$1 = "1/2/91"
$2 = $date($1) ; set $2 to the date 1-feb-1991
```

The above example gives the correct value in \$2 if the default date format is dd-mmm-yyyy. If the default date format was mmm-dd-yyyy, the above example would not be correct. The value of \$2 would be jan-2-1991.

The default date and time formats are defined as part of the **Language setup** facility in the **Miscellaneous IDF tasks** menu.

If the argument has a different format to the default, you must convert it into one that UNIFACE can work with. Typically, string information that represents a date, but is formatted in a different way than the default, will be either a text dump from a database or data from another package that uses a UNIFACE supported DBMS. You have the following choices when converting this data:

- Change the default so that the default reflects the data.
- Change the data so that the data reflects the default.

To change the default, you need to change the \$language and \$variation codes to select a default format that is the same as the string argument. Then use \$date to convert the value, then reset \$language and \$variation.

To change the data, you can use a combination of scan and the string extraction functions to build a string that is formatted according to the default date format. For example:

```
$1 = DATE_FIELD_AS_STRING ; formatted as "mm/dd/yyyy"
scan DATE_FIELD_AS_STRING, "/" ; find first "/" character
$3 = $result ; save position
scan DATE_FIELD_AS_STRING[$3], "/" ; find second "/" character
$4 = $result ; save position
$MONTHS = $number(DATE_FIELD_AS_STRING) ; extract month
$DAYS = $number(DATE_FIELD_AS_STRING[( $3 + 1)]) ; extract day
$YEAR = $number(DATE_FIELD_AS_STRING[( $4 + 1)]) ; extract year
; now rebuild date according to DD/MM/YYYY,
; and convert this to a date with $date
$CONVERTED_DATES = $date("##$DAYS/##$MONTHS/##$YEARS")
```

If you do have to implement something like this, it is usually worthwhile defining it as a central Proc, and using global rather than local registers. Functions like these tend to be quite useful. If these functions are frequently used, and speed is required, you can implement them in 3GL. Refer to the *Using 3GL with UNIFACE* manual for more information.

3.3.9 Converting to times

To convert a string into a time, you should use \$clock. The \$clock function uses the default time format. The default time format is defined as part of the **Language setup** facility in the **Miscellaneous IDF tasks** menu.

It is very simple to use \$clock, for example:

```
$1 = "12:23:00"
$2 = $clock($1) ; set $2 to the time 12:23:00
```

The time need not have separator characters, if the time has enough digits (six), \$clock converts it correctly too. For example:

```
$1 = "122300"
$2 = $clock($1) ; set $2 to the time 12:23:00
```

If, however, the argument has fewer than six digits (and no separators), \$clock assumes that part of the time has been omitted, for example, the time is in hours, not hours, minutes and seconds. How \$clock interprets times is shown in table 3-6:

Number of digits	Interpreted as
6	HH:MM:SS
5	H:MM:SS
4	HH:MM
3	H:MM
2	HH
1	H

table 3-6 Converting strings to time with \$clock.

A simple workaround for this situation is to define a numeric local or global register with a display format of DIS(999999). Assign the required value to this register, then convert this register into a time with the \$clock function. If the data may or may not contain separator characters, you may have to write a Proc module like the following:

```
; TEXT_TO_TIME - convert raw text to time
; jim.c
; This Proc converts a free format text field
; into a time field.
; It uses a numeric or string central register,
; depending on whether the data is formatted or not.
; Uses:
; $1 = source
; $2 = result, as a time
```

```

; $Sstring_time = central register, string
; $Snumber_time = central register, number, display as 999999
;
scan $1, ':' ; is $1 formatted ?
if ($result > 0) ; $1 contains a ':'
    $Sstring_time = $1 ; keep format
    $2 = $clock($Sstring_time) ; convert to time using formatted data
else
    $Snumber_time = $1 ; $1 is raw text data, so force leading zeros
    $2 = $clock($Snumber_time) ; convert six digit number to time
endif
endif

```

This central Proc is used in the example given in the description of the EXECUTE trigger.

3.3.10 Converting to a date and time

To convert a string into a combined date and time, you should use \$datim. The \$datim function expects the string to be formatted in the same way as that defined as part of the Language setup facility in the Miscellaneous IDF tasks menu. This is dd-mmm-yy hh:mm:ss for the USA, USYS variation.

The following example shows the use of \$datim to convert a string containing a time and a date into a combined date and time value:

```

$1 = "27-02-66 12:23:39" ; $1 will be a string
$2 = $datim($1) ; $2 will be a combined date and time

```

If either the time or the date that you are trying to convert is not formatted in the same way as the default, you should follow the same steps as those outlined in subsection 3.3.8 *Converting to dates*.

3.3.11 Converting to time from a number

The \$clock function converts a number into a time. As part of the conversion process, the number is converted into a string. By default, a number does not have any leading zeros, so \$clock does not correctly convert numeric values less than 10000. For example:

```

$1 = 000100 ; supposedly 1 minute, $1 will contain the value 100
$2 = $clock($1) ; $2 will be set to 1:00 hours

```

The correct way to convert a number into a time is to use a numeric register that has a DIS(999999) display format. This ensures values are correctly converted to strings, and thence to times.

3.4 Numbers

The following data types can be used to represent numbers:

- N - numeric.
- F - floating point.

3.4.1 Extracting values from numeric data

When you are working with data stored as a numeric or floating point value, you can extract information such as the integer part or the fractional part of this value. This extraction is done with an extraction code. The format for using extraction codes is:

(field | register) = source [code]

The type of data to be extracted is specified by the code. For example, the code for extracting the fractional part of a number is 'fraction'. All the possible extraction codes are listed in table 3-7:

Extract	Code
The integer part of a number	trunc
The rounded value of a number	round
The fractional part of a number	fraction

table 3-7 Number extraction codes.

Examples

The following examples show the use of the UNIFACE number extraction facilities:

```

$25 = 123.76
$1 = $25[fraction] ; extract the fractional value, so $1 = 0.76
$1 = $25[trunc] ; extract the truncated value, so $1 = 123
$1 = $25[round] ; extract the rounded value, so $1 = 124
$1 = $25[trunc] + .11 ; extract the truncated value, and add 0.11,
; so $1 = 123.11

```

3.4.2 Rounding

When rounding, UNIFACE always:

- Rounds up or down from the fraction 0.5 (with 0.5 being rounded up to 1).
- Uses the absolute value of *source* as the basis for rounding. This ensures that *source* [trunc] + *source* [fraction] is always equal to *source*. For example:

```
(where $25 = -123.76)
$25 = $25(round) ;round value in $25: $25 = -124
```

3.4.3 Converting to a number

UNIFACE provides the \$number function to convert strings into numbers. The format for \$number is:

```
{field = | register =} $number ("string")
```

The \$number function does not convert numeric information if it is preceded by alphabetic or punctuation characters. However, the starting position of numeric text can be found by using the scan statement. For example, the following sets \$result to the position of the first numeric character in \$1:

```
scan $1,'#'
```

Example

The following example shows how a combination of scan, \$number and string extraction can be used to extract the numeric part of a string.

```
$1 = "Amsterdam123jim"
scan $1,'#' ; find start of numeric data
if ($result > 0) ; string ($1) contains numeric data
    $3 = $result ; save $result (start position of numeric data)
else
    message "%$1 does not contain numeric data"
    return -1
endif
$2 = $number($1[$3])
putmess "numeric part of %%$1 is %%$2"
```

Chapter 4 Debugging Procs

4.1 Command line

While in debug mode, the bottom line of the screen displays information about Proc statements. This is shown in figure 4-1:

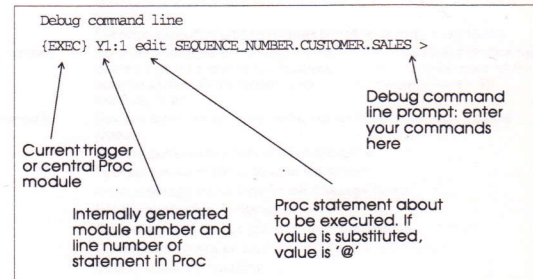


figure 4-1 The debug command line.

While in debug mode, the designer can control the operation of the application very precisely. Debug commands allow the designer to set break points, step through Proc one statement at a time, display the contents of \$registers, fields, status request functions, etc.

The commands available in debug mode are described in section 4.2 *Commands*. Each command is explained in greater detail in the following sections. The commands are summarized in table 4-1:

4.2 Commands

Debug command	Abbreviation	Meaning
break {module} {line}	b {module} {line}	Set a break point at <i>module</i> on <i>line</i> .
break	b	Cancel break point.
call on		Set break points on <i>call</i> statements.
call off		Cancel break points on <i>call</i> statements.
cmrmiss off		Allow messages to build up in the message frame.
cmrmiss on		Clear the message frame normally.
done		Return from 4GL Proc module without executing any further statements.
dump		Dump statements of the current Proc module, either to screen (<i>putmess of</i>), or message frame (<i>putmess on</i>).
dump module		Dump the Proc statements in the named central Proc library.
examine	ex	Display contents of next register.
examine {number}	ex {number}	Display contents of register <i>number</i> .
ex {fname}{,ename}		Display contents of field {fname}{,ename}.
ex number = {val}	num = {val}	Set register <i>number</i> to numeric <i>val</i> , or if <i>number</i> is in double quotation marks (*), a string value. <i>ex</i> is optional (<i>\$1 = 3</i> is also possible).
ex \$function{name}		Display one of the status request functions, for example, <i>sentname</i> , <i>\$fieldmod</i> .
go	g	Continue execution until next break condition or debug command.
ioprint {number}	io {number}	Send I/O messages to message frame, even if application definition has defined that none should be displayed. <i>number</i> = numeric code which determines the I/O messages to send to the message frame, for example, <i>io 63</i> .
line {number}	l {number}	Execute <i>number</i> Proc statements, but do not step through a called module.
nop		Skip the current statement without executing it.
putmess off		Redirect message frame input to the screen.
putmess on		Direct message frame input to the message frame.
quit		Exit the application immediately.
return on		Set break points on <i>return</i> and <i>done</i> statements.
return off		Cancel break points on <i>return</i> and <i>done</i> statements.
show	sh	Display break point settings.
step	s	Execute one (the next) Proc statement.
step {number}	s {number}	Execute <i>number</i> Proc statements and step into called subroutines.
trace on	tr on	Only to be used by Uniface authorized personnel. Use <i>xtrace</i> instead.
trace off	tr off	Only to be used by Uniface authorized personnel. Use <i>xtrace</i> instead.
trace {number}	tr {number}	Send message to message frame about which triggers are activated and when <i>number</i> = 0 or 1.
xtrace		Start the extended trace facility of the UNIFACE debugger.

table 4-1 Debug commands.

4.2.1 break (module) {line}, b (module) {line}

Set a break point at *line* in *module*.

4.2.2 break, b

Clear break points.

These commands are used to define and clear break points within a specific module. The debug command line appears when the statement defined here is about to be executed. For example, the following command sets a break point at the second line in the central Proc module *censtore*:

```
(EXEC) Y1:1 edit cename > b censtore 2
```

The module named can be either a central Proc module, a locally defined entry, or one of the internally generated module names.

4.2.3 call on

Set break points on *call* statements.

4.2.4 call off

Cancel break points on *call* statements.

These commands set and cancel break points at each *call* statement. The debug command line appears before any Proc module is started.

4.2.5 cirmess off

Allow messages to build up in the message frame.

4.2.6 cirmess on

Clear the message frame normally.

These commands control whether I/O and other messages sent to the message frame will be cleared. Sometimes it is useful to allow these messages to accumulate to see a complete picture of how an external schema is operating.

4.2.7 done

Return from the Proc module without executing any further statements.

4.2.8 dump

Dump the statements of the current Proc module either to the screen (*putmess off*) or to the message frame (*putmess on*).

4.2.9 dump module

Dump the Proc statements in the named central Proc library.

Use these commands to see a complete Proc module, in addition to the single statement shown on the debug command line. The statements are sent to the message frame or the screen, depending on the *putmess* status.

4.2.10 examine (number), ex {number}

Display the contents of register *\$number*.

4.2.11 examine, ex

Display the contents of the next register.

4.2.12 ex {field_name}{.entity_name}

Display the contents of a field.

4.2.13 ex number = {value}

Set register \$number to numeric value, or, if number is in double quotation marks ("), a string value. ex is optional. Scity\$ = "Amsterdam" is also possible, therefore.

4.2.14 ex {\$function(name)}

Display one of the status request functions, for example Sentname, \$fieldmod.

Use these commands to examine \$registers, fields and status request functions. In addition, \$registers can be set to a specific value. General registers (\$1 to \$99) can be referred to without the dollar sign, for example ex 79. The \$status and \$result registers are referred to as 100 and 101 respectively.

4.2.15 go, g

Continue execution until a break condition or another debug command is encountered.

This command takes the application out of debug mode until a break point is encountered, or another debug command is executed.

4.2.16 ioprint {number},io {number}

number = numeric code which determines I/O messages.

Send I/O messages to the message frame, even if the application definition has defined that none should be displayed. The following values show the different classes of messages available. The values may be summed to allow several different classes of message to be displayed.

0 No information.

- 1 Store sequence messages.
- 2 One-line I/O messages.
- 4 Return values from fetch and select statements.
- 8 Open description block.
- 16 where and order by description.
- 32 Generated SQL (if available).
- 64 System messages such as the command spawned by a spawn statement or an operating system error message.
- 128 Calls to UOBJECT and data I/O messages.

For example, to only allow store sequence messages and open description blocks, use a value of 9 (the sum of 1 + 8).

See the *Reference Guide* for more information.

4.2.17 line {number}, l {number}

Execute number Proc statements, but do not step through a called module.

These commands are used to execute a specific number of Proc statements. The difference between these two commands is that line considers a call statement as a complete unit; the statements in the called module are not executed individually.

4.2.18 nop

Skip the current statement without executing it.

4.2.19 putmess off

Redirect the message frame input to the screen.

4.2.20 putmess on

Direct the message frame input back to the message frame.

The `putmess off` command causes messages which are normally sent to the message frame to appear immediately on the terminal screen. Use `^REFRESH` to repaint the screen and get rid of these messages. The `putmess` status also determines whether the information requested with `trace` and `dump` is sent to the message frame or directly to the screen.

4.2.21 quit

Exit the application immediately. Using `quit` to leave the debugger when you are prototyping a form terminates the current IDF session.

4.2.22 return on

Set break points on `return` and `done` statements.

4.2.23 return off

Cancel break points on `return` and `done` statements.

These commands set and cancel break points at each `return` and `done` statement. The `debug` command appears just before any `Proc` subroutine is about to end.

4.2.24 show, sh

Display break point settings.

This command displays the current break point settings. For example, if there is a break point on `call`, `return` and line 2 of the `censtore` module, this command shows the following:

```
(EXEC) CENSTORE:2  return on call on> show
```

4.2.25 trace on, tr on

Send driver and 3GL performance information to the message frame (`putmess on`) or to the screen (`putmess off`).

When the tracer is on, the begin time, end time and elapsed time of each driver routine and 3GL routine started with a `perform` statement are sent to the message frame. This feature allows the designer to monitor precisely how long each routine takes to complete.

4.2.26 trace off, tr off

Only to be used by Uniface authorized personnel. Use `xtrace` instead.

4.2.27 trace 1, tr 1

Only to be used by Uniface authorized personnel. Use `xtrace` instead.

4.2.28 trace 0, tr 0

Only to be used by Uniface authorized personnel. Use `xtrace` instead.

4.2.29 step, s

Execute one `Proc` statement.

4.2.30 step {number}, s {number}

Execute *number* `Proc` statements.

4.2.31 xtrace

Start the extended trace facility. This statement puts the debugger into extended trace mode. All Proc statements are copied to the message frame as they are executed. The information written to the message frame includes which trigger the Proc statement is in, which module in the trigger the statement is in, and any arguments given to the Proc statement. This statement is new to version 5.2.

4.2.32 Examining contents of \$registers

Type in the \$register (general, local or central) at the debug command line prompt. For example, if you type in \$1, the debugger shows you what \$1 contains.

4.3 Trigger mnemonics

See table 4-2 for all triggers in the IDF, together with their abbreviations. The debugger and Proc listings use these abbreviations.

Abbreviation	Full trigger name	Level
APPL	APPLICATION EXEC	Application
ASYN	ASYN. INTERRUPT	Application
MNUA	<MENU>	Application
PULA	<PULLDOWN>	Application
SWIT	<SWITCH KEYBOARD>	Application
UKYA	<USER KEY>	Application
ACPT	<ACCEPT>	Form
CLR	<CLEAR>	Form
ERAS	<ERASE>	Form
EXEC	EXECUTE	Form
MNUS	<MENU>	Form
PRNT	<PRINT>	Form
PULS	<PULLDOWN>	Form
QUIT	<QUIT>	Form
RETR	<RETRIEVE>	Form

table 4-2 continues

Abbreviation	Full trigger name	Level
RETS	<RETRIEVE SEQUENTIAL>	Form
STOR	<STORE>	Form
UKYS	<USER KEY>	Form
AIO	<ADD/INS.OCCURRENCE>	Entity
DELE	DELETE	Entity
DTLE	<DETAIL>	Entity
DLUP	DELETE UP	Entity
ERRE	ON ERROR	Entity
HLPE	<HELP>	Entity
LMK	LEAVE MODIFIED KEY	Entity
LMO	LEAVE MODIFIED OCCURRENCE	Entity
LOCK	LOCK	Entity
LPO	LEAVE PRINTED OCCURRENCE	Entity
MNUE	<MENU>	Entity
OBA	OCCURRENCE BECOMES ACTIVE	Entity
READ	READ	Entity
RMO	<REMOVE OCCURRENCE>	Entity
WRIT	WRITE	Entity
WRUP	WRITE UP	Entity
DECR	DECRYPT	Field
DTLF	<DETAIL>	Field
ENCR	ENCRYPT	Field
ERRF	ON ERROR	Field
HLPF	<HELP>	Field
LFLD	LEAVE FIELD	Field
MNUF	<MENU>	Field
NFLD	<NEXT FIELD>	Field
PFLD	<PREVIOUS FIELD>	Field
SMOD	START MODIFICATION	Field
OPTN	TITLE / OPTION	Pulldown menus

table 4-2 Trigger mnemonics

Chapter 5 Naming conventions, reserved words and wildcards

The naming conventions are UNIFACE naming conventions. The reserved words are UNIFACE reserved words. The wildcards are UNIFACE wildcards. Your DBMS and operating system also have naming conventions, reserved words and wildcards. Make sure that you know those also.

5.1 Naming conventions

5.1.1 External schema, conceptual schema, application, field

- Length up to 32 characters, except for external schemas (up to 16).
- A-Z, a-z, 0-9 and underscore () allowed.
- First character must be a letter.
- UNIFACE reserved words are not allowed.

5.1.2 Frames on paint tableau

Frame	Rule
Area	No name allowed
Break	BREAK_FRAME_NAME
Entity	ENTITY.CONCEPTUAL_SCHEMA
Field in break	FIELD.BREAK_FRAME_NAME
Field in entity	FIELD.(ENTITY)
Field in header	FIELD.HEADER
Field in trailer	FIELD.TRAILER
Header	No name allowed
Named area	Same name as enclosed entity
Trailer	No name allowed

table 5-1 Naming conventions for frames.

5.1.3 Global models

- Length up to 16 characters.
- A-Z, a-z, 0-9 and underscore (_) allowed.
- Must begin with the 'at' (@) symbol.
- First character after the '@' must be a letter.

5.2 Reserved words

Do *not* use the following as names for any objects:

- Proc instructions (see *Proc Language Reference Manual*).
- Proc special functions (see *Proc Language Reference Manual*).
- IDF application dictionary names (see subsection 5.2.1 *IDF application dictionary names*).

Do *not* use the following as entity names:

- HEADER.
- PRATT.
- TRAILER.

- UNISCODELI.
- UNIS.

Avoid names which differ only in the first letter, an 'O'. UNIFACE creates overflow tables when needed by adding the letter 'O' to the front of the entity name.

Do *not* use the following as conceptual schema names:

- FRM.
- APS.
- DICT.
- TEXT.

5.2.1 IDF application dictionary names

These are all reserved words and should never be used as the names of any objects defined in your UNIFACE applications:

APPL	OULGROUP	UFIELD	ULREGIS
FORM	OUMISC	UFINT	UMISC
FORMAPPL	OUBJECT	UFLAY	UOBJECT
OAPPL	OUTABLE	UFSYN	UREGIS
OFORM	OUIVIEW	UGINT	URELA
ODOM	PRATT	UGROUP	USYSANA
OUIFIELD	UANA	UKEY	UTABLE
OUGROUP	UCROSS	UIFIELD	UVIEW
OULFIELD	UDOM	ULGROUP	

Avoiding reserved word conflicts

The following ways also avoid the name problem, if you want to use any of the reserved words listed here:

- The application dictionary and the run time application can be kept in separate accounts.
- Different DBMSs can be used for the IDF and the end application.

5.2.2 UNIFACE Reporter application dictionary names

These are all reserved words and should never be used as the names of any objects defined in your UNIFACE applications:

E_CMFLD	E_VIEW
E_FRAME	E_VWENT
E_MEMBER	E_VWFLD
E_REPFLD	E_VWSEL
E_REPORT	E_VWUGRP
E_REPSEL	E_VWUSR
E_SRTFLD	
E_USER	
E_USRGRP	

5.3 Wildcards

Code	Meaning
!	Not
*	0 - n characters (any character, printable or not)
<	Less than
<=	Less than or equal to
!=	Not null
=	Is null
>=	Greater than or equal to
>	Greater than
&	And
	Or
?	Any single character

table 5-2 Wildcard codes supported by UNIFACE.

You may use only one '&' operator or one '|' operator per field. Retrieve profiles in different fields are automatically connected by an '&' operator. The total retrieve profile possible can contain up to 4000 characters (not all DBMSs support this many characters).

Chapter 6 Interface definition

6.1 Data types (UNIFACE)

UNIFACE data type	Explanation
S	String (ISO Latin-1 character set)
SS	Special string (full UNIFACE character set)
R	Raw data
N	Numeric
F	Floating decimal point
D	Date
T	Time
E	Combined date and time
B	Boolean (true or false)
I	Image (for X-bitmaps only)
LD	Linear date
LE	Linear date and time (combined)
LT	Linear time

table 6-1 UNIFACE data types.

Strings and special strings

Strings allow only UNIFACE fonts 0 and 1 to be used; special strings allow all other UNIFACE fonts as well.

Image data type

The Image data type is for use with bitmaps only, for use in pushbuttons. See the *Developers' Guide for GUI Applications*.

6.2 UNIFACE packing codes

Packing codes are specified in the **Conceptual Field interface model** form, or the **Field Assignments** form.

Packing code	Explanation
C1-C*	Character (number if type 'N', in which case only C1-C32 are allowed; this causes numbers to be 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
M1	Money: eight-byte integer, scaling 2
M2	Money: double precision D-float
M4	SYBASE money format, scaling 4
N1-N32	Number, stored without decimal point
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
D2	ASCII date YYYYMMDD
D3	ASCII date DDDMMYYYY
D4	ASCII date YYMMDD
D5	ASCII date DDDMMYY
D6	Binary date YYMD
D7	Binary date DMY
D8	Binary date YMD

table 6-2 continues

Packing code	Explanation
D9	Binary date DMY
D10	Binary date YYMMDD
D11	Binary date DDDMMYY
E	Optimum DBMS combined date/time default
E1	SYBASE linear four-byte date and four-byte time
E2	RMS linear date and time
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 date/time format
E7	SYBASE ASCII date MM/DD/YYYY HH:NN:SS.TT
E8	ASCII datetime YYYYMMDDHHMMSS (like D2+T2)
T	Optimum DBMS time default.
T1	ASCII time HH:NN:SS
T2	ASCII time HHMMSS
T3	ASCII date DDDMMYYYY time HHMMSS
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

table 6-2 Packing codes allowed in UNIFACE.

6.3 Allowed combinations

UNIFACE packing codes	UNIFACE data types								Description	
	SS	S	R	N	F	LD	LE	LT		
C1-C*	•	•	•	•	•	•	•	•	•	Character (number if type 'N')
VC1-*	•									Variable character
SC1-*	•									Segmented character
U1-*	•	•								TRX character
VU1-*	•	•								TRX variable character
SU1-*	•	•								TRX segmented character
R1-*		•								Binary (raw)
VR1-*		•								Variable binary (raw)
SR1-*		•								Segmented binary (raw)
I1-I4										One-byte to four-byte integers
M1-M4					•					Various money formats
P1-8					•					Packed decimal, +/- at beginning
Q1-8					•					Packed decimal, +/- at end
F				•	•					Float (optimum DBMS format)
F4				•	•					Single precision F-float
F8				•	•					Double precision D-float
D								•		Date (optimum DBMS format)
D1-11								•		Various date formats
E						•	•	•		Date/time (optimum DBMS format)
E1-8						•	•	•		Various date/time formats
T								•		Time (optimum DBMS format)
T1-3								•		Various time formats
B1-4									•	Various Boolean (true/false)

Legend: SS = special string (full UNIFACE character set), S = string (ISO Latin-1 character set), R = raw data, N = numeric, F = floating decimal point, LD = linear date, D = date, LE = linear combined date/time, E = combined date/time, LT = linear time, T = time, B = Boolean (true or false), * = possible combination.

figure 6-1 Possible data type and packing code combinations.

6.4 Variable length techniques

String identification method

The string identification method uses ASCII strings to mark the field, the first subfield occurrence (if defined), and subsequent subfield occurrences (if defined). Type the actual string, or, if non-printing ASCII, the decimal value of the ASCII string that you want to use as a string identifier in the **Field identifier** entry. (For example, ^28.)

Length identification

Length identifier	Explanation
I	Identifier string only (default if an identifier string already defined).
1	One-byte binary length identifier.
2	Two-byte binary length identifier.
3	Three-byte binary length identifier.
4	Four-byte binary length identifier.
I1	String identifier, then one-byte binary length identifier.
I2	String identifier, then two-byte binary length identifier.
I3	String identifier, then three-byte binary length identifier.
I4	String identifier, then four-byte binary length identifier.
1I	One-byte binary length identifier, then string identifier.
2I	Two-byte binary length identifier, then string identifier.
3I	Three-byte binary length identifier, then string identifier.
4I	Four-byte binary length identifier, then string identifier.

table 6-3 Length identification for subfields.

Chapter 7 Syntax checks

7.1 Entry format

You can use syntax strings to check whether information entered into a field is in the required syntax. You can specify the required syntax string, or 'pattern' in the **Entry format** of the **Field syntax model** form. You can use the same syntax codes for checking syntax in an `if Proc` statement.

Syntax codes	Explanation
#	One digit.
#*	Zero-n digits.
&	One letter.
&*	Zero-n letters.
@	One letter or one digit or underscore.
@*	0-n letters, digits or underscores.
?	One ASCII character.
?*	0-n ASCII characters.
*	0-n ASCII characters (same as ?*).
A-Z	That uppercase letter, that is, A, or B, or C, etc.
a-z	That letter, upper or lowercase, that is, A or a, B or b, etc.
Any ASCII char	That ASCII character, not a syntax string code.
%any ASCII char (Any)	That literal ASCII character, that is, not a variable code. Syntax strings in brackets are optional. Syntax is checked only if data is present. If there are multiple possibilities, enclose each possibility in brackets. For example, (J)(N).

table 7-1 Syntax string codes.

7.1.1 Entry format examples

Entry format	Allowed (for example)	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 12345XZ	Smith? Mr. Smith 12345*2
# ##-G	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!

table 7-2 Example entry formats.

Procs

In Procs, enclose syntax strings in single quotation marks, for example '?*', or use \$syntax (syntax_string).

Examples (for string fields)

```
if (field1 != '??*') ;test for empty field
if (field1 = '*?#*') ;test for asterisk in last character
if (field1 = '?*#*?#*') ;test for asterisk anywhere in string
if (field1 = '#@*') ;test for field which starts with letter
```

Example \$syntax

```
string = "Amsterdam"
$1 = "A*clam"
if (string = $syntax($1))
  message "String checks out!"
endif
```

7.2 Display/Edit/Prompt

Code	Description
0	Display and edit (this is the default value).
1	Display only.
2	Edit only (for non-echoing passwords).
3	Display and edit, no prompt.
4	Display only, no prompt.
5	Edit only, no prompt.
6	No display, no edit and no prompt.

table 7-3 Display/Edit/Prompt codes.

7.3 Characters allowed in a field

Characters	What they are
Digits only	0-9
Numbers only	0-9, . + -
ASCII only	UNIFACE font 0
ISO Latin-1	UNIFACE fonts 0 and 1
Full char. set	UNIFACE fonts 0 through 7

table 7-4 Characters allowed in a field: what they are.

7.4 Shorthand codes for Field syntax model

Code	Description
ASC	UNIFACE font 0 only.
BRM	Check that brackets match.
DIG	Digits only.
DLC	Delete leading control characters.
DLS	Delete leading spaces.
DLZ	Delete leading zeros.
DTC	Delete trailing control characters.
ENT(<i>syntax code</i>)	Entry format (see section 7.1 <i>Entry format</i>).
FUL	Full character set allowed.
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.
NDCC	Do not delete any control characters.
NDCX	Do not delete text control characters.
NDI	Do not display this field.
NED	No edit allowed in this field.
NITA	Italics not allowed.
NPR	Do not prompt this field.

table 7-5 continues

Code	Description
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.
YITA	Italics allowed.
YUND	Underlining allowed.

table 7-5 Shorthand codes for Field syntax model.

Chapter 8 Display format

This section explains the codes used to define the **Display format** entry of the **Field layout model** form. Display format is used to specify how data should be echoed on the form.

8.1 String

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

table 8-1 String display format codes.

Examples

Display format	Input	Displayed
Mr. ?????	Smith	Mr. Smith
	Jumpin' Jack Flash	Mr. Jumpi
Mr. ??????%	Smith	Mr. Smith?
	Jumpin' Jack Flash	Mr. Jumpi?
Mr. ??????%%	Smith	Mr. Smith%
	Jumpin' Jack Flash	Mr. Jumpi%

table 8-2 Example display format codes for string fields.

8.2 Numeric (and float)

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

table 8-3 Numeric display format codes.

Examples

Display format	Input	Displayed
99999	12345	12345
	123	00123
	00123	00123
	123456	error: "too much data"
	-1234	error: "negatives not allowed"
	123.45	12345 (no point defined)
z	12345	12345
	123	123
	00123	123
	123456	error: "too much data"
	-1234	error: "negatives not allowed"
	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-
+zzzzz	123	+123
	-123	123
+zzzzz	123	+123
	-123	-1234
-Bzzz99	-123	- 123
	1234	1234
B-zzz99	-123	-123
	1234	1234
999P99	123	123.0
	123.45	123.4
	12.3	012.3
	1234.5	error: "too much data"
	123.456	error: "too much data"
zzz9P9zzzz	123	123.0
	.8970	0.897
	012.120	12.12
zz.zz.zzz	12345678	123.45.678
	12345	12.345
	1.234	1.234
	123.45.67	123.45.67

table 8-4 Example display format codes for numeric fields.

8.3 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 lowercase abbreviation from day name.
AA	As aa, by uppercase.
Aa	As aa, but initial caps.
aa*	Full day name in lowercase.
AA*	As aa*, but uppercase.
Aa*	As aa*, but initial caps.
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 lowercase abbreviation for month.
MMM	As mmm, but uppercase.
mmm*	Full month name in lowercase.
MMM*	As mmm*, but uppercase.
Mmm*	As mmm*, but initial caps.
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.

table 8-5 Date display format codes.

Examples (non-linear)

Display format	Displayed (1)	Displayed (2)
Mmm* d, yyyy	March 16, 1990	June 2, 1990
AA, MM d	FRI, MAR 16	SAT, JUN 2
dd/mm/yy	16/03/90	02/06/90
mm/dd/yy	03/16/90	06/02/90
d/m/yy	16/3/90	2/6/90
zd/zm/yy	16/ 3/90	2/ 6/90

table 8-6 Example display format codes for date fields.

Examples (linear)

Display format	Value	Displayed
Lzd.yyyy	25 December, 1990	359.1990
Ldd.mm.yyyy	25 December, 1990	25.11.1990
Ldd.mm.yyyy	11 months and 25 days	25.11.0

table 8-7 Example display format codes for linear date fields.

8.4 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).

table 8-8 Time display format codes.

Examples (non-linear)

Display format	Displayed (1)	Displayed (2)
hh:mm	16:15	09:05
h:mm	16:15	9:05
hh:mm.ss	16:15.2	09:05.0
h:mm.s	16:15.2	9:05.3
zh:zm:zs	16:15.2	9: 5: 3

table 8-9 Example display format codes for time.

Examples (linear)

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

table 8-10 Example display format codes for linear time.

8.5 Combined date and time

Combined date and time fields use date and time display format codes.

Example

Display format	Displayed
dd MM yyyy hh:mm:ss	2 APR 1991 14:15:39

table 8-11 Example display format for date and time.

8.6 Shorthand codes for Field layout model

Code	Description
BLI	Blinking.
BOR	Borderlines.
BRI	Bright.
CTR	Center alignment.
DEC	Decimal alignment.
DIS(<i>format</i>)	Display format (see section 8 <i>Display format</i>).
INV	Inverse video.
LFT	Left alignment.
NAV	No active field video.
NBR	Not bright.
NBL	Not blinking.
NIN	Not inverse video.
NUN	Not underline.
RGT	Right alignment.
SEP(<i>c</i>)	Use a subfield separator <i>c</i> .
UND	Underline.
WID(<i>n</i>)	Line width of <i>n</i> characters.

table 8-12 Shorthand codes for Field layout model.

Chapter 9 Video and color

9.1 Video attributes

These entries specify the video attributes of the frame you are currently defining. The default for all of these options is defined at installation.

Attribute	Explanation
Inverse	<code>inverse</code> (only valid for fields).
Bright	<code>VERY BRIGHT</code> (only valid for fields).
Blinking	Blinking fields: impossible to re-create here!
Underlined	<code>UNDERLINED</code> (only valid for fields).
Color number	See figure 9-1.

table 9-1 Video attributes in the Frame Definition form.

As with video attribute definitions anywhere in UNIFACE, you can combine these attributes if you want. That is, you can use more than one definition if required. Most of the definitions, as you can see in table 9-1, only apply to fields.

If you apply these definitions to *field frames*, these entries override the installation defaults and any entries in the **External schema definition** form (the latter provide the defaults for all the fields in the whole external schema). The video attribute definitions in the field layout model override the frame definition *and* those in the external schema definition. The settings supplied by the `field_video` Proc statement at run time can override all previous settings.

9.2 Color definition

The standard USYS colors are not available on all displays, and some local definitions (for example the 'palette' definitions of a VT340G) can cause these definitions to appear differently. Remember that some combinations provide very disturbing results, and can be almost illegible.

USYS color code matrix		Foreground colors							
		system	blue	green	cyan	red	purple	brown	white
Back-ground colors	system	0*	1	2	3	4	5	6	7
	blue	8	9*	10	11	12	13	14	15
	green	16	17	18*	19	20	21	22	23
	cyan	24	25	26	27*	28	29	30	31
	red	32	33	34	35	36*	37	38	39
	purple	40	41	42	43	44	45*	46	47
	brown	48	49	50	51	52	53	54*	55
	white	56	57	58	59	60	61	62	63*


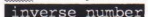
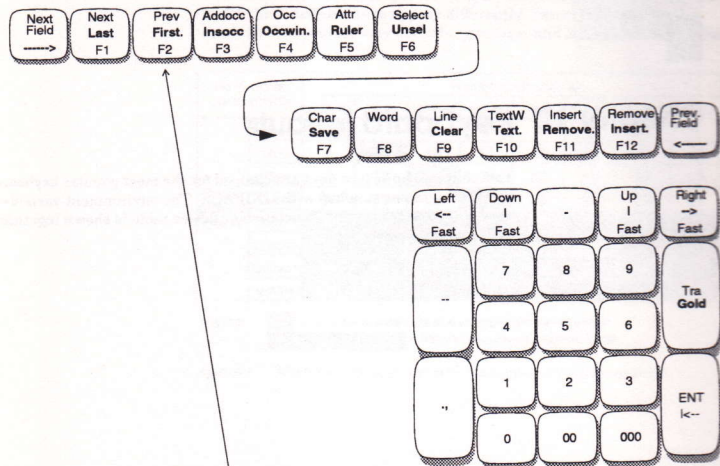
Legend:  = Uses system defaults due to impossible combinations.
 inverse number = Inverse on a monochrome terminal.

figure 9-1 USYS color codes, allowed in the Video color entry of frame definition.

Chapter 10 Keyboard layouts

This chapter shows how keys are mapped for the most popular keyboard translation tables supplied with UNIFACE. The environment variable needed to load the correct translation or device table is shown together with each keyboard chart.

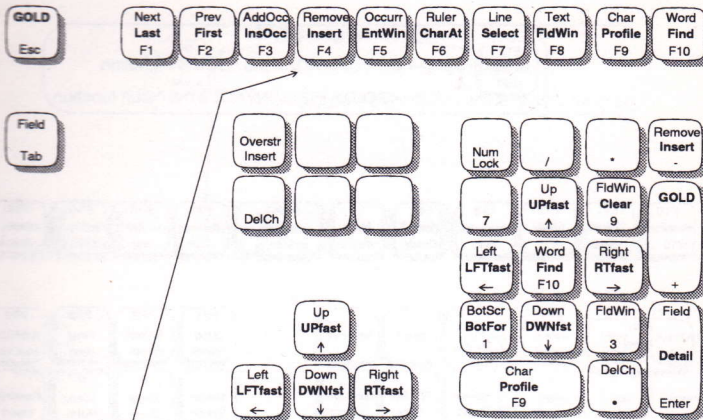
10.1 Bull TWS 2103



Prev First F2
 ← F2 in UNIFACE is the ^PREVIOUS function
 ← <SHIFT>F4 in UNIFACE is the ^FIRST function
 ← F2 outside UNIFACE

TERM = tws2103

10.2 Data General FKB4700



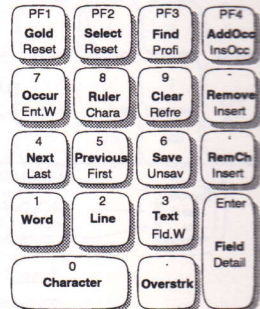
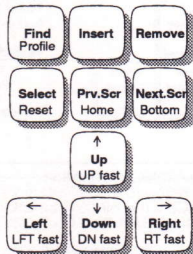
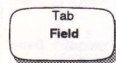
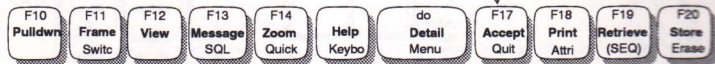
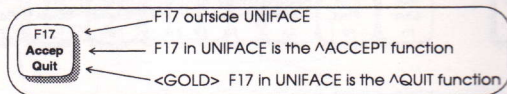
Remove Insert F4
 ← F4 in UNIFACE is the ^REMOVE function
 ← <GOLD>F4 in UNIFACE is the ^INSERT function
 ← F4 outside UNIFACE

UDISP = vt100, UKEYB = ncd

To see borderlines correctly in character mode, you should start an xterm with the following command:

xterm -d display -fn8x13 -fb8x13bold &

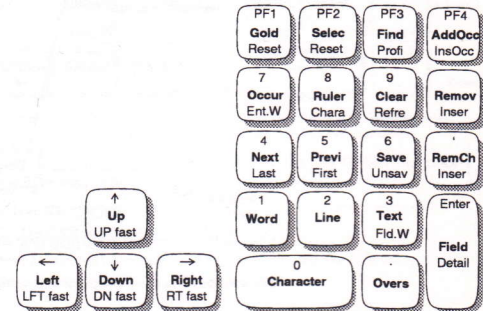
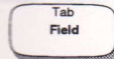
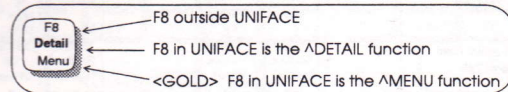
10.3 DEC VT100/200



Depending on your system:

TERM = vt100
 UKEYB = vt100
 UDISP = vt100

10.4 Hewlett Packard HP-HIL



UDISP = vt100, UKEYB = HP_HIL

10.5 IBM PC AT 83/84 key

10.6 IBM AT 101/102 key enhanced

Other:

Not in zoom: Field
In zoom: Tab
Tab

Not in zoom: Field
In zoom: Tab
Field

Next Last F1	Prev First F2	GOLD Esc	Num Lock	Scroll Lock	Sys Req
AddOcc InsOcc F3	Remove Insert F4	PrvScr TopFor 7	Up UPfast 8	FidWin Clear 9	PrtScr -
Occurr EntWin F5	Ruler CharAt F6	←	5	→	Remove Insert -
Line Select F7	Text FidWin F8	NxtScr BotFor 1	Down DWNfst 2	FidWin 3	Field Detail
Char Profile F9	Pulldwn F10	Overstrike/insert Detail 0	.	DelCh	Enter

Remove
Insert
F4

F4 in UNIFACE is the ^REMOVE function
 <GOLD>F4 in UNIFACE is the ^INSERT function
 F4 outside UNIFACE

This keyboard is set automatically under MS-DOS.

GOLD Esc	Next Last F1	Prev First F2	AddOcc InsOcc F3	Remove Insert F4	Occurr EntWin F5	Ruler CharAt F6	Line Select F7	Text FidWin F8	Char Profile F9	Pulldwn Find F10
-------------	-----------------	------------------	------------------------	------------------------	------------------------	-----------------------	----------------------	----------------------	-----------------------	------------------------

Other:

Not in zoom: Field
In zoom: Tab
Tab

Not in zoom: Field
In zoom: Tab
Field

Ovr/ins	TopScr TopFor	FidWin Clear	Num Lock	/	.	Remove Insert -
DelCh	BotScr BotFor	FidWin	TopScr TopFor 7	Up UPfast 8	FidWin Clear 9	GOLD
			←	5	→	
			4		6	+
Up UPfast ↑			BotScr BotFor 1	Down DWNfst 2	FidWin 3	Field Detail
Left LFTfast ←	Down DWNfst ↓	Right RTfast →	Overstrike/insert Detail 0	.	DelCh	Enter

Remove
Insert
F4

F4 in UNIFACE is the ^REMOVE function
 <GOLD>F4 in UNIFACE is the ^INSERT function
 F4 outside UNIFACE

UNIX: UKEYB & UDISP = IBMPCX

To force this layout on an MS-DOS machine, use the following command:

```
set ukeyb=ibmpce
```

To force enhanced keyboard calls, use the following command:

```
set enhkeybios=y
```

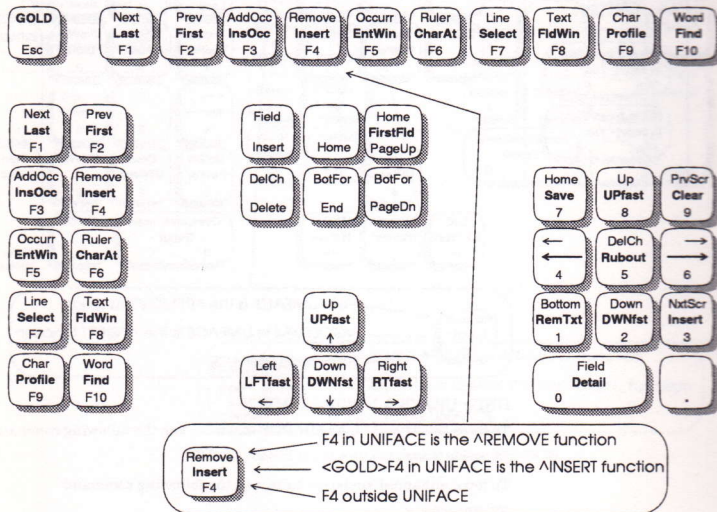
For HELP, use GOLD-H.

Caution: This keyboard translation table maps both UNIFACE characters 7.S and 7.s to IBM storage ^205 (the double-width horizontal line), because both characters have the same shape on the screen. This means that ^237 is also mapped in this way, which can be confusing if you



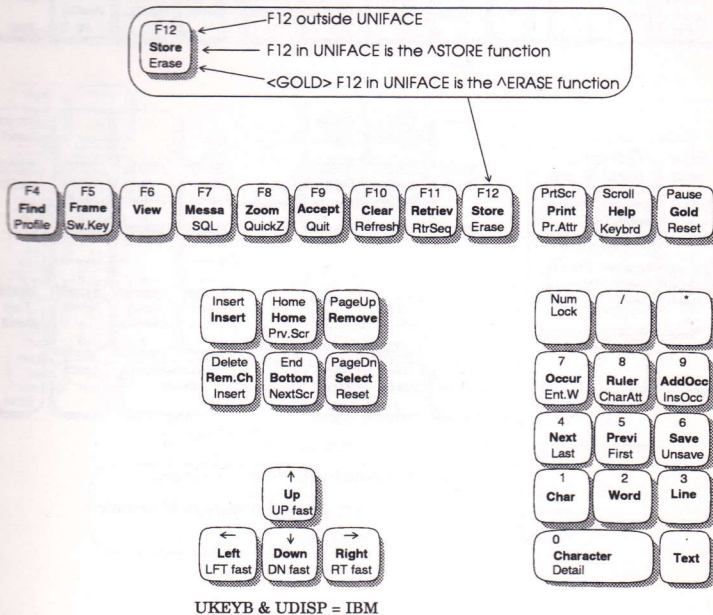
want to recognize ^237 as a different character. If you want to use ^237 separately, map it to 'phi small', which is UNIFACE character 4.u (4.^117).

10.7 IBM RS6000 console



UDISP & UKEYB = IBMRS6000

10.8 IBM 6150 RT PC console



UKEYB & UDISP = IBM

10.11 SCO-Xenix

GOLD Esc	Next Last F1	Prev First F2	AddOcc InsOcc F3	Remove Insert F4	Occur EntWin F5	Ruler CharAt F6	Line Select F7	Text FldWin F8	Char Profile F9	Word Find F10
-------------	--------------------	---------------------	------------------------	------------------------	-----------------------	-----------------------	----------------------	----------------------	-----------------------	---------------------

Next Last F1	Prev First F2	Field Insert	Home	Home FirstFld PageUp
AddOcc InsOcc F3	Remove Insert F4	DelCh Delete	BotFor End	BotFor PageDn
Occur EntWin F5	Ruler CharAt F6	Home Save 7	Up UPFast 8	PrvScr Clear 9
Line Select F7	Text FldWin F8	← 4	DelCh Rubout 5	→ 6
Char Profile F9	Word Find F10	Bottom RemTxt 1	Down DWNfst 2	NxtScr Insert 3
		Left LFTfst ←	Up UPfst ↑	Right RTfst →
			Field Detail 0	.

Remove Insert F4

- ← F4 in UNIFACE is the ^REMOVE function
- ← <GOLD>F4 in UNIFACE is the ^INSERT function
- ← F4 outside UNIFACE

UDISP & UKEYB = ibmpcx

10.12 Siemens 97801

Save F1	Ins sel Rem sel F2	Reset Select F3	Ins/Ove F4	F5	View F6	Attribute F7	Ruler F8	Frame F9	QuickZm Zoom F10	Message F11
------------	--------------------------	-----------------------	---------------	----	------------	-----------------	-------------	-------------	------------------------	----------------

Italic Bold F12	Font Undline F13	Retrieve F14	Rtr.Seq F15	Store F16	Ins.Occ Add Occ F17	Profile Find F18	SQL F19	Ins.File SaveFile F20	Menu F21	Erase F22
-----------------------	------------------------	-----------------	----------------	--------------	---------------------------	------------------------	------------	-----------------------------	-------------	--------------

Keyboard Help Help	Detail Start	End	Insert Remove	Char Word	Line	Field	TextWin Text
--------------------------	-----------------	-----	------------------	--------------	------	-------	-----------------

Other:

Prev.Field ←	Next Field →
-----------------	-----------------

rub		Prev Next	Top Form	Refresh Clear C/E
		Bottom Home	Bottom Form	
	Return	UpFast Up ↑		
		LeftFast Left ←	RightFast Right →	
	OccWin Occ.	DownFast Down ↓		Enter

Font Undline F13

- ← <ESC>F13 in UNIFACE is ^FONT function
- ← F13 in UNIFACE is ^UNDERLINE function
- ← F13 outside UNIFACE

TERM = 97801.

10.13 Stratus v102

Bold CHAR Word F0	Italic TEXTWINDO Text F1	Underline OCCWINDO Occurrence F2	Font FIELD Line F3	QUICK ZOOM Zoom F4	Refresh FRAME Message F5	REMOVEFILE Insert File F6
PROFILE Find text F7	Keyb help HELP Menu F8	PRINT ATTR Print F9	Add occur SWCH KEY Save F10	Ins. occur REM. SELECT Insert select F11	Rem. occur INSERT Remove F12	RESET SEL. Select F13
RETR. SEQ Retrieve F14	ERASE Store F15	VIEW Prev. Occurr F16	LAST Next F17	FIRST Previous F18	ATTR Next Occurr F19	QUIT Accept F20

RULER Detail F21	SQL Insert/Ovrstrike F22
-------------------------------	---------------------------------------

Other:

Not in zoom: Field
In zoom: Tab
ENTER

Clear
<C/E>

Cursor movements:

= Normal movements
 = Quick movements
 = Home
 = Bottom

← **Keyb help**
 HELP
 Menu
 F8
 ← <ESC><F8> in UNIFACE shows the keyboard help
 ← <SHIFT><F8> in UNIFACE is Help
 ← <F8> in UNIFACE is Menu
 ← <F8> outside UNIFACE

10.14 Sun-3

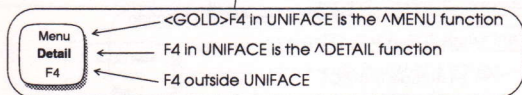
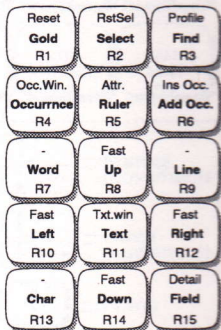
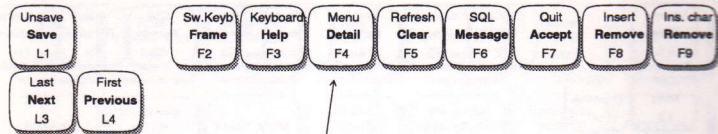
Unsave Save L1	Sw. Keyb Frame F2	Keyboard Help F3	Menu Detail F4	Refresh Clear F5	SQL Message F6	Quit Accept F7	Insert Remove F8	Ins. char Remove F9
Last Next L3	First Previous L4							

Reset Gold R1	RstSel Select R2	Profile Find R3
Occ. Win. Occurrence R4	Attr. Ruler R5	Ins Occ. Add Occ. R6
- Word R7	Fast Up R8	- Line R9
Fast Left R10	Txt. win Text R11	Fast Right R12
- Char R13	Fast Down R14	Detail Field R15

← <GOLD>F4 in UNIFACE is the ^MENU function
 ← F4 in UNIFACE is the ^DETAIL function
 ← F4 outside UNIFACE

UDISP & UKEYB = sun
Using the VT100 tool on the Sun console gives a better screen.

10.15 Sun-4 SPARC station



UDISP & UKEYB = sun

Using the VT100 tool on the Sun console gives a better screen.

10.16 Function key combinations (all keyboards)

Press	Then press	For
	A	Accept
	B	Bold (toggle)
	C	Command menu
	D	Detail
	E	Erase
	F	Frame
	G	clear
	H	Help
	I	Italic (toggle)
	J	compose character
	K	Keyboard help
	L	pulldown
	M	Message
	N	retrieve sequential
	O	Overstrike (toggle)
	P	Print
	Q	Quit
	R	Retrieve
	S	Store
	T	The ruler
U	Underline (toggle)	
V	View (toggle)	
W	SQL Workbench	
X	character attributes	
Y	switch keyboard	
Z	Zoom	
.	select	
*	find	
	refresh	
	Z	quick Zoom
	.	reset select
	,	profile
		reset gold

table 10-1 continues





Press	Then press	For
		named insert ('burp')
		named remove ('slurp')

table 10-1 Function key combinations (all keyboards).

10.17 'Super' key combinations

As with the above function keys, these combinations are possible on almost any keyboard. All you need to do, therefore, is find the GOLD key and the space bar. Note that some of the standard USYS keyboard layouts do not support a GOLD key, however.



Press	Then press	To set mode	Then press	To apply mode to
		A	C	Character
		I	W	Word
		R	L	Line
		T	.	selected block
		B	D	Data (text window)
		N	F	Field
		P	O	Occurrence
			E	Entity window
			S	Screen
			X	file

table 10-2 'Super' key combinations.

Example



Chapter 11 IDF command switches



Note: Some operating systems have difficulties with an asterisk (), so the IDF compilers interpret the percent symbol (%) in the same way as an asterisk. Note also that some operating systems attach their own interpretation to some wildcard codes. In this case you must use the convention for your operating system to ensure that the complete profile is passed to the IDF. For example, you might need to enclose the profile in double quotation marks ("").*

11.1 Switches

Switch or sub-switch	Purpose of switch
IDF compiler switches	
idf /all *	Compile everything.
idf /apl <i>application</i>	Compile <i>application</i> (s).
idf /app <i>application</i>	Compile <i>application</i> (s).
idf /bar	Compile menu bar and pulldowns in <i>variation</i> .
idf /cen <i>variation</i>	Compile central objects in <i>variation</i> .
idf /con <i>conceptual schema</i>	Compile <i>conceptual schema</i> .
idf /cross (all)	Start IDF with xref on, or compile & xref all.
idf /dev <i>variation</i>	Compile device tables in <i>variation</i> .
idf /frm <i>external schema(s)</i>	Compile <i>external schema</i> (s).
idf <i>external schema</i>	Compile <i>external schema</i> (s).
idf /ins <i>installation object</i>	Install and compile UOBJECT or demo.
idf /lib <i>library</i>	Compile central Procs in <i>library</i> .
idf /men <i>variation</i>	Compile menu bar and pulldowns in <i>variation</i> .
idf /mes <i>variation</i>	Compile messages in <i>variation</i> .
idf /obj <i>variation</i>	Compile central objects in <i>variation</i> .
idf /trn <i>variation</i>	Compile translation tables in <i>variation</i> .
idf /tra <i>variation</i>	Compile translation tables in <i>variation</i> .
IDF compiler sub-switches	
/fil	Name new translation table to compile.
/inf	Compile and return all compiler messages.
/lis	Compile with Proc listing.
/war	Compile with warning and error messages.
Other IDF switches	
idf /cpy <i>DBMSs and files</i>	Convert data <i>files</i> from <i>DBMS</i> to <i>DBMS</i> .
idf /exp <i>application export_file</i>	Export <i>application</i> to <i>export_file</i> , after /pre.
idf /hlp	Show this list.
idf /bel	Show this list.
idf /imp <i>export_file</i>	Import the <i>export_file</i> exported with /exp.
idf /key <i>file</i>	Create keyboard translation table <i>file</i> .
idf /lin <i>application</i>	Link <i>application</i> imported with /imp.
idf /lnk <i>application</i>	Link <i>application</i> imported with /imp.
idf /pre <i>application</i>	Prepare <i>application</i> for distribution.

table 11-1 continues

Switch or sub-switch	Purpose of switch
idf /prp <i>application</i>	Prepare <i>application</i> for distribution.
idf /pro <i>external schema</i>	Prototype <i>external schema</i> .
idf /zma	Start the UNIFACE run time manager.
idf /tst <i>external schema</i>	Test <i>external schema</i> .
idf /who	Show installation defaults.
Sub-switches for 'other IDF switches'	
/cut= <i>n</i>	Export /exp file to <i>n</i> Kbytes segment files.
/deb	/pro or /tst in debug mode.
/int	Initialize interval counter with /cpy.
/sup	Set 'supersede' on with /cpy.

table 11-1 IDF switches and sub-switches.

11.2 Sub-switches

Switch or sub-switch	Purpose of switch
General UNIFACE switches	
/asn= <i>assignment file</i>	Run application with <i>assignment file</i> .
/bat	Run application in batch mode.
/log= <i>login info</i>	Provide DBMS or network <i>login info</i> .
/pri= <i>n</i>	Send type <i>n</i> information to message frame.
/fi= <i>TFO_file</i>	Play back application with <i>TFO_file</i> .
/fto	Record application session to <i>TFO file</i> .

table 11-2 UNIFACE switches or IDF sub-switches.

Chapter 12 Assignments

12.1 Priorities

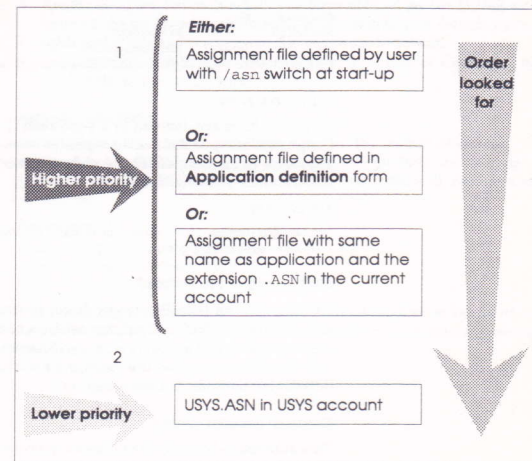


figure 12-1 When assignment files are read, and their priority.

Within any assignment table, the higher up in the file that the physical position of an assignment is, the higher the priority.

12.2 Syntax

With the exception of a small number of 'single word' assignments for UNIFACE system settings, an assignment always has two parts. The first part defines the string expected by UNIFACE. The second part gives the assignment for that string. Each part can be separated by spaces, tabs or an equal sign (=), as shown in the examples below:

```
part1 part2
part1 = part2
```

Examples

```
$language = USA
$variation development
entity1.conc_schema1 $SYB:entity1.*
file_name /home/central_park/textfiles/another_file
```

Comment lines

Comment lines are denoted by a semicolon (;) in the first position. Comments must be defined on separate lines. They can be inserted anywhere in an assignment file and are generally used to enhance the understanding of what is going on.

For example:

```
; these are comment lines which can contain information to
; enhance understanding
```

UPPERCASE or lowercase?

Path names and DBMS/network driver mnemonics are not case-sensitive. UNIFACE system settings and parameters are also not case-sensitive, and are shown here in lowercase. File and login specifications depend on the operating system or DBMS, or both, in use. (UNIX, for example, is case-sensitive.)

Context-specific syntax

This subsection has only introduced a generic format for all assignments. Make sure to read carefully the exact requirements of the following types of assignment:

- Entity assignments.
- Path definitions (path to path, and path to DBMS or network driver).
- Wildcards in assignments.

These are explained below.

12.3 Entity assignments

The general syntax of an entity assignment is:

```
entity.conceptual_schema = $path:table{.*|.extension}
```

Where:

- *entity* is the entity name used by UNIFACE.
- *conceptual_schema* is the conceptual schema containing that entity.
- *\$path* : is either the path which you have defined for the DBMS or network driver (in an assignment), or the installation default path.
- *table* is the name of the table or file in the DBMS itself.
- *.*|.extension* is either the extension given to table or file names by the DBMS in use, or an asterisk (*).

Asterisk extension

An asterisk in the extension position signifies the default extension assumed by the DBMS driver; for example *.rms* for RMS, nothing for ORACLE and SYBASE (tables in these last two DBMSs do not have an extension).

For example:

```
family.dictionary = $MYTEL_PATH:family.*
accounts.savings = $SYB:accounts.*
company.log = $INT:company.*
```

The remaining assessment of wildcard profile characters is the same as for any other UNIFACE files (text files, *.frm* files, application screens, and so on).

12.4 Path assignments

12.4.1 Path to DBMS or network driver

Each path to a DBMS that you specify results in a channel to the DBMS, that is, a different login. UNIFACE supports up to four different open channels per DBMS. Not all DBMSs support more than one login.

When naming a DBMS or network driver, you use the three-letter mnemonic for that driver, without the dollar sign (\$) used for paths, and followed by a colon (:).

The full syntax for this type of path definition is as follows:

```
$path = DBMS_driver: {database} | {username} | {password}
```

or:

```
$path = DBMS_driver: {servername} | {username} | {password}
```

or:

```
$path = network_driver: {network_node} | {username} | {password}
```

or:

```
$path = network_driver: {network_server} | {username} | {password}
```

If the driver mnemonic is not followed by login information, this indicates the end of the specification and UNIFACE assumes that any further information should be taken from the **Logon** form.

```
$SYB SYB:dict | berks | kennet
$RDB $SYB
$IDF SYB: ? | ? | ?
$DEF SYB: ? | ? | ?
$PRCD ORA: {davis} | {belmont}
$TEST $ORA
$BETA $ORA
```

12.4.2 Path to path

```
$path_1 = $path_2
```

For example:

```
$SYB = $RDB
$IDF = $MVRDB_PATH
$ORA = $MVRDB_PATH
$DEF = $VAX3
```

Here follows another example:

```
$ORACMD ORA: {scott} | {tiger}
$SYSTEM SYB: {pubs} | {public} | {berkeley}
$PRODUCTIONSYB: {prod} | {and} | {bracknell}
$TEST $PRODUCTION
; test assigned to production after Beta period
```

Reassigning default paths

For example:

```
$SYB = $DMS_RDB
```

Reassigning DBMS

For example:

```
$DMS_CIS = RMS;
```

12.5 Wildcard assignments

Two wildcard characters are permitted in certain cases. They are the asterisk (*) and question mark (?).

Asterisks

An asterisk can be used as a wildcard in all parts of the assignment file. For example:

- Entity assignments.
- Conceptual schema definition assignments.
- Assignment of files and table names in the file specification.
- Extension in the file specification.

Caution: The wildcard character '' can be used to make assignments for groups of files. The assignments in a table are evaluated sequentially, from the top down. Therefore, place specific assignments above assignments with a wildcard.*

Question marks

Question marks (?) stand for either of the following:

- One character, when assigning file names.
- One login parameter in driver assignments.



It is not necessary to provide complete information in path-to-driver definitions. If a question mark (?) is substituted for either *database*, *username*, or *password*, the **DBMS Logon** form appears and requests this information when UNIFACE opens the DBMS.

When using one or more wildcards, your assignment is effectively:

profile = *assignment*

The *profile* for any file, entity or conceptual schema assignment is the file name as understood and used by UNIFACE. Any part of this name can be substituted by wildcards.

All non-wildcard profile characters should match the corresponding part or parts of the UNIFACE file name. These characters are not case-dependent. That is, UPPERCASE and lowercase characters equate to each other. An asterisk in any place other than the extension for entity assignments means 'zero or more characters' in the corresponding part or parts of the UNIFACE entity name.

For example:

Profile	Matches which part of name 'ABCDEFGH.ABC'
A*	BCDEFGH.ABC
ABC*.ABC	DEFGH
AB*.A*	First *: CDEFGH, second *: BC

table 12-1 Example profile matching in assignments.

When comparing a file name with the resulting profile, if UNIFACE finds a match, an 'assembly' of the real profile and assignment is said to take place. By 'real', we mean the profile and assignment with the wildcards replaced by non-wildcard characters.

For example:

```
; UNIFACE file name is ABCDEFGH.ABC
A* = X*
; UNIFACE file mapped to XBCDEFGH.ABC
ABC*.ABC = X*.Y
; UNIFACE file mapped to XDEFGH.Y
AB*.A* = X*.Y*
```

12.6 UNIFACE system settings and options

Assignment	Values	Explanation	Default
Sactive_field	BOR	Sets active field indicator on	N/A
Sch_virt	True or false	True = keep in virtual memory (MS/PC-DOS only) False = keep control blocks in real memory	True for MS/PC-DOS False for all other systems
S\$default_term	Up to 16 letters	Default terminal table name	N/A
S\$def_video	(See 'Explanation')	INVerse, BRlght, UNderline, BLInk or bit value	INV
Sdisplay	Up to 16 letters	Display table name	N/A
S\$double_width	33	Permits use of 16-bit characters with, for example, Kanji	N/A
SGUI	GUI driver path	Which Graphical User Interface (GUI) to use. Can be tested in Proc	Installation default
Skeyboard	Up to 16 letters	Keyboard table name	N/A
Slanguage	Up to 3 letters	Language	USA
Smaxcursorsora	Number	Maximum number of cursors in ORACLE	46
Smaxfiles	Number	Maximum number of files simultaneously open	O/S-dependent
Smaxprin	Number (Kbytes)	Maximum size primary page swap area	1500 pages
Smax_que	Number (Kbytes)	Maximum size of input queue	512 bytes
S\$menu_bar	(See <i>Reference Guide</i>)	Where and how pulldown menu bar appears	Top, INV, BRI
Sno_async	None	Disable asynchronous terminal I/O under VMS	Enabled
Sno_busy	None	Disable *busy* sign	Enabled
Sno_link	None	Linking disabled (MS/PC-DOS only)	Disabled
Snot_empty	None	Check mandatory fields according to pre-V4 methods	Use post-V4 values
Soldtimer	True or false	True = stick to pre-V5 date and time definitions. False = use V5 date and time definitions	False
Sremote_path	Login information	This is for the PolyServer only	N/A
Stwo_phase_commit	None	Enable two-phase commit (only works with DBMSs which support this feature)	Disabled
Svariation	Up to 16 letters	Variation	USYS
Sworkfile	File specification	Name of secondary work file (page swap file) (MS/PC-DOS)	

table 12-2 System settings in assignment files.

12.7 Extensions used for UNIFACE run time and other files

File name	Meaning
*.aps	Compiled application screens.
*.frm	Compiled external schemas.
*.prt	Print files (VAX/VMS).
*.pn	Print files (all other systems) (<i>n</i> = sequential number).
*.exp	Export files.
*.trx	TRX files.
any	Any.

table 12-3 UNIFACE files, non-DBMS.

Example

```
*.aps      /home/central_park/applics/*.aps
*.prt      /home/central_park/applics/*.prt
file_name  /home/central_park/textfiles/another_file
```

In this way, whenever a reference is made to `file_name`, UNIFACE substitutes `another_file`.

```
file_load "file_name",textfield
```

The following file is actually loaded into TEXTFIELD:

```
/home/central_park/textfiles/another_file
```

12.8 PolyServer assignments

Assignment files are flat ASCII files which let you set UNIFACE variables, enable certain UNIFACE system settings and tell UNIFACE where to find data. These files can be edited with any text processor, or the **Text file editor** in the IDF. They are particularly important when working with PolyServer, as they are how UNIFACE knows which network driver to use and how to log on to remote machines.

Note: This section describes how to use assignment files specifically with networking. Details of the other possibilities and specific syntax used in assignment files are described in the Reference Guide, chapter 12 Assignments. If you have not already done so, make sure that you read this chapter of the Reference Guide.

An assignment causes UNIFACE to access the PolyServer on another node by creating a path to a network driver, instead of to a DBMS driver. See figure 12-2:

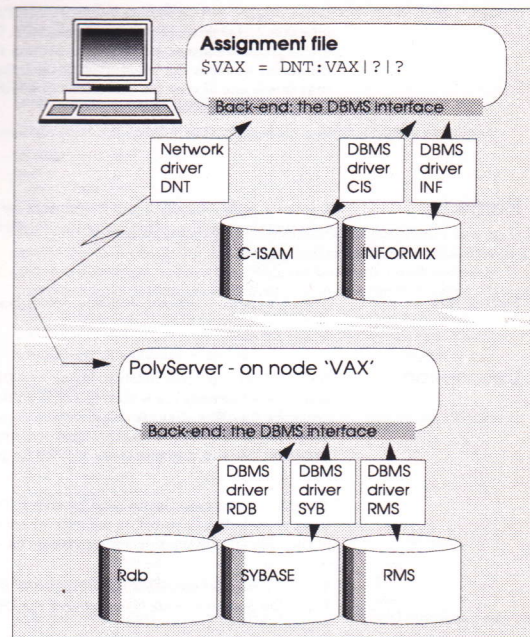


figure 12-2 Assignment paths to network drivers.



12.8.1 Providing login information with \$remote_path

\$remote_path is a UNIFACE and PolyServer system setting assignment.

When logging on to a remote machine, UNIFACE passes the login information for path, node, id and password. This comes from the client, and is therefore a reaction to a request from the server; in other words, the server becomes the 'master' until the information is provided. Note that this is one of very few situations in which the PolyServer asks UNIFACE for information.

You can supply this information with the \$remote_path assignment.

Name

\$remote_path - specify login information for a DBMS or network login on a remote machine, when using the PolyServer.

Synopsis

\$remote_path = driver:database|user|password

Description

If a PolyServer process needs to log on to a DBMS or another machine and does not already know the login information, it sends a request to the client for this. The client first looks for an assignment beginning with \$remote_ and ending with the requested path name. If this is not available, the client presents the **DBMS Logon** form for the user to fill in.

The *driver* parameter is not used by either UNIFACE or PolyServer. It must be included, however, to indicate where the database or node name begins. This is the three-letter mnemonic for the driver, followed by a colon (:).

You must provide complete login information with this assignment. Using the question mark (?) to request the DBMS Logon form is not supported with \$remote.

Examples

\$remote_london = syb:pubs|chertsey|park

The following assignment first creates a path named \$vax2 which uses the DNT network driver to access a remote machine. This assignment includes user name and login information. The next two lines assign entities in the DEMO conceptual schema, and the DICT conceptual schema to this path. The last line provides login information for the ORACLE database on the server machine:

```
$vax2 = dnt:vax2|myname|mypass
*.demo = $vax2:*.
*.dict = $vax2:*.
$remote_apdict = ora:|bickers|island
```

PSV.ASN in the login directory on the server machine is as follows:

```
$oradem = ora:|scott|tiger
$apdict = ora:|??
*.dict = $apdict:*.
*.demo = $oradem:*.

```

The PSV.ASN assignment file on the server machine creates two paths, one for the demo data and one for application dictionary data. The first path includes login information. Because the second path includes question marks, it requires information from the client in order to log on.

When PolyServer tries to access information via the path \$apdict, it goes back to the client application for the login information. This information is available with the \$remote_apdict assignment.

Incorrect usage

The following example is *incorrect*, because this makes the PolyServer try to log in using '?' as the user's password:

```
$vax2 = dnt:vax2|myname|mypass
*.demo = $vax2:?.
*.dict = $vax2:?.
$remote_apdict = ora:|bickers|?
```

12.8.2 Assigning entities to network drivers

Assigning entities to a network driver is done in exactly the same way an entity is assigned to a DBMS driver when running stand-alone. The *only* difference is that the assignment references a network driver instead of a DBMS driver (the syntax is identical).

The assignment file used by the UNIFACE application on the client side has to do two things: create a path which accesses the network driver and assign one or more entities to this path. These are discussed below.

Create a path which accesses the network driver

The path definition can optionally include node, user name and password information. If a question mark (?) is included in place of this information, PolyServer makes the Logon form appear to ask the user for the required information.

Assign one or more entities to this path

After the path has been created, assign the entities located on the server to this path.

Example

The following assignment file contains assignments for the data used in the demo application delivered with UNIFACE. The first two assignments create paths named \$vax2 and \$vax3. Both of these paths are accessed with the DECnet network driver.

```

;DEMO.asn Assignment file
$vac3      dnt:vax3|?|?
$vac2      dnt:vax2|?|?
visits.rbase $vac3:visits.*
*.rbase     $vac2:*.*
```

The assignments for these two paths include only the node name. The question marks appearing in the position of the user name and password mean that the user will be asked for this information when needed.

The next line assigns the VISITS entity from the conceptual schema RBASE to the \$vax3 path. The line after that assigns all other entities in this conceptual schema to the \$vax2 path.

When the user retrieves data, the Logon form appears, asking for user name and login information needed to access the node vax2. After logging onto vax2, the Logon form will appear again to ask for user name and login for vax3.

This assignment file can be used on any client platform where the DECnet driver is available. The syntax does not change. The syntax is also the same when using another network driver, for example Named Pipes or TCP/IP. The only difference is that TCP or NMP is substituted for DNT.

Password

Be aware that problems may be encountered if your password is longer than the length recognized by the operating system. For example, many UNIX systems only recognize the first eight characters and ignore the rest.

If your password is longer than the recognized operating system length, this mechanism of entering your password does not automatically truncate to the operating system limit, because the mechanism has been designed as an open system. A workaround is to enter only the first eight characters, or whatever the limit of the system is.

12.8.3 Hierarchy of assignment files

The following assignment files are possible:

On the UNIFACE client machine

1. *Either*, as shown in figure 12-3:
The assignment file specified at application start-up with the /asn switch.
Or:
The assignment file specified in the application definition.
Or:
application_name.ASN.
application_name.ASN.
2. USYS.ASN - the global defaults for all UNIFACE applications.

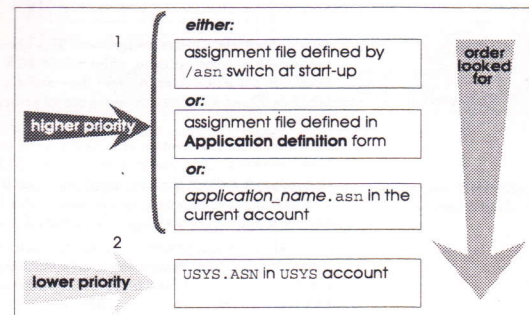


figure 12-3 Hierarchy of reading and priorities: UNIFACE assignment files.

On the PolyServer machine

1. *Either*, as shown in figure 12-4:
The assignment file defined for the PSV process with the /asn switch.
Or:
PSV.ASN - the specific file for the current PolyServer session.
2. PSYS.ASN - the global defaults for all PolyServer sessions.

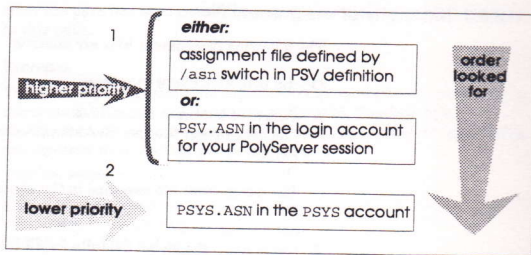


figure 12-4 Hierarchy of reading and priorities: PolyServer assignment files.

We deliberately list the assignment files above in numbered steps, because this is the order in which UNIFACE and PolyServer read and prioritize the assignments these files contain. In the first step for UNIFACE and PolyServer, the order of priority is also from top to bottom.

Be aware that this system effectively gives you two assignment 'environments'. If an assignment on the UNIFACE side assigns entities to a network driver, then an assignment on the PolyServer can reassign this assignment. As such, we can talk of the PolyServer assignments as having a higher priority than the UNIFACE ones.

Generally, the assignment files on the client machine determine which network driver and system login information should be used. The server machine assignment files, on the other hand, contain DBMS assignments and login information.

Separate hierarchies let you provide definitions at the appropriate place. For example, you probably do not want end users to know DBMS passwords on the server machine, as this might allow unauthorized entry. Include these in an assignment on the server machine.

Relationships between assignment files

The diagram in figure 12-5 shows how the various assignment files work together. This configuration has two client machines, each with two different users, using the same application and assignment files but starting in separate directories.

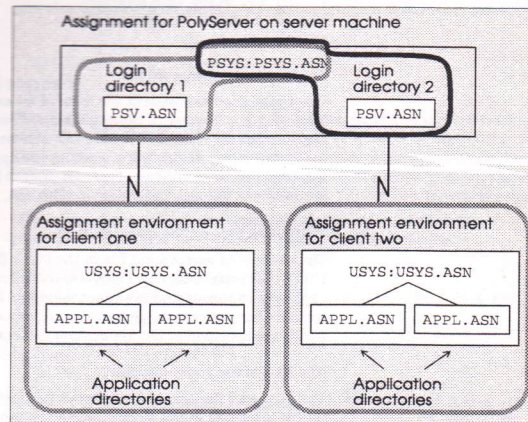


figure 12-5 Combination of assignment files.

Both of these client machines access the server machine via their own different login directories. There is one USYS directory on each client machine, and one PSYS directory on the server machine.

12.8.4 Kinds of assignment files

As explained at the beginning of this section, assignment files on the client machine usually determine which network driver and system login

information should be used; assignment files on the server machine usually contain DBMS assignments and login information.

This subsection explains:

- Application assignment file.
- USYS:USYS.ASN on the client machine.
- PSV.ASN in the login directory.
- /asn switch on PolyServer side.
- PSYS:PSYS.ASN.

Application assignment file

The application assignment file is valid for each application session. Typically, it is located in the application directory and has the same name as the application. This file can also be specified with the /asn=*file_name* switch when starting the application, or in an application level definition. This assignment file contains settings which are intended for each individual application.

USYS:USYS.ASN on the client machine

The USYS.ASN assignment file in the USYS directory (also called the UNIFACE installation directory) on the client machine is valid for all applications running on the client machine. It generally includes system-wide settings as opposed to individual application assignments, which are specified in the application assignment file.

PSV.ASN in the login directory

The PSV.ASN file is analogous to the application assignment file, except that it is located in the login directory on the server machine. This assignment file determines the assignments for all clients which use that login directory.

/asn switch on PolyServer side

Instead of PSV.ASN, you can use another assignment file by using the /asn=*file_name* switch when starting the PolyServer. If you use the /asn=*file_name* switch, you should include it in the definition of the 'PSV' process. How to do this differs per system.

For example, in a UNIX and TCP/IP environment, you define the PSV process as PSV=" \$PSV TCP: " (after setting the environment variables required to run PolyServer by running the inspoly script). Defining PSV this way, without the /asn switch, causes the PolyServer to use the PSV.ASN assignment file in the login directory, if it exists. Adjusting the

PSV definition to read PSV=" \$PSV /asn=*file_name* TCP: " causes the PolyServer to use the assignment file *file_name* instead of PSV.ASN.

PSYS:PSYS.ASN on the server machine

This assignment file is analogous to the USYS.ASN on the client machine, except that it is located in the PSYS directory on the server machine. It provides definitions for all PolyServers running on that server machine.

12.8.5 Priorities and scope

The assignments for client and server remain strictly separated from each other: the PolyServer's assignments take effect only when data reaches the server from the client. For example, an assignment on the server machine might reassign a \$path which has come from the client to another \$path, and no assignment on the client side can override this.

Within the client and server environments, however, strict rules of priority are applied to the various assignment files available.

UNIFACE client

UNIFACE reads the assignments into an internal table from each file in the order shown below:

1. Application assignment file.
2. USYS:USYS.ASN.

When UNIFACE looks for an item which could be assigned, it scans the internal table from top to bottom until it finds a match. Therefore, the assignments defined in the application assignment file have the highest priority, and those in USYS:USYS.ASN have the lowest.

PolyServer

In the same way, PolyServer reads the server's assignments into an internal table from each file in the order shown below:

1. PSV.ASN, or file specified with /asn switch.
2. PSYS:PSYS.ASN.

When the PolyServer looks for an item which could be assigned (and which has come from the UNIFACE client or another PolyServer), it scans the internal table from top to bottom until it finds a match. Therefore, the assignments defined in PSV.ASN have a higher priority, and those in PSYS:PSYS.ASN have a lower priority.

Chapter 13 Function codes

Mnemonic	Numeric code	Explanation
	^009	Tab
	^010	Line feed
	^012	Form feed
	^013	Carriage return
	^034	Double quotation marks (")
ACCEPT	^127^009	
ADD_OCC	^127^044	
ATTRIBUTE	^127^078	Define character attributes
BOLD	^127^147	
BOTTOM	^127^023	Cursor at window bottom
BOT_OF_FORM	^127^021	Cursor at form bottom
CHAR	^255^001	
CLEAR	^127^012	
COMPOSE	^127^088	Compose character
CURSOR_DOWN	^127^017	
CURSOR_FAST_DOWN	^127^026	Cursor eight lines down
CURSOR_FAST_LEFT	^127^027	Cursor eight spaces left
CURSOR_FAST_RIGHT	^127^028	Cursor eight spaces right
CURSOR_FAST_UP	^127^025	Cursor eight lines up
CURSOR_LEFT	^127^018	
CURSOR_RIGHT	^127^019	
CURSOR_UP	^127^016	

table 13-1 continues

Mnemonic	Numeric code	Explanation
DETAIL	^127^094	
ERASE	^127^008	
FIELD	^255^010	
FIND_TEXT	^127^150	
FIRST	^255^067	
FIRST_OCC	^127^037	
FIRST_TEXT	^127^129	
FONT	^127^151	
FRAME	^127^089	Define frame
HELP	^127^092	
HOME	^127^022	Cursor at window top
INSERT	^255^071	
INSERT	^255^074	Insert (removed)
INS_CHAR	^127^184	
INS_FIELD	^127^181	
INS_FILE	^127^180	Insert file
INS_LINE	^127^182	
INS_OCC	^127^043	
INS_OVER	^127^146	Insert/Overstrike
INS_SELECT	^127^195	Insert selected block
INS_TEXT	^127^177	
INS_WORD	^127^183	
ITALIC	^127^148	
KEY_HELP	^127^072	Keyboard layout help
LAST	^255^068	
LAST_OCC	^127^038	
LAST_TEXT	^127^128	
LINE	^255^004	
MENU	^127^101	
MESSAGE	^127^093	Message frame
NEXT	^255^065	Next mode
NEXT_CHAR	^127^142	
NEXT_FIELD	^127^046	
NEXT_LINE	^127^136	
NEXT_OCC	^127^039	
NEXT_TEXT	^127^163	
NEXT_WORD	^127^140	
OCCURRENCE	^255^011	

table 13-1 continues

Mnemonic	Numeric code	Explanation
OCC_WINDOW	^255^015	
PREV	^255^066	Previous mode
PREV_CHAR	^127^143	
PREV_FIELD	^127^047	
PREV_LINE	^127^137	
PREV_OCC	^127^040	
PREV_TEXT	^127^162	
PREV_WORD	^127^141	
PRINT	^127^098	
PRINT_ATTRIBUTES	^127^099	
PROFILE	^127^087	Define find profile
PULLDOWN	^127^086	
QUICK_ZOOM	^127^096	
QUIT	^127^010	
REFRESH	^127^067	
REMOVE	^255^073	
REM_CHAR	^127^172	Delete character
REM_FIELD	^127^166	
REM_FILE	^127^192	Remove file
REM_LINE	^127^167	
REM_OCC	^127^045	
REM_SELECT	^127^194	Remove selected block
REM_WORD	^127^169	
RESET_SELECT	^127^196	
RETRIEVE	^127^005	
RETRIEVE_SEQ	^127^003	
RUB_CHAR	^127^173	Backspace (delete character to left of cursor)
RULER	^127^081	Define ruler
SAVE	^127^179	Put selected text in \$selblk buffer
SELECT	^127^193	
SQL	^127^097	
STORE	^127^011	
SWITCH_KEY	^127^100	Switch to alternate keyboard

table 13-1 continues

Mnemonic	Numeric code	Explanation
TEXT	^255^009	
TEXT_WINDOW	^255^014	
TOP_OF_FORM	^127^020	Cursor at form top
UNDERLINE	^127^149	
USER_KEY	^127^091	
VIEW	^127^073	
WORD	^255^003	
ZOOM	^127^095	

table 13-1 Character codes for use in macro statements.

**Quick Reference Guide
Volume 11
101075201
21 September 1992**

unifAce

Advanced Software Technology

Uniface International

**Hogehilweg 16, 1101 CD Amsterdam, The Netherlands
Telephone +31(0)20-6976644, FAX +31(0)20-6912912**