Quick Reference Guide
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
Hogeheiweg 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
### 4 Debugging Proc.

#### 4.1 Command line

4.1.1 `break (module) (line), b (module) (line)`

4.1.2 `break, b`

4.1.3 `call on`.

4.1.4 `call off`.

4.1.5 `clrmess off`.

4.1.6 `clrmess on`.

4.1.7 `done`.

4.1.8 `dump`.

4.1.9 `dump module`.

4.1.10 `examine [number], ex [number]`

4.1.11 `examine, ex`

4.1.12 `ex (field_name) (entity_name)`.

4.1.13 `ex number = (value)`.

4.1.14 `ex (function(name))`.

4.1.15 `go, g`.

4.1.16 `iprint (number), io (number)`.

4.1.17 `line [number], l [number]`.

#### 4.2 Commands

4.2.18 `nop`.

4.2.19 `putmess off`.

4.2.20 `putmess on`.

4.2.21 `quit`.

4.2.22 `return on`.

4.2.23 `return off`.

4.2.24 `show, sh`

4.2.25 `trace on, tr on`.

4.2.26 `trace off, tr off`.

4.2.27 `trace 1, tr 1`.

4.2.28 `trace 0, tr 0`.

4.2.29 `step, s`.

4.2.30 `step (number), s (number)`.

4.2.31 `xtrace`.

4.2.32 `Examining contents of Sregisters`.

4.3 Trigger mnemonics.

### 5 Naming conventions, reserved words and wildcards

#### 5.1 Naming conventions

5.1.1 `External schema, conceptual schema, application, field`

5.1.2 `Frames on paint tableau`

5.1.3 `Global models`.

#### 5.2 Reserved words

5.2.1 `IDF application dictionary names`.

5.2.2 `UNIFACE Reporter application dictionary names`.

#### 5.3 Wildcards

### 6 Interface definition

#### 6.1 Data types (UNIFACE)

6.2 `UNIFACE packing codes`
7 Syntax checks ........................................ 7-1

7.1 Entry format ........................................ 7-1
  7.1.1 Entry format examples .......................... 7-2

7.2 Display/Edit/Prompt ................................. 7-3

7.3 Characters allowed in a field ..................... 7-4

7.4 Shorthand codes for Field syntax model ........ 7-4

8 Display format ....................................... 8-1

8.1 String .............................................. 8-1

8.2 Numeric (and float) ............................... 8-2

8.3 Date ................................................ 8-4

8.4 Time ................................................. 8-6

8.5 Combined date and time ............................ 8-7

8.6 Shorthand codes for Field layout model ....... 8-7

9 Video and color ....................................... 9-1

9.1 Video attributes .................................. 9-1

9.2 Color definition ................................... 9-2

10 Keyboard layouts .................................... 10-1

10.1 Bull TWS 2103 ...................................... 10-2

10.2 Data General FKB4700 ............................ 10-3

10.3 DEC VT100/200 .................................... 10-4

10.4 Hewlett Packard HP-HIL ......................... 10-5

10.5 IBM PC AT 83/84 key .............................. 10-6

10.6 IBM AT 101/102 key enhanced ..................... 10-7

10.7 IBM RS6000 console ............................... 10-8

10.8 IBM 6150 RT PC console .......................... 10-9

10.9 OS/2 .............................................. 10-10

10.10 SCO-Unix ........................................ 10-11

10.11 SCO-Xenix ........................................ 10-12

10.12 Siemens 97801 .................................. 10-13

10.13 Stratus v102 ...................................... 10-14

10.14 Sun-3 ............................................. 10-15

10.15 Sun-4 SPARC station ............................. 10-16

10.16 Function key combinations (all keyboards) .... 10-17

10.17 'Super' key combinations ........................ 10-18

11 IDF command switches .............................. 11-1

11.1 Switches .......................................... 11-2

11.2 Sub-switches ...................................... 11-3

12 Assignments .......................................... 12-1

12.1 Priorities ......................................... 12-1

12.2 Syntax ............................................. 12-2

12.3 Entity assignments ............................... 12-3

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
Chapter 1  Proc statements

Name  addmonths - add the specified number of months to the date.

Synopsis  addmonths amount, "date", "start_date"

Return Value  The resulting date is stored in $result. The date type returned depends on the date type given as an argument. If it is given as a literal string, it is always returned as a combined date and time field, with the time part set to 0. If, however, a central or local register is used as a parameter, the date type in $result depends on the type of the central or local register.

Name  apexit - exit the application immediately.

Synopsis  apexit

Return Value  None.
Name

askmess - display a message and wait for the user response.

Synopsis

askmess[/no beep] *message* 1,*reply_1,*...,*reply_n* 1(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 'Y' (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 [spath]

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  clrmessage - clear the message frame of text.

Synopsis  clrmessage

Return Value  None.

Name  commit - commit a transaction to the database.

Synopsis  commit [dbms | $path]

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.

Name  compare - compare fields of two adjacent occurrences.

Synopsis  compare [/next[/previous] {field1,field2,...,fieldn}) from 'entity'

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 fields could not be accessed. This can occur when entity 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] destination[/init] = expression | constant

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 | 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 | debug - start the interactive debugger.

Synopsis | debug

Return Value | None.

Name | delete - delete an occurrence from the database.

Synopsis | delete
**Return Value**

**Name**

- **done** - exit from a Proc.

**Synopsis**

- **done**

**Return Value**

$\text{status}$ remains unchanged.

**Name**

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

**Synopsis**

- **edit[/menu[/nowander] [field]]**

**Return Value**

$\text{status}$ 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 $\text{batch}$ to avoid this.

**Name**

- **eject** - eject a page during printing.

**Synopsis**

- **eject**

**Return Value**

$\text{status}$ remains unchanged.

**Return Value**

$\text{status}$ is set to the following values:

- **0** - Any other situation.

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

**Name**

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

**Synopsis**

- **else (Proc_statement)**

**Return Value**

None.

**Name**

- **end** - mark the ending of a Proc.

**Synopsis**

- **end**

**Return Value**

$\text{status}$ 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 I/O error on write request.
- -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.
### field_syntax

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

**Synopsis**
field_syntax "field", "attribute_1 [ , ..., attribute_n]"

**Return Value**
None.

### field_video

**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.

### file_load

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

**Synopsis**
file_load "[path]/file", destination

**Return Value**
Status 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.

### goto

**Name**
goto - unconditional branch to the specified label.

**Synopsis**
goto label

**Return Value**
None.

### file_dump

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

**Synopsis**
file_dump[/append] field, "[path]/file"

**Return Value**
None.

### help

**Name**
help - display the specified message in a help box and wait for the user response.

**Synopsis**

**Return Value**
Status is set by the help statement.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>With ^ACCEPT.</td>
<td>1</td>
</tr>
<tr>
<td>With ^QUIT.</td>
<td>0</td>
</tr>
<tr>
<td>If the help file USYS:USYSTXT could not be found.</td>
<td>-1</td>
</tr>
<tr>
<td>If the field does not exist.</td>
<td>-1</td>
</tr>
<tr>
<td>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).</td>
<td>-2</td>
</tr>
</tbody>
</table>

### Name

- `if` - start of an if/else block.
- `if (condition)
  statements
  [else
   statements]
endif`

### Synopsis

- `length string`
- `lock`

### Return Value

- `result is set to the number of characters in the string.`
- `status is set by the lock statement.`
- `result is always returned when the form is being prototyped.`
- `result is returned when the occurrence cannot be modified (for example, during a run/query).`
- `result is returned if there is no active occurrence.`
- `result is returned if the occurrence has been removed since it was retrieved.`
- `result is returned if the hit for the occurrence does not exist.`
- `result is returned when there is no hit for the occurrence and message 2008 - Occurrence cannot be modified due to fetch error is displayed.`
- `result 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.`
- `result is returned to indicate the occurrence has been modified or removed since it was retrieved, and a reload should be executed.`
- `result is returned if the occurrence is already locked.`

### Name

- `/init - initialize a field without changing the status of Soccmod, $formmod or $fieldmod.`

### Synopsis

- `field/init = value`

### Return Value

- `None.`

---

Quick Reference Guide (10107521, 21 September 1992)
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[/nobeep] "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.
**Summary**

**numset** - set the specified counter to a new value.

**Synopsis**

numset "counter", init_value {, "library"}

**Return Value**

$\text{status}$ is set to 0 on success, -1 otherwise.

---

**open** - open a database for access.

**Synopsis**

open "parameters", "path"

**Return Value**

If the open operation fails for any reason, $\text{status}$ 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.

---

**perform** - call the specified 3GL function.

**Synopsis**

perform[/noterm] "function"

**Return Value**

$\text{status}$ is set to the value returned by $\text{function}$, or -1 if $\text{function}$ could not be found. Consequently, do not return -1 in a 3GL function, as this is indistinguishable from UNIFACE not being able to find $\text{function}$.

---

**print** - activate printing, optionally using a print model.

**Synopsis**

print[/ask] {"printer_model"}, {"print_option"}

**Return Value**

$\text{status}$ is set by the print statement.

- 0: On success.
- 1: Printing is already being performed ($\text{printing}$ is 1).
- 1: ^QUIT was used in the Print Attribute form.
- 1: An invalid print_option 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 $\text{result}$.

---

**print_break** - print the specified break frame.

**Synopsis**

print_break "frame_name"

**Return Value**

$\text{status}$ is set by the print_break statement.

- 1: When not printing or inside a header or footer.
- 1: When the Proc code in the OCCURRENCE BECOMES ACTIVE trigger for the break frame returns a negative value.
- 0: If the Proc code in the OCCURRENCE BECOMES ACTIVE trigger returns a positive value.
UNIFACE V5.2

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

Synopsis pulldown[/load] [*menu_bar_name*]

Return Value $status is set by the pulldown and pulldown/load statements:
-1 If the pulldown menu does not exist.
0 If the OPTION trigger of the selected pulldown menu item is empty.
Otherwise $status is set to the value returned by the Proc code in the OPTION trigger of the selected pulldown 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*\} \{order by *field [desc] [...]\}
**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:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Function completed successfully.</td>
</tr>
<tr>
<td>-2</td>
<td>Occurrence not found.</td>
</tr>
<tr>
<td>-3</td>
<td>Hardware or software error.</td>
</tr>
<tr>
<td>-11</td>
<td>Occurrence currently locked.</td>
</tr>
<tr>
<td>-16</td>
<td>Network error.</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th><code>sequence_number</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Of the removed occurrence.</td>
</tr>
<tr>
<td></td>
<td>If the occurrence could not be removed.</td>
</tr>
</tbody>
</table>

**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 modificable functions are `$formmod`, `$fieldcheck` and `$occccheck`.

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

**Synopsis**
`retrievel[!e "entity*"]!/o ["entity*"] ["wildcard_character"]`

**Return Value**
`@status` is set by the retrieve statement. Common values returned include:
No data to retrieve.

The occurrence was found in the external schema. The current occurrence is removed and the cursor repositioned on the found occurrence.

The occurrence was found among the removed occurrences; it was un-removed.

The entity is painted as a foreign entity and one hit was found in the database.

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.

Success.

Unexpected end of file encountered.

The entity is painted as a foreign entity and the key value was not found during the database lookup.

Exceptional I/O error.

Open request for the file or table failed.

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).

Occurrence currently locked.

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.

DBMS network error.

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

return [(expression)]

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

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

rollback idbms | $path

$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.

run - activate the specified external schema.


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  *schema* could not be found.
0   The schema did not contain an **edit** or **display**
    statement in the **EXECUTE** trigger.
9   The user left *schema* with ^ACCEPT.
10  The user left *schema* 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:

$\text{result} > 0$   Starting position of the match.
$\text{result} = 0$   Profile not found.
$\text{result} = 0$   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"
{where clause} to destinations

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

**Return Value**

**set** - set the specified $\text{function}$ to 1.

**Synopsis**

**set** $\text{function}$

**Return Value**

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

**Name**

**setocc** - make a specific occurrence the current occurrence.

**Synopsis**

**setocc** "entity", **sequence_number**

**Return Value**

$\text{status}$ is set by the **setocc** statement. One of the following values are
returned:

**sequence_number**   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 [expression]

**Return Value**

$\text{status}$ is set to -1 if UNIFACE is not printing. $\text{status}$ is set to 0 on success, or if the statement is ignored.

**Name**

spawn - execute the specified command, using the operating system.

**Synopsis**

spawn "command"

**Return Value**

$\text{status}$ is -1 if a null command (""") has been given as an argument. Otherwise, spawn returns either an operating system code or the command return code.

**Name**

sql - pass a SQL statement to the specified DBMS.

**Synopsis**

sql "statement", "path"

**Return Value**

$\text{status}$ is set to the number of hits, $\text{result}$ 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 path 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 {"entity"}]

**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 $dberror.

Name: while - mark the start of a while/endlwhile block.

Synopsis: while (expression)
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:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>This trigger is the best place for the statement.</td>
</tr>
<tr>
<td>-</td>
<td>You should not use the statement in the trigger.</td>
</tr>
<tr>
<td>?</td>
<td>You can use the statement in the trigger, but there is usually a better place for it.</td>
</tr>
<tr>
<td>•</td>
<td>You can use the statement in the trigger, but correct usage depends heavily on what you want to do with it; caution is recommended.</td>
</tr>
</tbody>
</table>

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.
Chapter 2 Special functions

Name  $appname - return the name of the application.

Synopsis  $appname

Return Value  The $appname 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  $batch

Return Value  $batch returns the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>UNIFACE is not in a batch process.</td>
</tr>
<tr>
<td>1</td>
<td>UNIFACE is in a batch process.</td>
</tr>
</tbody>
</table>
**UNIFACE V5.2**

### Name

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

### Synopsis

$$\text{field} = \text{register} \rightarrow \text{char}$$

### Return Value

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

### Name

$\text{clock}$ - return the system time or convert the argument to the time data type.

### Synopsis

$$\text{field} = \text{register} \rightarrow \text{clock} \{\text{source}\}$$

### 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

$\text{currhits}$ - return the number of occurrences in the hitlist.

### Synopsis

$$\text{field} = \text{register} \rightarrow \text{currhits} \{\text{entity}\}$$

### 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

$\text{curocc}$ - return the sequence number of the current occurrence in the hitlist.

### Synopsis

$$\text{field} = \text{register} \rightarrow \text{curocc} \{\text{entity}\}$$

### Return Value

$\text{curocc}$ returns the sequence number in the hitlist of the current occurrence, or -1 if `entity` does not exist. The following statements and triggers affect the value of $\text{curocc}$:

- `<NEXT><OCCURRENCE>` sets $\text{curocc}$.
- `<PREVIOUS><OCCURRENCE>` sets $\text{curocc}$.
- `<ADD><OCCURRENCE>` modifies $\text{curocc}$.
- `<INSERT><OCCURRENCE>` modifies $\text{curocc}$.
- `<REMOVE><OCCURRENCE>` resets $\text{curocc}$ to 1.

### Name

$\text{date}$ - return the current date or convert a date string into the date data type.

### Synopsis

$$\text{field} = \text{register} \rightarrow \text{date} \{\text{source}\}$$

### Return Value

The value of `source` is returned as a date data type. If `source` is omitted, $\text{date}$ returns the current system date.
**UNIFACE V5.2**

### Name

$\texttt{datetime}$ - return the system date and time, or convert the argument to the date and time data type.

### Synopsis

(field = | register =) $\texttt{datetime} \{\texttt{source}\}$

### Return Value

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

### Name

$\texttt{dberror}$ - return the specific DBMS error code.

### Synopsis

(field = | register =) $\texttt{dberror}$

### Return Value

The value $\texttt{dberror}$ 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

$\texttt{dbocc}$ - return the sequence number of the current occurrence in the database.

### Synopsis

(field = | register =) $\texttt{dbocc} \{\texttt{entity}\}$

### Return Value

The following values can be returned by $\texttt{dbocc}$:

- sequence_number in the database of the current or specified entity.

### Name

$\texttt{direction}$ - return the structure editor mode (NEXT or PREVIOUS).

### Synopsis

(field = | register =) $\texttt{direction}$

### Return Value

One of the following values is returned:

- 0 NEXT mode.
- 1 PREVIOUS mode.

### Name

$\texttt{display}$ - return the name of the current display device table.

### Synopsis

(field = | register =) $\texttt{display}$

### Return Value

The value returned by $\texttt{display}$ 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:

- There are no occurrences of entity or named_area_frame containing data, and the frame definition of entity or named_area_frame has Supp. on Empty set to 'Y' (Yes).
- The entity or named_area_frame contains at least one occurrence with data in.
- The entity or named_area_frame does not exist (usually due to a spelling mistake by you).

**Name**

$entity - return the name of the current entity.

**Synopsis**

'field = $register $entity'

**Return Value**

$entity 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:

<table>
<thead>
<tr>
<th>Error code</th>
<th>Default message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0102</td>
<td>Not enough occurrences in entity entity.</td>
</tr>
<tr>
<td>0103</td>
<td>Too many occurrences in entity entity.</td>
</tr>
<tr>
<td>0118</td>
<td>More occurrences are not allowed.</td>
</tr>
<tr>
<td>0139</td>
<td>Entity entity still has restricted links to entity.</td>
</tr>
<tr>
<td>0148</td>
<td>First occurrence.</td>
</tr>
<tr>
<td>0149</td>
<td>Last occurrence.</td>
</tr>
<tr>
<td>0204</td>
<td>No modifications allowed on occurrence of this entity.</td>
</tr>
<tr>
<td>0206</td>
<td>Occurrence locked.</td>
</tr>
<tr>
<td>0212</td>
<td>Occurrence in external schema does not match database occurrence.</td>
</tr>
<tr>
<td>0213</td>
<td>Occurrence no longer exists.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error code</th>
<th>Default message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0105</td>
<td>Not allowed to change primary/candidate key field.</td>
</tr>
<tr>
<td>0120</td>
<td>Error on field &quot;field&quot;; subfield too large.</td>
</tr>
<tr>
<td>0121</td>
<td>Error on field &quot;field&quot;; subfield too small.</td>
</tr>
<tr>
<td>0122</td>
<td>Error on field &quot;field&quot;; incorrect check digit.</td>
</tr>
<tr>
<td>0123</td>
<td>Error on field &quot;field&quot;; illegal format for numeric field.</td>
</tr>
</tbody>
</table>

*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:

<table>
<thead>
<tr>
<th>Error code</th>
<th>Default message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0020</td>
<td>Not allowed to change primary/candidate key field.</td>
</tr>
<tr>
<td>0021</td>
<td>Error on field &quot;field&quot;; subfield too large.</td>
</tr>
<tr>
<td>0022</td>
<td>Error on field &quot;field&quot;; subfield too small.</td>
</tr>
<tr>
<td>0023</td>
<td>Error on field &quot;field&quot;; incorrect check digit.</td>
</tr>
<tr>
<td>0024</td>
<td>Error on field &quot;field&quot;; illegal format for numeric field.</td>
</tr>
</tbody>
</table>

*table 2-2 continues*
<table>
<thead>
<tr>
<th>Error code</th>
<th>Default message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0124</td>
<td>Error on field &quot;field&quot;; illegal format for date field.</td>
</tr>
<tr>
<td>0125</td>
<td>Error on field &quot;field&quot;; illegal format for time field.</td>
</tr>
<tr>
<td>0126</td>
<td>Error on field &quot;field&quot;; illegal syntax format.</td>
</tr>
<tr>
<td>0127</td>
<td>Error on field &quot;field&quot;; illegal entry format.</td>
</tr>
<tr>
<td>0128</td>
<td>Error on field &quot;field&quot;; subfield too large to check.</td>
</tr>
<tr>
<td>0129</td>
<td>Error on field &quot;field&quot;; subfield(s) are required.</td>
</tr>
<tr>
<td>0130</td>
<td>Error on field &quot;field&quot;; too many subfields specified.</td>
</tr>
<tr>
<td>0131</td>
<td>Error on field &quot;field&quot;; font not allowed.</td>
</tr>
<tr>
<td>0133</td>
<td>Error on field &quot;field&quot;; ruler/frames not allowed.</td>
</tr>
<tr>
<td>0134</td>
<td>Error on field &quot;field&quot;; italic not allowed.</td>
</tr>
<tr>
<td>0135</td>
<td>Error on field &quot;field&quot;; underline not allowed.</td>
</tr>
<tr>
<td>0136</td>
<td>Error on field &quot;field&quot;; bold not allowed.</td>
</tr>
<tr>
<td>0137</td>
<td>Error on field &quot;field&quot;; open/close brackets do not match.</td>
</tr>
<tr>
<td>0138</td>
<td>Error on field &quot;field&quot;; illegal format for floating field.</td>
</tr>
<tr>
<td>0150</td>
<td>Requested number of &quot;&amp;&quot; and &quot;(&quot; operators not supported.</td>
</tr>
</tbody>
</table>

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

#### Name

$@fieldendmod$ - return the modification status of a field when the field is left.

#### Synopsis

```
(field = | register =) $@fieldendmod$
```

#### Return Value

The value returned is only valid in the LEAVE FIELD trigger. It is always 1 if the programmer has used a set $@fieldcheck$ for the current field. The value can be one of:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not modified</td>
</tr>
<tr>
<td>1</td>
<td>Modified</td>
</tr>
</tbody>
</table>

#### Name

$@fieldmod$ - return the modification status of a field.

#### Synopsis

```
(field = | register =) $@fieldmod([field], [entity])$
```

#### Return Value

The following values are returned by $@fieldmod$:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not modified</td>
</tr>
<tr>
<td>1</td>
<td>Modified</td>
</tr>
<tr>
<td>-1</td>
<td>field or entity does not exist (flagged as a warning at compile-time).</td>
</tr>
</tbody>
</table>

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

- `<EXECUTE>` $@fieldmod$ is set to 0.
- `clear` $@fieldmod$ is set to 0.
- `erase` $@fieldmod$ is set to 0.
- `release` $@fieldmod$ is set to 0.
- `reload` $@fieldmod$ is set to 0.
- `retrieve` $@fieldmod$ is set to 0.
store $fieldmod is set to 0.

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

Synopsis \(\text{\textbf{field = \{register\} $\text{\textit{fieldName}}}}\)

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 \(\text{\textbf{field = \{register\} $\text{\textit{formdb}}}}\)

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.

Name $formdbmod - test if any database field has been modified.

Synopsis \(\text{\textbf{field = \{register\} $\text{\textit{formdbmod}}}}\)

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. If 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.
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 $formdb 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.

### Name

$formmod - test if the form has been modified.

### Synopsis

\[(field = | register =) \text{\$formmod}\]

### 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.
- **set**: $formmod cannot be set. Unlike reset, setting $formdb has no effect on $formdbmod.
- **store**: $formmod is reset to 0.
- **store/e**: $formmod 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.
**UNIFACE V5.2**

- **erase**
  - $\text{formmod}$ is reset to 0.

- **erase/e**
  - $\text{formmod}$ 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, $\text{formmod}$ is not reset to 0.

- **examine**
  - $\text{formmod}$ 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, $\text{formmod}$ is reset to 0 and displayed.

- **release**
  - $\text{formmod}$ is reset to 0.

- **release/e**
  - $\text{formmod}$ 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, $\text{formmod}$ is reset to 0 and displayed.

- **release/e/mod**
  - $\text{formmod}$ is set to 1.

- **release/mod**
  - $\text{formmod}$ is set to 1.

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

- **reset**
  - $\text{formmod}$ is reset to 0. For consistency, $\text{formdepthmod}$ is also reset.

- **retrieve**
  - $\text{formmod}$ 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, $\text{formmod}$ is not reset to 0. A \( ^* \text{RETRIEVE} \) causes the first outermost entity to be retrieved along with its related entities. Any unrelated entities are not automatically retrieved.

- **retrieve/e**
  - $\text{formmod}$ 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, $\text{formmod}$ is not reset to 0.

- **set**
  - $\text{formmod}$ is set to 1. Unlike set, set does not change the value of $\text{formdepthmod}$.

- **store**
  - $\text{formmod}$ is reset to 0.

- **store/e**
  - $\text{formmod}$ 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, $\text{formmod}$ is not reset to 0.

---

**UNIFACE V5.2**

- **Name**
  - \( \text{formname} \) - return the name of the form (external schema).

- **Synopsis**
  - \( \text{field = register = formname} \)

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

---

**UNIFACE V5.2**

- **Name**
  - \( \text{framedepth = depth of the painted frame.} \)

- **Synopsis**
  - \( \text{field = register = framedepth } \) \( \text{frame} \)

- **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.

---

**UNIFACE V5.2**

- **Name**
  - \( \text{sgui = mnemonic for the user interface UNIFACE is using.} \)

- **Synopsis**
  - \( \text{field = register = sgui} \)

- **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.
UNIFACE V5.2

**Name**

$\text{hits} - \text{return the number of occurrences in the hitlist.}$

**Synopsis**

$\text{[field = 1 register =] \text{\{entity\}\{entity\}\{entity\}}}$

**Return Value**

$\text{hits}$ 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 $\text{hits}$:

- **clear** $\text{hits is reset to 0.}$
- **release** $\text{hits is reset to 0.}$

**Name**

$\text{sioprint} - \text{return the type of message in the message frame.}$

**Synopsis**

$\text{[field = 1 register =] \text{\{entity\}}}$

**Return Value**

The following values are returned by $\text{sioprint}$:

- $0$: 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**

$\text{sk} - \text{set or return the current keyboard translation table.}$

**Synopsis**

$\text{[field = 1 register =] \text{\{entity\}}}$

**Return Value**

The value returned is the keyboard model currently in use.

**Name**

$\text{slanguage} - \text{set or return the current language code.}$

**Synopsis**

$\text{[field = 1 register =] \text{\{entity\}}}$

**Return Value**

$\text{slanguage}$ returns the country code currently in use.

**Name**

$\text{slines} - \text{return the number of lines left on the current (printed) page.}$

**Synopsis**

$\text{[field = 1 register =] \text{\{entity\}}}$

**Return Value**

When printing ($\text{printing is 1}$), $\text{slines}$ returns the number of lines remaining on the page, not including the header or trailer frames. When UNIFACE is not printing ($\text{printing is 0}$), the value of $\text{slines}$ is 0, and $\text{status}$ is set to -1.
**Name**

$\text{next}$ - return the value of the next occurrence of a field.

**Synopsis**

\[
\text{field} = | \text{register} = | \text{next} (\text{field})
\]

**Return Value**

The $\text{next}$ function returns the value of $\text{field}$ in the next occurrence. A NULL value will be returned when there is no next occurrence. This can be tested for with:

\[
\text{if} (\text{next} (\text{field}) = '')
\]

**Name**

$\text{number}$ - return the value of the numeric part of a string.

**Synopsis**

\[
\text{field} = | \text{register} = | \text{number} ("\text{string}")
\]

**Return Value**

$\text{number}$ returns the value of the leading numeric part it encounters of $\text{string}$. If $\text{string}$ contains no numeric text, or starts with alphabetic text, $\text{number}$ returns 0.

**Name**

$\text{occcheck}$ - require modification checks for an occurrence.

**Synopsis**

set $\text{occcheck} (\text{entity})$

**Return Value**

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

**Name**

$\text{occmod}$ - return the modification status of an occurrence.

**Synopsis**

\[
\text{field} = | \text{register} = | \text{occmod} (\text{entity})
\]

**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: entity does not exist (flagged as a warning at compile time).

$\text{occmod}$ is modified by the following statements and triggers:

- erase: $\text{occmod}$ is set to 1.
- $<\text{REMOVE OCCURRENCE}>$: $\text{occmod}$ is set to 1.

**Name**

$\text{occdepth}$ - depth of the painted occurrence.

**Synopsis**

\[
\text{field} = | \text{register} = | \text{occdepth}
\]

**Return Value**

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

The following values are returned by $occmod:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Not modified.</td>
</tr>
<tr>
<td>1</td>
<td>Modified.</td>
</tr>
<tr>
<td>-1</td>
<td>Entity does not exist or is not painted on the external schema.</td>
</tr>
</tbody>
</table>

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

**EXECUTE**
- $occmod is set to 0.
- store $occmod is set to 0.
- release $occmod is set to 0.
- retrieve $occmod is set to 0.
- clear $occmod is set to 0.
- reload $occmod is set to 0.

### Name

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

### Synopsis

**(field = 1 register = 1 $spage**

### Return Value

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

### Name

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

### Synopsis

**(field = 1 register = 1 $spassword(path**

### Return Value

$spassword 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

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

### Synopsis

**(field = 1 register = 1 $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

$spassword - return 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.

### Synopsis

**(field = 1 register = 1 $spassword(path**

### Return Value

$spassword 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

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

### Synopsis

**(field = 1 register = 1 $spassword(path**

### Return Value

$spassword 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

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

### Synopsis

**(field = 1 register = 1 $spassword(path**

### Return Value

$spassword 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

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

### Synopsis

**(field = 1 register = 1 $spassword(path**

### Return Value

$spassword 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

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

### Synopsis

**(field = 1 register = 1 $spassword(path**

### Return Value

$spassword 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

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

### Synopsis

**(field = 1 register = 1 $spassword(path**

### Return Value

$spassword 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

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

### Synopsis

**(field = 1 register = 1 $spassword(path**

### Return Value

$spassword 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

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

### Synopsis

**(field = 1 register = 1 $spassword(path**

### Return Value

$spassword 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

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

### Synopsis

**(field = 1 register = 1 $spassword(path**

### Return Value

$spassword 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.
Return Value
None.

Name
$rresult$ - the value returned by certain Proc statements.

Synopsis
$(field = \mid register = )$ $rresult$

Return Value
$rresult$ is set by many Proc statements. Refer to the documentation for the individual Proc statements for the values $rresult$ can contain.

Name
$rsetype$ - return the retrieval mode of the outermost entity.

Synopsis
$(field = \mid register = )$ $rsetype$

Return Value
The following values are returned when in a READ trigger:
78 Next occurrence.
82 Retrieve.
110 Retrieve (seq).

In the <ADD/INS. OCCURRENCE> trigger, the following additional values are returned:
65 Add occurrence.
73 Insert occurrence.

$rsetype$ is set by the following:
- read.
- ^ADD ^OCCURRENCE.
- ^INSERT ^OCCURRENCE.

Name
$ssetblk$ - return or set the contents of the current select buffer.

Synopsis
$(field = \mid register = )$ ssetblk (= field)

Return Value
ssetblk returns the current contents of the structure editor select buffer. When data is sent to ssetblk, the new data overwrites any previous contents of the buffer. Field data is sent to this buffer with the following functions:
- ^REMOVE (Selected block).
- ^SAVE (Selected block).
- ^REMOVE ^FIELD
- ^GOLD ^REMOVE ^FIELD
- Insert removed field.
- ^GOLD ^REMOVE ^TEXT
- Insert removed text.
- ^GOLD ^SAVE ^TEXT
- Unsave text.

Name
$sstatus$ - the condition code returned by several Proc statements.

Synopsis
$(field = \mid register = )$ sstatus

Return Value
sstatus is always an integer value. If a decimal value is assigned to sstatus, UNIFACE rounds it to the nearest integer. In general:
- A negative value in sstatus indicates an error.
- A positive value indicates a warning or information.
- 0 indicates a successful operation.
- sstatus contains the value of scurocc after a setocc statement.
**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 (Fieldbocc = 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**: Status - return the system time (pre-version 5.0), use `Sclk` instead.

**Synopsis**: `[field] [register] Status`

**Return Value**: `Status` returns the system time, accurate to one second.

**Name**: Storedbocc - return the number of occurrences of the entity that have been retrieved from a database.

**Synopsis**: `[field] [register] Storedbocc([entity])`

**Return Value**: The total number of occurrences of `entity` currently fetched from the database. The following triggers and statements affect the value of `Storedbocc`:
- `^NEXT ^OCCURRENCE` `Storedbocc` is set to the number retrieved.
- `^PREVIOUS ^OCCURRENCE` `Storedbocc` is set to the number retrieved.
retrieve  $totdbocc is set to the number retrieved.
store  $totdbocc is set to the number stored.
clear  $totdbocc is set to 0.

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

Synopsis  \( \text{[field = 1 register =]} \) $totlines

Return Value  When UNIFACE is printing ($printing = 1), $totlines 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 $totlines is used (that is, $printing = 0), UNIFACE returns a value of -1 to $status. The value of $totlines is 0 when UNIFACE is not printing.

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

Synopsis  \( \text{[field = 1 register =]} \) $totocc((entity))

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

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

Name  $user - return the user name.

Synopsis  \( \text{[field = 1 register =]} \) $user((path))

Return Value  $user returns the current user name.

Name  $variation - return or set the variation code.

Synopsis  \( \text{[field = 1 register =]} \) $variation(=* string*)

Return Value  $variation returns the current variation code.

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

Synopsis  \( \text{[field = 1 register =]} \) $workfilesize

UNIFACE V5.2
UNIFACE V5.2

Quick Reference Guide (101075201, 21 September 1992)
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:

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

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 ($10). This $register contains the value 2, which is used as offset into the string.

- $1 = NAME(4:6) ; extract positions 4 to 6 ; (LENGTH)
- $1 = NAME(1:3) ; extract positions 1 to 3 ; (KI)
- $1 = NAME(10:4) ; extract positions 2 to 5 ; (KLL)
- $1 = NAME(3) ; extract positions 3 to end ; (LENGTH)

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 $register to the length of string
- scan string, 'profile' ; set $register to the position of the start of the string matching 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(99P9P99), 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Interpreted as</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Date (null) time (null)</td>
</tr>
<tr>
<td>0.5</td>
<td>Date (null) 12:00:00</td>
</tr>
<tr>
<td>1</td>
<td>1-JAN-0000 time (00:00:00)</td>
</tr>
<tr>
<td>1.5</td>
<td>1-JAN-0000 12:00:00</td>
</tr>
</tbody>
</table>

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: 's') is 1/86400000 (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:

<table>
<thead>
<tr>
<th>Code (as fraction of 1 day)</th>
<th>Meaning/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Day</td>
</tr>
<tr>
<td>h</td>
<td>Hour</td>
</tr>
<tr>
<td>n</td>
<td>Minute</td>
</tr>
<tr>
<td>s</td>
<td>Second</td>
</tr>
<tr>
<td>t</td>
<td>Tick</td>
</tr>
</tbody>
</table>

Table 3-3 Date and time arithmetic codes.

In version 5.0, 'm' 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
3.3.3 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 point value. For example, table 3-4 shows how various expressions are interpreted by UNIFACE:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value (as a fraction of 1 day)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1t</td>
<td>1/8640000</td>
</tr>
<tr>
<td>1s</td>
<td>1/86400</td>
</tr>
<tr>
<td>1n</td>
<td>1/1440</td>
</tr>
<tr>
<td>1h</td>
<td>1/24</td>
</tr>
<tr>
<td>1d12h</td>
<td>1.5</td>
</tr>
<tr>
<td>2t</td>
<td>1/4320000</td>
</tr>
<tr>
<td>2s</td>
<td>1/720</td>
</tr>
<tr>
<td>2h</td>
<td>1/12</td>
</tr>
</tbody>
</table>

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

\[
\left(4 * \text{elapsed time} \right) / 31
\]

is reasonable, if the ELAPSED_TIME field contains an elapsed time which is not too big for this calculation, and if $31$ 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 "8640000" (8640000 ticks) is the same as "1d" (one day); UNIFACE treats both these expressions as one day, because there are 8640000 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 61 days. However, 61 days is the same number of days as 2 months and 1 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 365 days and not 366, 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: \texttt{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
two 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, \texttt{LDATE} is a field with the display format
\texttt{DIS(d.m.y)}. The example sets \$1 to \$5 to various date values, then
sets the LDATE field to various values by subtracting one \$Register
from another.

\begin{verbatim}
S1 = date("1 Feb 89")
S2 = date("1 Mar 89")
S3 = date("1 Apr 89")
S4 = date("1 May 89")
S5 = date("1 Jan 88")
\end{verbatim}

\begin{verbatim}
LDATE = S2 - S1 ;\texttt{LDATE} is now 28.0.0 (28 days)
LDATE = S3 - S2 ;\texttt{LDATE} is now 0.1.0 (31 days)
LDATE = S4 - S3 ;\texttt{LDATE} is now 1.2.0 (61 days)
LDATE = S1 - S5 ;\texttt{LDATE} is now 0.1.1 (397 days)
LDATE = S1 + S5 ;\texttt{LDATE} is now 6.1.1989
\end{verbatim}

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

\begin{verbatim}
S1 = start_datetime - end_datetime ;elapsed (date) time
S2 = datefield + pickup a date value
S3 = timefield + pickup a time value
S4 = S2 + S3 ;make combined date/time
S5 = S4 + S1 ;add elapsed date/time in \$1 to \$5
S6 = S5 + 1s ;add 1 second to date/time in \$5
S7 = S6 + 1min ;add 1 minute
S8 = S7 + 1hr ;add 1 hour
S9 = S8 + 1day ;add 1 day
S10 = S9 + 1mon ;add 1 month
S11 = S10 + 1yr ;add 1 year
\end{verbatim}

\begin{verbatim}
datefield = datefield + 6d ;add 6 days
datefield = datefield + .5d ;add half a day (12 hours)
S1 = datefield + 7.5d ;more than 6 days; week at 12 noon
S2 = datefield + .5d12h ;add one week and half a day
\end{verbatim}

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:

\texttt{(field | register) = source \{code\}}

The type of data to be extracted is specified by the code; each type of
eextractable 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:

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{Extract} & \textbf{Code} \\
\hline
Day number & D \\
Month number & M \\
Year (four digits) & Y \\
Fiscal year (four digits) & X \\
Week number & \textbf{W} \\
Day of week (Monday = 1) & \textbf{A} \\
Hour (24 hour clock) & \textbf{H} \\
\hline
\end{tabular}
\caption{Extraction Codes}
\end{table}
The following example shows how date and time arithmetic can be combined with the extraction facilities provided by UNIFACE:

```pascal
if (time < datimfield(clock) + 22000)
  message "This date/time is < 22.00 seconds from now"
end
```

### 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 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. 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 6.0 compatibility. Use 'N' instead.
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:

<table>
<thead>
<tr>
<th>Number of digits</th>
<th>Interpreted as</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>HH:MM:SS</td>
</tr>
<tr>
<td>5</td>
<td>H:MM:SS</td>
</tr>
<tr>
<td>4</td>
<td>HH:MM</td>
</tr>
<tr>
<td>3</td>
<td>HH</td>
</tr>
<tr>
<td>2</td>
<td>H</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

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:

```plaintext
; TEXT_TO_TIME - convert raw text to time
; jit.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
```

If you do not 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.
3.3.10 Converting to a date and time

To convert a string into a combined date and time, you should use $sdatim. The $sdatim 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-yyyy hh:mm:ss for the USA, USYS variation.

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

```plaintext
S1 = "27-02-66 12:23:39" ; $1 will be a string
S2 = $sdatim($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:

```plaintext
S1 = 000100 ; supposedly 1 minute, $1 will contain the value 100
S2 = $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:

<table>
<thead>
<tr>
<th>Extract</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>The integer part of a number</td>
<td>trunc</td>
</tr>
<tr>
<td>The rounded value of a number</td>
<td>round</td>
</tr>
<tr>
<td>The fractional part of a number</td>
<td>fraction</td>
</tr>
</tbody>
</table>

Table 3-7 Number extraction codes.

Examples

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

```plaintext
S25 = 123.74
S1 = S25[fraction] ; extract the fractional value, so $1 = 0.74
S1 = S25[trunc] ; extract the truncated value, so $1 = 123
S1 = S25[round] ; extract the rounded value, so $1 = 124
S1 = S25[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:

\[
\text{(where } S25 = -123.75) \\
S25 = S25[\text{round}] \quad \text{; round value in } S25: S25 = -124
\]

3.4.3 Converting to a number

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

\[
|\text{field} = | \text{register} | \text{number} ("\text{string}")
\]

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:

\[
\text{scan } S1, '8'
\]

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.

\[
S1 = "\text{New York New York}" \\
\text{scan } S1, '8' \quad \text{; find start of numeric data} \\
\text{if } (\text{result} > 0) \quad \text{; string } (S3) \text{ contains numeric data} \\
S3 = \text{result} \quad \text{; save result (start position of numeric data)} \\
\text{else} \\
\quad \text{message } "\$S1 \text{ does not contain numeric data}" \\
\quad \text{return } -1 \\
\text{endif} \\
S2 = \text{$number}(S1[S3]) \\
\text{puts "numeric part of } \$S1 \text{ is } \$S2"
\]

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:

![Debugging Procs Figure 4-1](image.png)

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 Procs 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

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

4.2.1 Breakpoint commands

The \texttt{break}, \texttt{call on}, \texttt{call off}, \texttt{cirmess on}, \texttt{cirmess off}, and \texttt{done} commands are used for setting breakpoints. A breakpoint is a point in the program where execution is stopped and control is transferred to the debug mode. The \texttt{break} command sets a breakpoint at the line specified by \texttt{module} and \texttt{line}. The \texttt{call on} command sets breakpoints on call statements. The \texttt{call off} command cancels these breakpoints. The \texttt{cirmess on} and \texttt{cirmess off} commands control message buildup in the message frame. The \texttt{done} command cancels all breakpoints when the application is completed.

4.2.2 Dump commands

The \texttt{dump}, \texttt{dump module}, and \texttt{examine} commands are used for dumping various types of information. The \texttt{dump} command displays contents of the next register. The \texttt{dump module} command displays contents of register $number. The \texttt{examine} command displays contents of field (\texttt{frame}.enname). The \texttt{examine (number)} command displays contents of register $number, or if $number is in double quotation marks ('), a string value. \texttt{ex} is optional ($\texttt{ex} = 3$ is also possible).

4.2.3 Execution commands

The \texttt{go}, \texttt{icprint}, \texttt{line}, \texttt{nof}, \texttt{putmess off}, \texttt{putmess on}, \texttt{quit}, \texttt{return on}, \texttt{return off}, \texttt{show}, \texttt{step}, \texttt{step (number)}, \texttt{trace on}, \texttt{trace off}, \texttt{trace (number)}, and \texttt{xtrace} commands are used for controlling program execution. The \texttt{go} command continues execution until the next break condition or \texttt{debug command}. The \texttt{icprint} command sends I/O messages to the message frame, even if the application definition has defined that none should be displayed. The \texttt{line (number)} command executes \texttt{number} Proc statements, but does not step through a called module. The \texttt{nof} command skips the current statement without executing it. The \texttt{putmess off} command redirects message frame input to the screen. The \texttt{putmess on} command redirects message frame input to the message frame. The \texttt{quit} command exits the application immediately. The \texttt{return on} command sets breakpoints on return and done statements. The \texttt{return off} command cancels these breakpoints. The \texttt{show} command displays break point settings. The \texttt{step} command executes one (the next) Proc statement. The \texttt{step (number)} command executes \texttt{number} Proc statements and steps into called subroutines. The \texttt{trace on} and \texttt{trace off} commands control message frame output. The \texttt{trace (number)} command sends messages to the message frame about which triggers are activated and when \texttt{number} = 0 or 1. Start the extended trace facility of the UNIFACE debugger.

4.2.4 Miscellaneous commands

The \texttt{xtrace} command starts 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 censore:

(IDB) Yrli edit censore > b censore 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 clrmsg off

Allow messages to build up in the message frame.

4.2.6 clrmsg 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 (putmsg off) or to the message frame (putmsg 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 putmsg 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. city =
"Amsterdam" is also possible, therefore.

4.2.14 ex [[function(name)]
Display one of the status request functions, for example $entname,
$ filedmod.
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 pumess off
Redirect the message frame input to the screen.

4.2.20 pumess 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 `calendar` module, this command shows the following:

```
(DBUG) CENSORE: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.

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

`table 4-2 continues`
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.
5.2 Reserved words

Do not use the following as names for any objects:

- IDF application dictionary names (see subsection 5.2.1 IDF application dictionary names).

Do not use the following as entity names:

- HEADER.
- PRATT.
- TRAILER.

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:

- UNISCODELL
- 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.

5.2.2 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:
Chapter 6  Interface definition

6.1 Data types (UNIFACE)

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

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).

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.

<table>
<thead>
<tr>
<th>Packing code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1-C*</td>
<td>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)</td>
</tr>
<tr>
<td>VC1-VC*</td>
<td>Variable length character string</td>
</tr>
<tr>
<td>SC1-SC*</td>
<td>Segmented character string</td>
</tr>
<tr>
<td>U1-U*</td>
<td>TRX character</td>
</tr>
<tr>
<td>VU1-VU*</td>
<td>TRX length variable character string</td>
</tr>
<tr>
<td>SU1-SU*</td>
<td>TRX segmented character string</td>
</tr>
<tr>
<td>R1-R*</td>
<td>Binary (raw)</td>
</tr>
<tr>
<td>SR1-SR*</td>
<td>Segmented binary (raw)</td>
</tr>
<tr>
<td>VR1-VR*</td>
<td>Variable length binary (raw)</td>
</tr>
<tr>
<td>I1</td>
<td>One-byte integer</td>
</tr>
<tr>
<td>I2</td>
<td>Two-byte integer</td>
</tr>
<tr>
<td>I3</td>
<td>Three-byte integer</td>
</tr>
<tr>
<td>I4</td>
<td>Four-byte integer</td>
</tr>
<tr>
<td>I8</td>
<td>Eight-byte integer</td>
</tr>
<tr>
<td>M1</td>
<td>Money: eight-byte integer, scaling 2</td>
</tr>
<tr>
<td>M2</td>
<td>Money: double precision D-float</td>
</tr>
<tr>
<td>M4</td>
<td>SYBASE money format, scaling 4</td>
</tr>
<tr>
<td>N1-N32</td>
<td>Number, stored without decimal point</td>
</tr>
<tr>
<td>P1-P8</td>
<td>Packed decimal, +/- at beginning of field</td>
</tr>
<tr>
<td>Q1-Q8</td>
<td>Packed decimal, +/- at end of field</td>
</tr>
<tr>
<td>F</td>
<td>Optimum DBMS floating point default</td>
</tr>
<tr>
<td>F4</td>
<td>Single precision F-float</td>
</tr>
<tr>
<td>F8</td>
<td>Double precision D-float</td>
</tr>
<tr>
<td>D</td>
<td>Optimum DBMS date default</td>
</tr>
<tr>
<td>D1</td>
<td>ASCII date DD-MM-YYYY</td>
</tr>
<tr>
<td>D2</td>
<td>ASCII date YYYYMMDD</td>
</tr>
<tr>
<td>D3</td>
<td>ASCII date DDMMYYYY</td>
</tr>
<tr>
<td>D4</td>
<td>ASCII date YYYYYY</td>
</tr>
<tr>
<td>D5</td>
<td>ASCII date DDMMY</td>
</tr>
<tr>
<td>D6</td>
<td>Binary date YMD</td>
</tr>
<tr>
<td>D7</td>
<td>Binary date DMYY</td>
</tr>
<tr>
<td>D8</td>
<td>Binary date YMD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Packing code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>D9</td>
<td>Binary date DMY</td>
</tr>
<tr>
<td>D10</td>
<td>Binary date YYMMD</td>
</tr>
<tr>
<td>D11</td>
<td>Binary date DDMYY</td>
</tr>
<tr>
<td>E</td>
<td>Optimum DBMS combined date/time default</td>
</tr>
<tr>
<td>E1</td>
<td>SYBASE linear four-byte date and four-byte time</td>
</tr>
<tr>
<td>E2</td>
<td>RMS linear date and time</td>
</tr>
<tr>
<td>E3</td>
<td>ASCII date DDMMYYYYY time HH:NN:SS</td>
</tr>
<tr>
<td>E4</td>
<td>ASCII date DDMMMYYYY time HHNNSS</td>
</tr>
<tr>
<td>E5</td>
<td>Ingres date DD-MM-YYYY time HH:NN:SS</td>
</tr>
<tr>
<td>E6</td>
<td>ORACLE internal date/time format</td>
</tr>
<tr>
<td>E7</td>
<td>SYBASE ASCII date MM/DD/YYYY HH:NN:SS.TT</td>
</tr>
<tr>
<td>E8</td>
<td>ASCII datetime YYYYMMMDH:MMSS (like D2+T2)</td>
</tr>
<tr>
<td>T</td>
<td>Optimum DBMS time default</td>
</tr>
<tr>
<td>T1</td>
<td>ASCII time HH:NN:SS</td>
</tr>
<tr>
<td>T2</td>
<td>ASCII time HHMMSS</td>
</tr>
<tr>
<td>T3</td>
<td>ASCII date DDMMYYYY time HHMMSS</td>
</tr>
<tr>
<td>B</td>
<td>Optimum DBMS Boolean default</td>
</tr>
<tr>
<td>B1</td>
<td>ASCII Boolean 0/1</td>
</tr>
<tr>
<td>B2</td>
<td>ASCII Boolean F/T</td>
</tr>
<tr>
<td>B3</td>
<td>ASCII Boolean N/Y</td>
</tr>
<tr>
<td>B4</td>
<td>Binary Boolean 0/1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Packing code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>D9</td>
<td>Binary date DMY</td>
</tr>
<tr>
<td>D10</td>
<td>Binary date YYMMD</td>
</tr>
<tr>
<td>D11</td>
<td>Binary date DDMYY</td>
</tr>
<tr>
<td>E</td>
<td>Optimum DBMS combined date/time default</td>
</tr>
<tr>
<td>E1</td>
<td>SYBASE linear four-byte date and four-byte time</td>
</tr>
<tr>
<td>E2</td>
<td>RMS linear date and time</td>
</tr>
<tr>
<td>E3</td>
<td>ASCII date DDMMYYYYY time HH:NN:SS</td>
</tr>
<tr>
<td>E4</td>
<td>ASCII date DDMMMYYYY time HHNNSS</td>
</tr>
<tr>
<td>E5</td>
<td>Ingres date DD-MM-YYYY time HH:NN:SS</td>
</tr>
<tr>
<td>E6</td>
<td>ORACLE internal date/time format</td>
</tr>
<tr>
<td>E7</td>
<td>SYBASE ASCII date MM/DD/YYYY HH:NN:SS.TT</td>
</tr>
<tr>
<td>E8</td>
<td>ASCII datetime YYYYMMMDH:MMSS (like D2+T2)</td>
</tr>
<tr>
<td>T</td>
<td>Optimum DBMS time default</td>
</tr>
<tr>
<td>T1</td>
<td>ASCII time HH:NN:SS</td>
</tr>
<tr>
<td>T2</td>
<td>ASCII time HHMMSS</td>
</tr>
<tr>
<td>T3</td>
<td>ASCII date DDMMYYYY time HHMMSS</td>
</tr>
<tr>
<td>B</td>
<td>Optimum DBMS Boolean default</td>
</tr>
<tr>
<td>B1</td>
<td>ASCII Boolean 0/1</td>
</tr>
<tr>
<td>B2</td>
<td>ASCII Boolean F/T</td>
</tr>
<tr>
<td>B3</td>
<td>ASCII Boolean N/Y</td>
</tr>
<tr>
<td>B4</td>
<td>Binary Boolean 0/1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Packing code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>D9</td>
<td>Binary date DMY</td>
</tr>
<tr>
<td>D10</td>
<td>Binary date YYMMD</td>
</tr>
<tr>
<td>D11</td>
<td>Binary date DDMYY</td>
</tr>
<tr>
<td>E</td>
<td>Optimum DBMS combined date/time default</td>
</tr>
<tr>
<td>E1</td>
<td>SYBASE linear four-byte date and four-byte time</td>
</tr>
<tr>
<td>E2</td>
<td>RMS linear date and time</td>
</tr>
<tr>
<td>E3</td>
<td>ASCII date DDMMYYYYY time HH:NN:SS</td>
</tr>
<tr>
<td>E4</td>
<td>ASCII date DDMMMYYYY time HHNNSS</td>
</tr>
<tr>
<td>E5</td>
<td>Ingres date DD-MM-YYYY time HH:NN:SS</td>
</tr>
<tr>
<td>E6</td>
<td>ORACLE internal date/time format</td>
</tr>
<tr>
<td>E7</td>
<td>SYBASE ASCII date MM/DD/YYYY HH:NN:SS.TT</td>
</tr>
<tr>
<td>E8</td>
<td>ASCII datetime YYYYMMMDH:MMSS (like D2+T2)</td>
</tr>
<tr>
<td>T</td>
<td>Optimum DBMS time default</td>
</tr>
<tr>
<td>T1</td>
<td>ASCII time HH:NN:SS</td>
</tr>
<tr>
<td>T2</td>
<td>ASCII time HHMMSS</td>
</tr>
<tr>
<td>T3</td>
<td>ASCII date DDMMYYYY time HHMMSS</td>
</tr>
<tr>
<td>B</td>
<td>Optimum DBMS Boolean default</td>
</tr>
<tr>
<td>B1</td>
<td>ASCII Boolean 0/1</td>
</tr>
<tr>
<td>B2</td>
<td>ASCII Boolean F/T</td>
</tr>
<tr>
<td>B3</td>
<td>ASCII Boolean N/Y</td>
</tr>
<tr>
<td>B4</td>
<td>Binary Boolean 0/1</td>
</tr>
</tbody>
</table>

Table 6-2: Packing codes allowed in UNIFACE.
6.3 Allowed combinations

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

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 strings that you want to use as a string identifier in the Field identifier entry. (For example, ^28.)

Length identification

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

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.

<table>
<thead>
<tr>
<th>Syntax codes</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>One digit.</td>
</tr>
<tr>
<td>#*</td>
<td>Zero-n digits.</td>
</tr>
<tr>
<td>&amp;</td>
<td>One letter.</td>
</tr>
<tr>
<td>&amp;*</td>
<td>Zero-n letters.</td>
</tr>
<tr>
<td>@</td>
<td>One letter or one digit or underscore.</td>
</tr>
<tr>
<td>@*</td>
<td>0-n letters, digits or underscores.</td>
</tr>
<tr>
<td>?</td>
<td>One ASCII character.</td>
</tr>
<tr>
<td>?*</td>
<td>0-n ASCII characters.</td>
</tr>
<tr>
<td>*</td>
<td>0-n ASCII characters (same as ?*).</td>
</tr>
<tr>
<td>A-Z</td>
<td>That uppercase letter, that is, A, or B, or C, etc.</td>
</tr>
<tr>
<td>a-z</td>
<td>That letter, upper or lowercase, that is, A or a, B or b, etc.</td>
</tr>
<tr>
<td>Any ASCII char</td>
<td>That ASCII character, not a syntax string code.</td>
</tr>
<tr>
<td>%any ASCII char</td>
<td>That literal ASCII character, that is, not a variable code.</td>
</tr>
<tr>
<td>(Any)</td>
<td>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).</td>
</tr>
</tbody>
</table>

Table 7-1 Syntax string codes.
### 7.1.1 Entry format examples

<table>
<thead>
<tr>
<th>Entry format</th>
<th>Allowed (for example)</th>
<th>Not allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>*</code></td>
<td>1000</td>
<td>1.000</td>
</tr>
<tr>
<td></td>
<td>[nothing]</td>
<td>0000.0</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>34 456</td>
</tr>
<tr>
<td></td>
<td>1234567</td>
<td>10A</td>
</tr>
<tr>
<td><code>*</code></td>
<td>Any text with an asterisk (*) as the last character.</td>
<td>Text without an asterisk as the last character.</td>
</tr>
<tr>
<td><code>*</code> <code>*</code></td>
<td>1000.00</td>
<td>1.000.00</td>
</tr>
<tr>
<td></td>
<td>0.50</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>0.50</td>
<td>0.50</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>1000.00</td>
</tr>
<tr>
<td>Mr. Smith</td>
<td>MR. SMITH</td>
<td>mr. smith</td>
</tr>
<tr>
<td></td>
<td>Mr. Smith</td>
<td>Mr. Smith</td>
</tr>
<tr>
<td></td>
<td>Mr. Smith</td>
<td>Mr. Smith</td>
</tr>
<tr>
<td></td>
<td>Mr. Smith</td>
<td>Mr. Jones</td>
</tr>
<tr>
<td>000-0000</td>
<td>404 396-3040</td>
<td>41 396-3040</td>
</tr>
<tr>
<td></td>
<td>396-3040</td>
<td>396-3040</td>
</tr>
<tr>
<td>000-0000</td>
<td>(404) 396-3040</td>
<td>396-3040</td>
</tr>
<tr>
<td></td>
<td>396-3040</td>
<td>396-3040</td>
</tr>
<tr>
<td><code>*</code></td>
<td>Smith</td>
<td>Smith</td>
</tr>
<tr>
<td></td>
<td>Smith, Jones</td>
<td>Mr. Smith</td>
</tr>
<tr>
<td></td>
<td>1234562</td>
<td>12345*2</td>
</tr>
<tr>
<td><code>#</code> <code>#</code></td>
<td>1 23-a</td>
<td>123-a</td>
</tr>
<tr>
<td></td>
<td>1 23-a</td>
<td>1 23-a</td>
</tr>
<tr>
<td><code>*</code> <code>#</code></td>
<td>123,45</td>
<td>12345</td>
</tr>
<tr>
<td></td>
<td>0.30</td>
<td>.3</td>
</tr>
<tr>
<td></td>
<td>a30</td>
<td>.30</td>
</tr>
<tr>
<td></td>
<td>121/33</td>
<td>121.300</td>
</tr>
<tr>
<td><code>#</code></td>
<td>Anything else!</td>
<td>Anything else!</td>
</tr>
<tr>
<td><code>[ ]</code> <code>[ ]</code></td>
<td>J</td>
<td>Anything else!</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>Anything else!</td>
</tr>
</tbody>
</table>

| Table 7-2 | Example entry formats. |

### 7.2 Display/Edit/Prompt

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Display and edit (this is the default value).</td>
</tr>
<tr>
<td>1</td>
<td>Display only.</td>
</tr>
<tr>
<td>2</td>
<td>Edit only (for non-echoing passwords).</td>
</tr>
<tr>
<td>3</td>
<td>Display and edit, no prompt.</td>
</tr>
<tr>
<td>4</td>
<td>Display only, no prompt.</td>
</tr>
<tr>
<td>5</td>
<td>Edit only, no prompt.</td>
</tr>
<tr>
<td>6</td>
<td>No display, no edit and no prompt.</td>
</tr>
</tbody>
</table>

| Table 7-3 | Display/Edit/Prompt codes. |

**Procs**

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

**Examples (for string fields)**

- `if (field == '???')`: test for empty field
- `if (field == '***')`: test for asterisk in last character
- `if (field == '***')`: test for asterisk anywhere in string
- `if (field == '***')`: test for field which starts with letter

**Example $syntax**

```bash
gsnyisrk
if (string == 'Vmrden')
   EL = 'Notan';
if (string = $syntax(string))
   message "String checks out!";
   endif
```

---

Quick Reference Guide (101075201, 21 September 1992)
7.3 Characters allowed in a field

<table>
<thead>
<tr>
<th>Characters</th>
<th>What they are</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digits only</td>
<td>0-9</td>
</tr>
<tr>
<td>Numbers only</td>
<td>0-9, . , + -</td>
</tr>
<tr>
<td>ASCII only</td>
<td>UNIFACE font 0</td>
</tr>
<tr>
<td>ISO Latin-1</td>
<td>UNIFACE fonts 0 and 1</td>
</tr>
<tr>
<td>Full char. set</td>
<td>UNIFACE fonts 0 through 7</td>
</tr>
</tbody>
</table>

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

7.4 Shorthand codes for Field syntax model

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM</td>
<td>Numbers only.</td>
</tr>
<tr>
<td>NUND</td>
<td>Underlining not allowed.</td>
</tr>
<tr>
<td>OVS</td>
<td>Overstrike.</td>
</tr>
<tr>
<td>PRO(chars)</td>
<td>Profile allowed.</td>
</tr>
<tr>
<td>RCS</td>
<td>Replace contiguous spaces with one space.</td>
</tr>
<tr>
<td>REP(n-m)</td>
<td>Repetition of subfield: ( n = ) minimum, ( m = ) maximum.</td>
</tr>
<tr>
<td>UPC</td>
<td>All uppercase.</td>
</tr>
<tr>
<td>YBLD</td>
<td>Bold allowed.</td>
</tr>
<tr>
<td>YDCC</td>
<td>Delete all control characters.</td>
</tr>
<tr>
<td>YDCX</td>
<td>Delete all text control characters.</td>
</tr>
<tr>
<td>YITA</td>
<td>Italics allowed.</td>
</tr>
<tr>
<td>YUND</td>
<td>Underlining allowed.</td>
</tr>
</tbody>
</table>

table 7-5 Shorthand codes for Field syntax model.
# 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

<table>
<thead>
<tr>
<th>Display format</th>
<th>What is displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>Character from data element.</td>
</tr>
<tr>
<td>%?</td>
<td>One question mark.</td>
</tr>
<tr>
<td>%%</td>
<td>One percent symbol.</td>
</tr>
<tr>
<td>Any ASCII char</td>
<td>That ASCII character as a constant.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Examples</th>
<th>Display format</th>
<th>Input</th>
<th>Displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mr. ??????</td>
<td>Smith</td>
<td>Mr. Smith</td>
</tr>
<tr>
<td></td>
<td>Mr. ??????</td>
<td>Jumpin' Jack Flash</td>
<td>Mr. Jump</td>
</tr>
<tr>
<td></td>
<td>Mr. ??????%</td>
<td>Smith</td>
<td>Mr. Smith</td>
</tr>
<tr>
<td></td>
<td>Mr. ?????%?</td>
<td>Jumpin' Jack Flash</td>
<td>Mr. Jump</td>
</tr>
<tr>
<td></td>
<td>Mr. ?????%?%</td>
<td>Smith</td>
<td>Mr. Smith</td>
</tr>
<tr>
<td></td>
<td>Mr. ?????%%</td>
<td>Jumpin' Jack Flash</td>
<td>Mr. Jump</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Examples</th>
<th>Display format</th>
<th>Input</th>
<th>Displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mr. ??????</td>
<td>Smith</td>
<td>Mr. Smith</td>
</tr>
<tr>
<td></td>
<td>Mr. ??????</td>
<td>Jumpin' Jack Flash</td>
<td>Mr. Jump</td>
</tr>
<tr>
<td></td>
<td>Mr. ??????%</td>
<td>Smith</td>
<td>Mr. Smith</td>
</tr>
<tr>
<td></td>
<td>Mr. ?????%?</td>
<td>Jumpin' Jack Flash</td>
<td>Mr. Jump</td>
</tr>
<tr>
<td></td>
<td>Mr. ?????%?%</td>
<td>Smith</td>
<td>Mr. Smith</td>
</tr>
<tr>
<td></td>
<td>Mr. ?????%%</td>
<td>Jumpin' Jack Flash</td>
<td>Mr. Jump</td>
</tr>
</tbody>
</table>
### 8.2 Numeric (and float)

<table>
<thead>
<tr>
<th>Display format</th>
<th>What is displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Digit, or leading/trailing zero.</td>
</tr>
<tr>
<td>z</td>
<td>Digit, suppress zeros if leading or trailing (after decimal).</td>
</tr>
<tr>
<td>B</td>
<td>Spaces for suppressed zeros, '+' and '-' signs.</td>
</tr>
<tr>
<td>+</td>
<td>+ to left or right if value is positive (&gt;0).</td>
</tr>
<tr>
<td>-</td>
<td>- to left or right if value is negative (&lt;0).</td>
</tr>
<tr>
<td>P</td>
<td>Fixed decimal point.</td>
</tr>
<tr>
<td>K</td>
<td>Fixed decimal comma.</td>
</tr>
<tr>
<td>.</td>
<td>Layout decimal point.</td>
</tr>
<tr>
<td>,</td>
<td>Layout decimal comma.</td>
</tr>
</tbody>
</table>

*Table 8-3: Numeric display format codes.*

<table>
<thead>
<tr>
<th>Display format</th>
<th>Input</th>
<th>Displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>99999</td>
<td>12345</td>
<td>12345</td>
</tr>
<tr>
<td></td>
<td>123</td>
<td>00123</td>
</tr>
<tr>
<td></td>
<td>00123</td>
<td>00123</td>
</tr>
<tr>
<td></td>
<td>123456</td>
<td>error: &quot;too much data&quot;</td>
</tr>
<tr>
<td></td>
<td>-1234</td>
<td>error: &quot;negatives not allowed&quot;</td>
</tr>
<tr>
<td></td>
<td>12345</td>
<td>12345 (no point defined)</td>
</tr>
<tr>
<td>zzzzz</td>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td>01234</td>
<td>01234</td>
</tr>
<tr>
<td>-zzzz</td>
<td>123</td>
<td>-123</td>
</tr>
<tr>
<td>-zzzzB</td>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td>zzzzz-</td>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td>+zzzz</td>
<td>123</td>
<td>+123</td>
</tr>
<tr>
<td>++zzzz</td>
<td>123</td>
<td>+123</td>
</tr>
<tr>
<td>~zzzz99</td>
<td>123</td>
<td>-123</td>
</tr>
<tr>
<td>3~zzz99</td>
<td>123</td>
<td>-123</td>
</tr>
<tr>
<td>999999</td>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td>12345</td>
<td>12345</td>
</tr>
<tr>
<td></td>
<td>12.3</td>
<td>012.3</td>
</tr>
<tr>
<td></td>
<td>1234.5</td>
<td>error: &quot;too much data&quot;</td>
</tr>
<tr>
<td></td>
<td>123.456</td>
<td>error: &quot;too much data&quot;</td>
</tr>
<tr>
<td>zzz99zzz99</td>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td>.997</td>
<td>0.997</td>
</tr>
<tr>
<td></td>
<td>012.120</td>
<td>12.12</td>
</tr>
<tr>
<td>zzz.zz.zzz</td>
<td>12345678</td>
<td>123.45,678</td>
</tr>
<tr>
<td></td>
<td>12345</td>
<td>12.345</td>
</tr>
<tr>
<td></td>
<td>1.234</td>
<td>1.234</td>
</tr>
<tr>
<td></td>
<td>123.45</td>
<td>123.45,67</td>
</tr>
</tbody>
</table>

*Table 8-4: Example display format codes for numeric fields.*
8.3 Date

<table>
<thead>
<tr>
<th>Display format</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Day number in one or two digits.</td>
</tr>
<tr>
<td>dd</td>
<td>Day number in two digits.</td>
</tr>
<tr>
<td>zd</td>
<td>Day number in two digits or one space and one digit.</td>
</tr>
<tr>
<td>aa</td>
<td>Two-letter lowercase abbreviation from day name.</td>
</tr>
<tr>
<td>AA</td>
<td>As aa, by uppercase.</td>
</tr>
<tr>
<td>Aa</td>
<td>As aa, but initial caps.</td>
</tr>
<tr>
<td>aa*</td>
<td>Full day name in lowercase.</td>
</tr>
<tr>
<td>AA*</td>
<td>As aa*, but uppercase.</td>
</tr>
<tr>
<td>Aa*</td>
<td>As aa*, but initial caps.</td>
</tr>
<tr>
<td>m</td>
<td>Month number in one or two digits.</td>
</tr>
<tr>
<td>mm</td>
<td>Month number in two digits.</td>
</tr>
<tr>
<td>zm</td>
<td>Month number in two digits or one space and one digit.</td>
</tr>
<tr>
<td>mmm</td>
<td>Three-letter lowercase abbreviation for month.</td>
</tr>
<tr>
<td>MMM</td>
<td>As mmm, but uppercase.</td>
</tr>
<tr>
<td>mmm*</td>
<td>Full month name in lowercase.</td>
</tr>
<tr>
<td>MMM*</td>
<td>As mmm*, but uppercase.</td>
</tr>
<tr>
<td>Mmm*</td>
<td>As mmm*, but initial caps.</td>
</tr>
<tr>
<td>w</td>
<td>Week number in one or two digits.</td>
</tr>
<tr>
<td>ww</td>
<td>Week number in two digits.</td>
</tr>
<tr>
<td>zw</td>
<td>Week number in two digits or one space and one digit.</td>
</tr>
<tr>
<td>yyyy</td>
<td>Calendar year in four digits.</td>
</tr>
<tr>
<td>yy</td>
<td>Calendar year in two digits.</td>
</tr>
<tr>
<td>xxx</td>
<td>Fiscal year in four digits.</td>
</tr>
<tr>
<td>xx</td>
<td>Fiscal year in two digits.</td>
</tr>
<tr>
<td>lcode</td>
<td>Number of days, months or years as a linear value, using one of the above codes.</td>
</tr>
</tbody>
</table>

**Examples (non-linear)**

<table>
<thead>
<tr>
<th>Display format</th>
<th>Displayed (1)</th>
<th>Displayed (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mmm* d, yyyy</td>
<td>March 16, 1990</td>
<td>June 2, 1990</td>
</tr>
<tr>
<td>AA. MMM d</td>
<td>PST, Mar 16</td>
<td>EST, Mar 2</td>
</tr>
<tr>
<td>dd/mm/yyyy</td>
<td>16/03/90</td>
<td>02/06/90</td>
</tr>
<tr>
<td>mm/dd/yyyy</td>
<td>03/14/90</td>
<td>06/02/90</td>
</tr>
<tr>
<td>dd/mm/yyyy</td>
<td>16/3/90</td>
<td>2/6/90</td>
</tr>
<tr>
<td>zd/mm/yyyy</td>
<td>16/3/90</td>
<td>2/6/90</td>
</tr>
</tbody>
</table>

**Examples (linear)**

<table>
<thead>
<tr>
<th>Display format</th>
<th>Value</th>
<th>Displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ldd.mm. yyyy</td>
<td>25 December, 1990</td>
<td>25.11.1990</td>
</tr>
<tr>
<td>Ldd.mm.mm. yyyy</td>
<td>11 months and 25 days</td>
<td>25.11.0</td>
</tr>
</tbody>
</table>

**Note:**
- Table 8-5: Date display format codes.
- Table 8-6: Example display format codes for date fields.
- Table 8-7: Example display format codes for linear date fields.
8.4 Time

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

Table 8-8: Time display format codes.

Examples (non-linear)

<table>
<thead>
<tr>
<th>Display format</th>
<th>Displayed (1)</th>
<th>Displayed (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16:15</td>
<td>09:05</td>
<td></td>
</tr>
<tr>
<td>16:15</td>
<td>09:05</td>
<td></td>
</tr>
<tr>
<td>16:15.2</td>
<td>09:05.0</td>
<td></td>
</tr>
<tr>
<td>16:15.2</td>
<td>09:05.3</td>
<td></td>
</tr>
<tr>
<td>16:15.2</td>
<td>9: 5. 3</td>
<td></td>
</tr>
</tbody>
</table>

Table 8-9: Example display format codes for time.

Examples (linear)

<table>
<thead>
<tr>
<th>Display format</th>
<th>Value</th>
<th>Displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>27 days, 3 hours, 31 minutes</td>
<td>27.3.31.0</td>
<td></td>
</tr>
<tr>
<td>71 minutes, 29 seconds</td>
<td>0.1.11.29</td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Display format</th>
<th>Displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>dd MM yyyy hhmmss</td>
<td>2 APR 1991 14:15:39</td>
</tr>
</tbody>
</table>

Table 8-11: Example display format for date and time.

8.6 Shorthand codes for Field layout model

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

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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inverse</td>
<td>ONEMORE (only valid for fields).</td>
</tr>
<tr>
<td>Bright</td>
<td>VERY BRIGHT (only valid for fields).</td>
</tr>
<tr>
<td>Blinking</td>
<td>Blinking fields: impossible to re-create here!</td>
</tr>
<tr>
<td>Underlined</td>
<td>UNDERLINED (only valid for fields).</td>
</tr>
<tr>
<td>Color number</td>
<td>See figure 9-1.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>USYS color code matrix</th>
<th>Foreground colors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>system</td>
</tr>
<tr>
<td>system</td>
<td>0*</td>
</tr>
<tr>
<td>blue</td>
<td>8</td>
</tr>
<tr>
<td>green</td>
<td>16</td>
</tr>
<tr>
<td>cyan</td>
<td>24</td>
</tr>
<tr>
<td>red</td>
<td>32</td>
</tr>
<tr>
<td>purple</td>
<td>40</td>
</tr>
<tr>
<td>brown</td>
<td>48</td>
</tr>
<tr>
<td>white</td>
<td>56</td>
</tr>
</tbody>
</table>

Legend: 
- = Uses system defaults due to impossible combinations.
\* = 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

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

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

- F17 outside UNIFACE
- F17 in UNIFACE is the ^ACCEPT function
- <GOLD> F17 in UNIFACE is the ^QUIT function

F10: Pulldown
F11: Frame Switch
F12: View
F13: Message SQL
F14: Zoom Quick
Help
F1: Detail Menu
F17: Accept Quit
F18: Print Attr
F19: Retrieve (SEQ)
F20: Store Erase

Tab
Field
Find Profile
Insert
Remove
Select Reset
Prv.Sr Home
Next.Sr Bottom
Up UP fast
Down DN fast
Left LFT fast
Right RT fast
1 Word
2 Line
3 Text Fid.W
Enter
Field Detail
0 Character
Overstrk

Depending on your system:
- TERM = vt100
- UKEYB = vt100
- UDISP = vt100

10.4 Hewlett Packard HP-HIL

- F8 outside UNIFACE
- F8 in UNIFACE is the ^DETAIL function
- <GOLD> F8 in UNIFACE is the ^MENU function

F2: Pulldown CompTr
F3: Frame Switch
F4: View
F5: Message SQL
F6: Zoom Quick
Help
F7: Detail Menu
F8: Accept Quit
Print Attr
Retrieve (SEQ)
Store Erase

Tab
Field
PF1: Gold Reset
PF2: Select Reset
PF3: Find Prof
PF4: AddDoc InsDoc
7 Occur Ent.W
8 Ruler Chara
9 Clear Refine
RemCh Insert
4 Next Last
5 Previous First
6 Save Unsave
RemCh Insert
1 Word
2 Line
3 Text Fid.W
Enter Field Detail
0 Character
Overstrk
Up UP fast
Down DN fast
Left LFT fast
Right RT fast
UDISP = vt100, UKEYB = HP_HIL
10.5 IBM PC AT 83/84 key

Other:
- Not in zoom: Field In zoom: Tab
- Not in zoom: Field In zoom: Tab

GOLD
- Esc
- Num Lock
- Scroll Lock
- Sys Req

F3
- AddDoc
- InsDoc
- Remove
- Insert

F4
- AddDoc
- InsDoc
- Remove
- Insert

F5
- Occur
- EnWin
- F6
- Ruler
- CharAt

F7
- Line Select
- Text
- FltWin

F8
- Down
- DWinSet
- F9
- Char Profile
- F10
- Pulldown

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.

10.6 IBM AT 101/102 key enhanced

Other:
- Not in zoom: Field In zoom: Tab
- Not in zoom: Field In zoom: Tab

GOLD
- Esc
- Num Lock
- Scroll Lock
- Sys Req

F3
- AddDoc
- InsDoc
- Remove
- Insert

F4
- AddDoc
- InsDoc
- Remove
- Insert

F5
- Occur
- EnWin
- F6
- Ruler
- CharAt

F7
- Line Select
- Text
- FltWin

F8
- Down
- DWinSet
- F9
- Char Profile
- F10
- Pulldown

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

UNIX: UKKEYB & UDISP = IBMPCX
To force this layout on an MS-DOS machine, use the following command:
```
set usekey=ukcory
```
To force enhanced keyboard calls, use the following command:
```
set usekey=ukcory
```
For HELP, use GOLD-H.

Caution: This keyboard translation table maps both UNIFACE characters 7.8 and 7.9 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.x (4.^117).

10.7 IBM RS6000 console

**UNIFACE V5.2**

**10.8 IBM 6150 RT PC console**

F12 outside UNIFACE

F12 in UNIFACE is the ^STORE function

<^GOLD> F12 in UNIFACE is the ^ERASE function

F4 in UNIFACE is the ^REMOVE function

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

F4 outside UNIFACE

UDISP & UKKEYB = IBMRS6000

Quick Reference Guide (101075201, 21 September 1992)
10.11 SCO-Xenix

UDISP & UKEYB = ibmpecx

10.12 Siemens 97801

TERM = 97801.
UDISP & UKEYB = sun
Using the VT100 tool on the Sun console gives a better screen.
10.15 Sun-4 SPARC station

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

<table>
<thead>
<tr>
<th>Press</th>
<th>Then press</th>
<th>To set mode</th>
<th>Then press</th>
<th>To apply mode to</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Add</td>
<td>C</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>Insert</td>
<td>W</td>
<td>Word</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>Remove</td>
<td>L</td>
<td>Line</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>first (Top)</td>
<td>.</td>
<td>selected block</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>last (Bottom)</td>
<td>D</td>
<td>Data (text window)</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>Next</td>
<td>F</td>
<td>Field</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>Previous</td>
<td>O</td>
<td>Occurrence</td>
<td></td>
</tr>
<tr>
<td></td>
<td>space bar</td>
<td>E</td>
<td>Entity window</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>S</td>
<td>Screen</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>file</td>
<td></td>
</tr>
</tbody>
</table>

Example

GOLD space bar R . = remove selected block

Table 10-2 ‘Super’ key combinations.
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

<table>
<thead>
<tr>
<th>Switch or sub-switch</th>
<th>Purpose of switch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IDF compiler switches</strong></td>
<td></td>
</tr>
<tr>
<td>idf /all*</td>
<td>Compile everything.</td>
</tr>
<tr>
<td>idf /ap application</td>
<td>Compile application(s).</td>
</tr>
<tr>
<td>idf /bar</td>
<td>Compile menu bar and pulldowns in variation.</td>
</tr>
<tr>
<td>idf /con variation</td>
<td>Compile central objects in variation.</td>
</tr>
<tr>
<td>idf /con conceptual schema</td>
<td>Compile conceptual schema.</td>
</tr>
<tr>
<td>idf /cross [all]</td>
<td>Start IDF with xref on, or compile &amp; xref all.</td>
</tr>
<tr>
<td>idf /dev variation</td>
<td>Compile device tables in variation.</td>
</tr>
<tr>
<td>idf /ext external schema(s)</td>
<td>Compile external schema(s).</td>
</tr>
<tr>
<td>idf /install installation_object</td>
<td>Install and compile UOBJECT or demo.</td>
</tr>
<tr>
<td>idf /lib</td>
<td>Compile central Proc in library.</td>
</tr>
<tr>
<td>idf /man variation</td>
<td>Compile man bar and pulldowns in variation.</td>
</tr>
<tr>
<td>idf /mes variation</td>
<td>Compile messages in variation.</td>
</tr>
<tr>
<td>idf /obj variation</td>
<td>Compile object central objects in variation.</td>
</tr>
<tr>
<td>idf /tcm variation</td>
<td>Compile translation tables in variation.</td>
</tr>
<tr>
<td>idf /tram variation</td>
<td>Compile translation tables in variation.</td>
</tr>
<tr>
<td><strong>IDF compiler sub-switches</strong></td>
<td>Name new translation table to compile.</td>
</tr>
<tr>
<td>/fil</td>
<td>Compile and return all compiler messages.</td>
</tr>
<tr>
<td>/inf</td>
<td>Compile with Proc listing.</td>
</tr>
<tr>
<td>/lib</td>
<td>Compile with warning and error messages.</td>
</tr>
<tr>
<td>/war</td>
<td>Other IDF switches</td>
</tr>
<tr>
<td>idf /exp application export_file</td>
<td>Export application to export file, after /pre.</td>
</tr>
<tr>
<td>idf /help</td>
<td>Show this list.</td>
</tr>
<tr>
<td>idf /hel</td>
<td>Show this list.</td>
</tr>
<tr>
<td>idf /imp export_file</td>
<td>Import the export file exported with /exp.</td>
</tr>
<tr>
<td>idf /key file</td>
<td>Create keyboard translation table file.</td>
</tr>
<tr>
<td>idf /link</td>
<td>Link application imported with /imp.</td>
</tr>
<tr>
<td>idf /link application</td>
<td>Link application imported with /imp.</td>
</tr>
<tr>
<td>idf /pre application</td>
<td>Prepare application for distribution.</td>
</tr>
</tbody>
</table>

### 11.2 Sub-switches

<table>
<thead>
<tr>
<th>Switch or sub-switch</th>
<th>Purpose of switch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General UNIFACE switches</strong></td>
<td>Run application with assignment file.</td>
</tr>
<tr>
<td>/asm</td>
<td>Run application in batch mode.</td>
</tr>
<tr>
<td>/bat</td>
<td>Provide DBMS or network login info.</td>
</tr>
<tr>
<td>/log</td>
<td>Send type n information to message frame.</td>
</tr>
<tr>
<td>/login info</td>
<td>Play back application with TFO_file.</td>
</tr>
<tr>
<td>/ps=</td>
<td>Record application session to TFO file.</td>
</tr>
<tr>
<td>/ifo</td>
<td>Other IDF switches</td>
</tr>
<tr>
<td>idf /exp DBMS and files</td>
<td>Convert data files from DBMS to DBMS.</td>
</tr>
<tr>
<td>idf /exp exported file</td>
<td>Export application to export file, after /pre.</td>
</tr>
<tr>
<td>idf /help</td>
<td>Show this list.</td>
</tr>
<tr>
<td>idf /hel</td>
<td>Show this list.</td>
</tr>
<tr>
<td>idf /imp exported file</td>
<td>Import the export file exported with /exp.</td>
</tr>
<tr>
<td>idf /key file</td>
<td>Create keyboard translation table file.</td>
</tr>
<tr>
<td>idf /link</td>
<td>Link application imported with /imp.</td>
</tr>
<tr>
<td>idf /link application</td>
<td>Link application imported with /imp.</td>
</tr>
<tr>
<td>idf /pre application</td>
<td>Prepare application for distribution.</td>
</tr>
</tbody>
</table>

**table 11-1 continues**
Chapter 12 Assignments

12.1 Priorities

Either:

1. Assignment file defined by user with /asn switch at start-up

Or:

2. Assignment file defined in Application definition form

Higher priority

Or:

3. Assignment file with same name as application and the extension .asn in the current account

Lower priority

4. USYS.ASN in USYS account

Order looked for

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:

\[ \text{part1} \quad \text{part2} \]
\[ \text{part1} = \text{part2} \]

Examples

- Language = USA
- Surname = development
- entity1.concept_schema = $SYS$entity1.*
- file_name = /home/central/param/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 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:

\[ \text{entity.conceptual_schema} = \text{spath : table}[* .extension] \]

Where:

- entity is the entity name used by UNIFACE.
- conceptual_schema is the conceptual schema containing that entity.
- spath : 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 .ims for KMS, nothing for ORACLE and SYBASE (tables in these last two DBMSs do not have an extension).

For example:

- family.dictionary = $SYBASE\_INST\_family.*
- accounts.savings = $SYBASE\_accounts.*
- company.log = $SYBASE\_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:

$\text{path} = \text{DBMS}\_\text{driver}[:\{\text{database}\} | \{\text{username}\} | \{\text{password}\}]$

or:

$\text{path} = \text{DBMS}\_\text{driver}[:\{\text{servername}\} | \{\text{username}\} | \{\text{password}\}]$

or:

$\text{path} = \text{network}\_\text{driver}[:\{\text{network\_node}\} | \{\text{username}\} | \{\text{password}\}]$

or:

$\text{path} = \text{network}\_\text{driver}[:\{\text{network\_server}\} | \{\text{username}\} | \{\text{password}\}]$

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.

$\text{SYDB} = \text{SYB}\_\text{dict}\_\text{berkeley}\_\text{kernel}$

$\text{SRCB} = \text{SYDB}$

$\text{SDEF} = \text{SRCB}:??:7$

$\text{SREP} = \text{SRCB}:7?:7$

$\text{STRBD} = \text{ORA}\_\text{devdir}\_\text{belmont}$

$\text{STZBT} = \text{ORA}$

SHEMA $\text{ORA}$

12.4.2 Path to path

$\text{path}\_1 = \text{path}\_2$

For example:

$\text{SYDB} = \text{SRCB}$

$\text{SDEF} = \text{SDEF}\_\text{PATH}$

$\text{SORA} = \text{SDEF}\_\text{PATH}$

$\text{SDE}\_7 = \text{SVAXI}$

Here follows another example:

$\text{SCREW} = \text{ORA}\_\text{scott}\_\text{tiger}$

$\text{SPEX} = \text{SYDB}\_\text{public}\_\text{berkeley}$

$\text{SQLER} = \text{SYDB}\_\text{prod}\_\text{devdir}\_\text{belmont}$

$\text{STZBT} = \text{STZBT}\_\text{production}$

; test assigned to production after Beta period

Reassigning default paths

For example:

$\text{SYDB} = \text{SIBS}\_\text{SDE}$

Reassigning DBMS

For example:

$\text{SIBS}\_\text{SDE} = \text{RSA}$

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:

<table>
<thead>
<tr>
<th>Profile</th>
<th>Matches which part of name 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*</td>
<td>BCDEFGH.ABC</td>
</tr>
<tr>
<td>ABC* .ABC</td>
<td>DEFGH</td>
</tr>
</tbody>
</table>

**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 ADEFGH.ABC
  - A* = X
- UNIFACE file mapped to BCDEFGH.ABC
  - ABC* .ABC = X* .Y
- UNIFACE file mapped to XDEFGH.Y
  - A3* .A* = X3* .Y
- UNIFACE file mapped to XDEFGH.YC

**12.6 UNIFACE system settings and options**

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Values</th>
<th>Explanation</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>active_field</td>
<td>BOR</td>
<td>Sets active field indicator on</td>
<td>N/A</td>
</tr>
<tr>
<td>Sch.virt</td>
<td>True or false</td>
<td>True = keep in virtual memory (MS/PC-DOS only)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>False = keep control blocks in real memory</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default terminal table name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>INVerse, BRight,UNDERline, BLink or bit value</td>
<td></td>
</tr>
<tr>
<td>default_term</td>
<td>Up to 16 letters</td>
<td>(See 'Explanation')</td>
<td></td>
</tr>
<tr>
<td>def_file</td>
<td>Up to 16 letters</td>
<td>(See 'Explanation')</td>
<td></td>
</tr>
<tr>
<td>def_video</td>
<td>33</td>
<td>Permits use of 16-bit characters with, for example, Kanji</td>
<td></td>
</tr>
<tr>
<td>display</td>
<td></td>
<td>Display table name</td>
<td></td>
</tr>
<tr>
<td>double_width</td>
<td></td>
<td>Maximum number of columns</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>in ORACLE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum number of files simultaneously open</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Swap area</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum size primary page</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Swap area</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum size of input queue</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Where and how pulldown menu bar appears</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Top, INV, BRI</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disabled</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use post-V4 values</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>False</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>This is for the PolyServer only</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enable two-phase commit (only works with DBMSS which support this feature)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Variation</td>
<td>USYS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Name of secondary work file</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(page swap file) (MS/PC-DOS)</td>
<td></td>
</tr>
</tbody>
</table>

**Table 12-2 System settings in assignment files.**

Quick Reference Guide (101075201, 21 September 1992)
12.7 Extensions used for UNIFACE run time and other files

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

Table 12-3 UNIFACE files, non-DBMS.

Example

```
*.aps   /home/centralpark/apps/* .aps
*.prt   /home/centralpark/apps/* .prt
file_name/home/centralpark/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/centralpark/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:

![Assignment paths to network drivers](image)
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 $vox2 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:

$vox2 = dnt:vox2|server|user|password
*.*.demo = $vox2.*
*.*.dict = $vox2.*
$remote_apdict = ora:thickers@land

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

$remote = ora:thickers@tiger
$apdict = ora:thickers?
*.*.dict = $apdict.*
*.*.demo = $remote.*

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:

$vox2 = dnt:vox2|server|user|password
*.*.demo = $vox2.*
*.*.dict = $vox2.*
$remote_apdict = ora:thickers?

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 svax2 and svax3. Both of these paths are accessed with the DECnet network driver.

```markdown
:DEMO.am Assignment file
svax3  dont:svax3?1?1
svax2  dont:svax2?1?1
visite.rbase (svax3:visits,*
  *.rbase (svax2):*,
```

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 svax3 path. The line after that assigns all other entities in this conceptual schema to the svax2 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.

2. USYS.ASN - the global defaults for all UNIFACE applications.

```markdown
1
   either:
       assignment file defined by
       /asn switch at start-up
   or:
       assignment file defined in
       Application definition form
       or:
       application_name. ASN in the
       current account
2
   lower priority
   USYS.ASN in USYS account
```

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.

![Hierarchy of reading and priorities: PolyServer assignment files.](image)

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.

![Combination of assignment files.](image)

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 assignment file is valid for each application session. Typically, it is located in the login directory and has the same name as the application. This file can also be specified with the /ASN file 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 switch when starting the PolyServer. If you use the /ASN file 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 "$PSV_TCP: *` (after setting the environment variables required to run PolyServer by running the `inspsvc` 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

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

*Table 13-1 continues*
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Numeric code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DETAIL</td>
<td>^127^094</td>
<td></td>
</tr>
<tr>
<td>ERASE</td>
<td>^127^008</td>
<td></td>
</tr>
<tr>
<td>FIELD</td>
<td>^255^010</td>
<td></td>
</tr>
<tr>
<td>FIND_TEXT</td>
<td>^127^150</td>
<td></td>
</tr>
<tr>
<td>FIRST</td>
<td>^255^067</td>
<td></td>
</tr>
<tr>
<td>FIRST_OCC</td>
<td>^127^037</td>
<td></td>
</tr>
<tr>
<td>FIRST_TEXT</td>
<td>^127^129</td>
<td></td>
</tr>
<tr>
<td>FONT</td>
<td>^127^151</td>
<td></td>
</tr>
<tr>
<td>FRAME</td>
<td>^127^089</td>
<td>Define frame</td>
</tr>
<tr>
<td>HELP</td>
<td>^127^092</td>
<td></td>
</tr>
<tr>
<td>HOME</td>
<td>^127^022</td>
<td>Cursor at window top</td>
</tr>
<tr>
<td>INSERT</td>
<td>^255^071</td>
<td>Insert (removed)</td>
</tr>
<tr>
<td>INSERT</td>
<td>^255^074</td>
<td></td>
</tr>
<tr>
<td>INS_CHAR</td>
<td>^127^184</td>
<td></td>
</tr>
<tr>
<td>INS_FILE</td>
<td>^127^180</td>
<td>Insert file</td>
</tr>
<tr>
<td>INS_LINE</td>
<td>^127^182</td>
<td></td>
</tr>
<tr>
<td>INS_OCC</td>
<td>^127^043</td>
<td></td>
</tr>
<tr>
<td>INS_OVER</td>
<td>^127^146</td>
<td>Insert selected block</td>
</tr>
<tr>
<td>INS_SELECT</td>
<td>^127^195</td>
<td></td>
</tr>
<tr>
<td>INS_TEXT</td>
<td>^127^177</td>
<td></td>
</tr>
<tr>
<td>INS_WORD</td>
<td>^127^183</td>
<td>Insert/Overstrike</td>
</tr>
<tr>
<td>ITALIC</td>
<td>^127^148</td>
<td></td>
</tr>
<tr>
<td>KEY_HELP</td>
<td>^127^072</td>
<td>Keyboard layout help</td>
</tr>
<tr>
<td>LAST</td>
<td>^255^068</td>
<td></td>
</tr>
<tr>
<td>LAST_OCC</td>
<td>^127^038</td>
<td></td>
</tr>
<tr>
<td>LAST_TEXT</td>
<td>^127^128</td>
<td></td>
</tr>
<tr>
<td>LINE</td>
<td>^255^004</td>
<td></td>
</tr>
<tr>
<td>MENU</td>
<td>^127^101</td>
<td>Message frame</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>^127^093</td>
<td>Next mode</td>
</tr>
<tr>
<td>NEXT</td>
<td>^255^065</td>
<td></td>
</tr>
<tr>
<td>NEXT_CHAR</td>
<td>^127^142</td>
<td></td>
</tr>
<tr>
<td>NEXT_FIELD</td>
<td>^127^046</td>
<td></td>
</tr>
<tr>
<td>NEXT_LINE</td>
<td>^127^136</td>
<td></td>
</tr>
<tr>
<td>NEXT_OCC</td>
<td>^127^039</td>
<td></td>
</tr>
<tr>
<td>NEXT_TEXT</td>
<td>^127^163</td>
<td></td>
</tr>
<tr>
<td>NEXT_WORD</td>
<td>^127^140</td>
<td></td>
</tr>
<tr>
<td>OCCURRENCE</td>
<td>^255^011</td>
<td></td>
</tr>
</tbody>
</table>

* table 13-1 continues

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Numeric code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCC_WINDOW</td>
<td>^255^015</td>
<td>Previous mode</td>
</tr>
<tr>
<td>PREV</td>
<td>^255^066</td>
<td></td>
</tr>
<tr>
<td>PREV_CHAR</td>
<td>^127^143</td>
<td></td>
</tr>
<tr>
<td>PREV_FIELD</td>
<td>^127^047</td>
<td></td>
</tr>
<tr>
<td>PREV_LINE</td>
<td>^127^137</td>
<td></td>
</tr>
<tr>
<td>PREV_OCC</td>
<td>^127^040</td>
<td></td>
</tr>
<tr>
<td>PREV_TEXT</td>
<td>^127^162</td>
<td></td>
</tr>
<tr>
<td>PREV_WORD</td>
<td>^127^141</td>
<td></td>
</tr>
<tr>
<td>PRINT</td>
<td>^127^088</td>
<td></td>
</tr>
<tr>
<td>PRINT_ATTRIBUTES</td>
<td>^127^099</td>
<td>Define find profile</td>
</tr>
<tr>
<td>PROFILE</td>
<td>^127^087</td>
<td></td>
</tr>
<tr>
<td>PULLDOWN</td>
<td>^127^086</td>
<td></td>
</tr>
<tr>
<td>QUICK_ZOOM</td>
<td>^127^096</td>
<td></td>
</tr>
<tr>
<td>QUIT</td>
<td>^127^010</td>
<td></td>
</tr>
<tr>
<td>REFRESH</td>
<td>^127^087</td>
<td>Delete character</td>
</tr>
<tr>
<td>REMOVE</td>
<td>^255^073</td>
<td>Remove file</td>
</tr>
<tr>
<td>REM_CHAR</td>
<td>^127^172</td>
<td>Remove selected block</td>
</tr>
<tr>
<td>REM_FIELD</td>
<td>^127^166</td>
<td></td>
</tr>
<tr>
<td>REM_FILE</td>
<td>^127^192</td>
<td></td>
</tr>
<tr>
<td>REM_LINE</td>
<td>^127^167</td>
<td></td>
</tr>
<tr>
<td>REM_OCC</td>
<td>^127^045</td>
<td></td>
</tr>
<tr>
<td>REM_SELECT</td>
<td>^127^194</td>
<td></td>
</tr>
<tr>
<td>REM_WORD</td>
<td>^127^189</td>
<td></td>
</tr>
<tr>
<td>RESET_SELECT</td>
<td>^127^196</td>
<td></td>
</tr>
<tr>
<td>RETRIEVE</td>
<td>^127^056</td>
<td></td>
</tr>
<tr>
<td>RETRIEVE_SEQ</td>
<td>^127^003</td>
<td></td>
</tr>
<tr>
<td>RUB_CHAR</td>
<td>^127^173</td>
<td>Backspace (delete character to left of cursor)</td>
</tr>
<tr>
<td>RULER</td>
<td>^127^081</td>
<td></td>
</tr>
<tr>
<td>SAVE</td>
<td>^127^179</td>
<td>Put selected text in $eiblk buffer</td>
</tr>
<tr>
<td>SELECT</td>
<td>^127^193</td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>^127^097</td>
<td></td>
</tr>
<tr>
<td>STORE</td>
<td>^127^011</td>
<td></td>
</tr>
<tr>
<td>SWITCH_KEY</td>
<td>^127^100</td>
<td>Switch to alternate keyboard</td>
</tr>
</tbody>
</table>

* table 13-1 continues
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Numeric code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>^255^009</td>
<td></td>
</tr>
<tr>
<td>TEXT_WINDOW</td>
<td>^255^014</td>
<td></td>
</tr>
<tr>
<td>TOP_OF_FORM</td>
<td>^127^020</td>
<td>Cursor at form top</td>
</tr>
<tr>
<td>UNDERLINE</td>
<td>^127^149</td>
<td></td>
</tr>
<tr>
<td>USER_KEY</td>
<td>^127^091</td>
<td></td>
</tr>
<tr>
<td>VIEW</td>
<td>^127^073</td>
<td></td>
</tr>
<tr>
<td>WORD</td>
<td>^255^003</td>
<td></td>
</tr>
<tr>
<td>ZOOM</td>
<td>^127^095</td>
<td></td>
</tr>
</tbody>
</table>

*Table 13-1: Character codes for use in macro statements.*