The WINDOW Options

Window options are available to control various aspects of the window. The window options must be coded following the WINDOW statement and the WINDOW body.

1. Optional Clauses

The following window options are available:

[*window help-text* [$TR-key]]
[SEQUENCE *id1* [*Clear-opt*], *id2* [*Clear-opt*]]
[EDT] [ADD] [INS] [ENQ] [SEL] [MNT] [DEL] [UDL]
[REPEAT [UNTIL [NEXT | CURRENT RECORD]]]
[LOCK | NOLOCK | XNOLOCK]
[SCROLL [REVERSE] *n1* [BY *n2*] [SPLIT *n3* OFFSET *n4*]]
[LINE *l1* [*l2* ... *l8*]]
[BOX *line col width depth* [*title*]]
[BASE AT *line col*]
[ENABLE [NXT] [ABO]]
[DISABLE [SKP] [CLR] [HME]]
[AUTOPGE | AUTOBPG]
[TYPE NORMAL|HEADER|INFO|ERROR|ENQ|SELECT|TRAILER]
[*LI CL Longname LI CL Name* Pic *Options*...]
[*LI CL* Text BTN U*nn*]
[POP-UP]
[QREM]
[SBOX]
[REVSUBKEY]

2. Window Help-text

Window help-text lines have the backslash character \ as the first significant character on the line, must be contiguous, and follow immediately after the WINDOW statement. Key-top names may be embedded in the help-text by coding $TR-key where key is any of the function-key mnemonics listed. For example, to display the help-text "Key End for next window" you should code "\Key $TR-NXT for next window". The help window is displayed when <HLP> is keyed twice (i.e. <HLP><HLP>).

The window help lines will not be shown when <HLP> is keyed if external help from a help file is available for the window

3. Sequence Clause

Each window in the Window Division may be executed individually under the control of the Procedure Division. Chains of windows can be constructed, however, in much the same way as chains of frames. Each window may contain a sequence statement that specifies the next window to execute following successful or unsuccessful completion.

By using the sequence statement, several windows may therefore be executed before control is finally returned. Any of the windows in the sequence may actually return control, provided the sequence statement permits this. If it is important to know **which**

window actually returned control, this can be achieved by setting a flag within the routines section.

A chain of windows will therefore complete successfully or unsuccessfully, and this will cause control to be returned. If the window was invoked using the ENTER statement within the Procedure Division, control will be returned to the statement immediately following it. If unsuccessful completion occurred, this will be indicated by an exception condition, which may be trapped by an ON EXCEPTION statement within the Procedure Division. If no Procedure Division was coded, control will be returned to the frame manager.

The sequence clause is coded:

SEQUENCE *id1* [*Clear-opt*], *id2* [*Clear-opt*]

where *id1* is the window-id to be entered on unsuccessful completion of the window, and *id2* is the window-id to be entered following successful completion of the window. The keyword EXIT may be coded for either *id1* and/or *id2*, and this causes the window manager to return control on completion, instead of executing a further window.

Clear-opt specifies a window clearing action to be taken on completion of the window and may be one of the following:

CLR Clear Screen. The screen is totally cleared, leaving only the screen header displayed.

CLW Clear Window. The window is removed from the screen. If the window is a POP-UP, the prior image is re-displayed. Otherwise the area occupied by the window is over-written with spaces in the window background attribute.

CLD Clear Data. Data displayed within the window is cleared. If the sequence statement is omitted, an exit will take place both on successful and unsuccessful completion, and no clearing action will take place.

For example, coding:

SEQUENCE W1, W3

will cause control to be transferred to window W1 on unsuccessful completion (backward exit), and otherwise to window W3 (forward exit). The keyword "EXIT" may also be coded in this statement, and this causes control to be returned. Omitting the sequence statement is the same as coding:

SEQUENCE EXIT, EXIT

which causes control to be returned under all circumstances.

4. Mode Enabling Clauses

This section describes the following optional processing modes:

EDT
ADD
INS
ENQ
SEL
MNT
DEL
UDL

If **none** of the above clauses is coded, defaults are allocated as follows. If the USING clause has been coded in the window statement (i.e. the window operates on a target record type) all clauses other than EDT and UDL are enabled. Otherwise only the ADD clause is enabled. These defaults are over-ridden by coding any of the above clauses that are described below:

4.1 The EDT Clause

The **EDT** clause enables edit mode, used only on windows that do not operate on a target record type and should not be used in scrolled windows. It allows the data fields processed by the window to be pre-initialised using the display verb and entry into the window then allows the operator to edit the displayed fields as a whole. This clause is typically used in windows, which accept run-time parameters (e.g. those required in a print program). Coding EDT automatically enables ADD mode.

4.2 The ADD Clause

The **ADD** clause enables add mode, normally entered when the cursor is positioned on a blank record. This allows the operator to enter the fields on the record, on completion of which a new record is written to the database. Unless this mode is enabled addition of new records cannot take place, so only existing records can be processed by the window.

4.3 The INS Clause

The **INS** clause enables insert mode, which allows a new record to be inserted before an existing record displayed in the window. When the operator keys <INS>, the window is scrolled apart to create a new blank record display area into which the new record can be inserted. Since the insert instruction requires existing records to be displayed, the INS clause also enables ENQ and DSP modes. ADD mode is also enabled by this instruction.

4.4 The ENQ Clause

The **ENQ** clause enables enquiry and display modes. Enquiry mode is activated whenever a database search is initiated, and is therefore required in order to display existing records within the window. This mode is also entered explicitly when <ENQ> is keyed. If the database search is successful, display mode is activated in order to display the retrieved records.

If the ENQ clause is not coded, the user will be unable to retrieve and display records from the database, and is therefore restricted simply to adding new records. Since enquiry mode requires a target record type, the window statement must contain the USING clause.

4.5 The MNT Clause

The **MNT** clause, when coded, the operator may key <RET> to select the current record for maintenance. Unprotected fields on the record may then be edited, on completion of which the record is re-written to the database. Since maintenance operates on existing records, ENQ and DSP modes are automatically enabled.

4.6 The SEL Clause

The **SEL** clause allows an existing record to be selected from the window. This clause is coded when a window is used for selection purposes only, an example of this being the selection of a customer prior to invoice entry. It is used instead of the MNT clause when no maintenance is to take place on the selected record. The window manager suppresses record editing and the re-write of the record that would otherwise take place. Since record selection operates on existing records, ENQ and DSP modes are automatically enabled. Unless the SEL clause is coded, the operator is able to position the cursor on a different record within the display, but is unable to select it.

4.7 The DEL Clause

The **DEL** clause allows the operator to select a record for deletion by placing the cursor on the required record, which is then deleted by keying <DTE>. If this clause is omitted, record deletion is disabled. Since deletion operates on existing records, ENQ and DSP modes are automatically enabled.

4.8 The UDL Clause

The **UDL** clause allows the operator to undelete the last deleted record by keying <UDL>. Coding this clause causes the compiler to create an area within the frame that will contain a copy of the last deleted record. Keying <UDL> activates ADD or INS mode, each field on the record being moved from the saved area instead of being accepted from the operator. The UDL clause automatically activates ADD, INS, ENQ, DSP and DEL modes.

4.9 Programming Notes

If the window does **not** have a target record type (i.e. the USING clause was not specified in the WINDOW statement) the window may only operate in ADD and EDT modes. Coding any of the clauses INS, ENQ, SEL, MNT, DEL, UDL will therefore result in an error during compilation.

The MNT and SEL clauses are mutually exclusive, and may not be coded for the same window. The SEL clause means that MNT mode is **not** to be entered following selection of a record. Coding SEL alone means that the window may only be used to do an enquiry and select a particular record. This option is often used to set up a controlling record for use by a subsequent dependent window (e.g. to select a given customer prior to processing that customer's invoices in a subsequent window).

If ENQ is the window's only valid mode, the processing cycle described earlier in this chapter can never complete, since the window simply stays "stuck" in enquiry mode. However, if the REPEAT UNTIL NEXT clause was coded, the operator will be able to use the <NXT> key to indicate successful completion. Otherwise, **successful completion cannot occur**.

5. Record Status Controlling Clauses

The following clauses control record lock status and window termination. They are coded:

REPEAT [UNTIL [NXT | CURRENT RECORD]]
LOCK | NOLOCK | XNOLOCK

5.1 The REPEAT option

The **REPEAT** option causes the window to loop until a terminating condition is reached. If the option is not coded, successful completion will be returned following a single processing cycle. When the REPEAT clause is coded on its own, the processing cycle will continue indefinitely, and no successful exit is therefore possible.

5.2 The REPEAT UNTIL NXT option

The **REPEAT UNTIL NXT** option enables the <NXT> key, and allows the operator to indicate completion of the window at any time.

5.3 The REPEAT UNTIL CURRENT RECORD option

The **REPEAT UNTIL CURRENT RECORD** option specifies that a current record is required before successful completion is permitted. When coded, this option ensures that a valid target record is contained in the I/O channel, and this record will be locked in accordance with the locking options discussed in the following section. This option must therefore be coded whenever the status of the I/O channel on successful completion is important.

Note that this option has no effect on the operation of the <ABO> and <BCK> keys, which may be used to force unsuccessful completion of the window at any time. The status of the I/O channel on **unsuccessful** completion of a window is therefore undefined under all circumstances.

5.4 The LOCK, NOLOCK and XNOLOCK options

The **LOCK** and **NOLOCK** options are used to specify the lock level required when a record is selected without further processing. When a record is added, deleted, or maintained a full (exclusive) lock is always placed on the record. The locking options are therefore only used with the SEL option described in section 6.6.2.4.

When the SEL option is coded, a selected record will normally be delete-protected (non-exclusively locked) so that it cannot be deleted by another concurrently executing frame. Coding the LOCK option causes a full lock to be placed on the record. Coding the NOLOCK option suppresses locking of the target record.

Note that the record lock, specified using the above options or as defaulted, is normally released on completion of the window. It will only be retained if REPEAT UNTIL CURRENT RECORD is coded. The lock status of the I/O channel following unsuccessful completion is undefined.

The XNOLOCK option is a very specialised version of the NOLOCK option ($COMPILE only) and must be used with great care. It does no record locking on the records in the window even if adding, deletion and maintenance is allowed. It is usually useful when used in conjunction with the EXIT WITH 5 option from the R-REWRITE routine in

maintenance windows where no fields in the record can be updated but other fields may be.

6. The SCROLL statement

The SCROLL statement is used to define a scrolled region within the window. It is coded:

SCROLL [REVERSE] *n1* [BY *n2*] [SPLIT *n3* OFFSET *n4*]

where *n1* is the number of records in the scrolled area, each of *n2* contiguous lines. The number of columns in which the window is arranged for vertical split scrolling is defined by *n3* and the offset between these columns by *n4*.

If the REVERSE clause is used the then the records in the scrolled area will be displayed in reverse order.

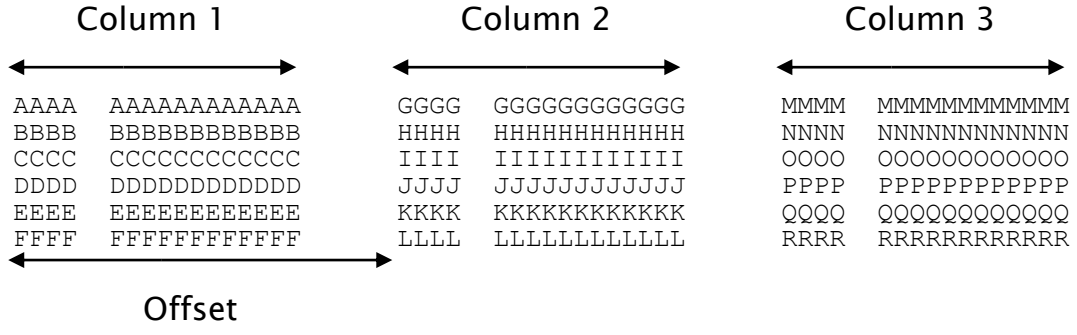
The scroll statement specifies that the **data fields** defined later within the window are to be scrolled, thus allowing multiple records to be displayed within the window at once. Note that **text fields are never scrolled**, and are therefore always displayed at the coded line and column positions.

The number of records to be displayed is defined by variable *n1*. Normally, each record will take up only one line within the display, but multi-line records can be specified by using the BY *n2* clause. Thus, if the scrolled region is to contain eight records, each using two physical lines on the screen, the following would be coded:

SCROLL 8 BY 2

The scrolled region would therefore occupy sixteen lines (8 x 2).

The Window Manager allows this scrolled region to be split vertically into several columns, and this is achieved by the SPLIT clause. The number of columns into which the scrolled region is to be split is specified by variable *n3*, and the vertical displacement between the two columns is specified in characters by variable *n4*. For example, consider a scrolled region split into three columns:



This scrolled region contains eighteen record display areas (areas AAAA to RRRR), each taking up one line. The region has been split into three columns, and the offset between the columns is twenty characters. This is coded:

SCROLL 18 BY 1 SPLIT 3 OFFSET 20

The BY 1 clause could be omitted, since this is the default.

The SCROLL statement applies to all the following data fields coded within the window, except those with the NSC option - see Section 6.6.4.2. It is normal for a window to contain both scrolled and non-scrolled fields, and these do not necessarily have to be coded in order. For example, the window may have a few scrolled fields, followed by a few non-scrolled fields, and ending again with scrolled fields.

7. LINE and BOX Statements

All windows are displayed with a box around the outer extremities of the displayed text and data items. The dimensions of the box are calculated by the Speedbase compiler so that the top and bottom lines of the box are immediately above and below the first and last used display lines respectively. The vertical lines of the box are normally displayed two characters before and after the first and last used column respectively. Fields should therefore start at, or after, character position three on the screen, and should not be placed higher than line two, or line three if a screen header is also displayed.
The line and box statements allow lines and boxes to be drawn within the window.

7.1 The LINE Statement

The **LINE** statement specifies horizontal lines that will be joined to the vertical lines of the window's box. It is coded:

LINE *l1* [*l2* ... *l8*]

where *l1* to *l8* are up to eight line numbers at which lines are to be drawn.

7.2 The BOX Statement

The box statement specifies the dimensions and heading of a box within the window. It is coded as follows:

BOX *line col width depth* [*heading*]

where *line* and *col* are the line and column numbers of the top left-hand position of the box; *width* is the width of the box in columns, and *depth* gives the vertical dimension in lines. The optional *heading* is a literal text string that is displayed over the top line of the box. Note that all of these items must be coded as literals.

Depth is expressed as the arithmetic difference between the top and bottom lines of the box. Thus if the top line of the box is 4, with the bottom line at 7, depth will be 3 (even though the box will physically occupy 4 lines). Box width is similarly calculated. The minimum width and depth of a box is 2, meaning a box must enclose no less than a single character.

Note that the line and column co-ordinates are affected by the Window Base (as coded by the BASE clause), even if the BASE clause is coded after the BOX.

The BOX statement should not be used to enclose a group of buttons. A special group button version is available for this. (See WINDOW Body).

8. The BASE AT Statement

The **BASE AT** statement specifies the window position offset. It is coded:

BASE AT *line col*

where *line col* is the line and column position of the top left-hand corner of the window (e.g. coding BASE AT 9 20 places a window 8 lines down and 19 characters towards the right of the screen).

9. The ENABLE Statement

The ENABLE statement is used to enable the <NXT> and <ABO> functions. Coding:

ENABLE ABO

has the effect of enabling the Abort function, which allows the operator to abort a chain of windows. This function avoids the user having to key <BCK> several times in order to terminate a series of windows.

Windows supporting MNT mode normally cause the operator to enter maintenance mode on selection, before the next window is entered. Coding:

ENABLE NXT

allows the operator to select the current record using the <NXT> key while in display mode. The window manager then automatically skips through all the fields on the record just as if the operator had keyed <RET> in response to each field. This feature is useful when processing master/servant windows, when maintenance is not necessarily carried out within the initiating window.

10. The DISABLE Statement

The DISABLE statement allows the <SKP>, <CLR> and <HME> options to be disabled. It is coded:

DISABLE [SKP] [CLR] [HME]

The statement simply removes these function keys during processing of the window. This is useful in simple windows such as menus, where functions such as <CLR> could cause confusion.

11. The Auto-page Statements

The AUTOPGE and AUTOBPG statements cause a page or back-page operation to occur when the window is initially entered, as if <PGE> or <BPG> had been keyed. These options are useful, for example, in enquiry windows, where it is often convenient to display the first or last page of records on entry. The options are mutually exclusive, and operate only when a clear (i.e. empty) window is entered.

12. The TYPE Statement

This TYPE statement defines a window as belonging to one of several classes. Each class has its own set of options in the GX.INI file that controls all aspects if the window display attributes.

The window types available are

NORMAL	The default is the type statement is omitted
HEADER	A batch header r similar summary type window
INFO	An information or help window
ERROR	An error window
ENQ	An enquiry only window
SELECT	A standard search window
TRAILER	Trailer type window

It is possible to modify the settings for the various types of window in the GX.INI file in the GX directory on the system. Here is an example:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;
;
;           Application window colour settings
;
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;

ApplicationWindowBackground=192,192,192
ApplicationScrolledBackground=255,255,255
ApplicationLabelText=0,0,0
ApplicationFieldBackground=255,255,255
ApplicationFieldGreyedBackground=212,212,212
ApplicationFieldGreyedText=0,0,0
ApplicationFieldHighlightBackground=0,127,0
ApplicationFieldHighlightText=255,255,255
ApplicationFieldWarningBackground=127,127,0
ApplicationFieldWarningText=0,0,0
ApplicationFieldErrorBackground=255,0,0
ApplicationFieldErrorText=255,255,255
ApplicationFieldCurrentRecordBackground=0,0,255
ApplicationFieldCurrentRecordText=255,255,255
ApplicationFieldOtherRecordMusicBackground=196,253,200
ApplicationFieldAcceptBackground=252,254,188
ApplicationFieldAcceptText=0,0,0
For each other window type a set of ini file settings can be specified to override
the above.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                     ;
;        Select (Search/Browse) window colour settings                ;
;                                                                     ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

SelectFieldGreyedBackground=212,212,212
SelectFieldBackground=212,212,212
SelectFieldOtherRecordMusicBackground=196,253,200

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                     ;
;                     Info window colour settings                     ;
;                                                                     ;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

InfoWindowBackground=94,255,249
InfoFieldGreyedBackground=84,248,239
InfoFieldBackground=84,248,239
InfoFieldOtherRecordMusicBackground=196,253,200

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                    ;
;               Error window colour settings                         ;
;                                                                    ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

ErrorWindowBackground=250,204,194
ErrorFieldGreyedBackground=250,204,194
ErrorFieldOtherRecordMusicBackground=196,253,200
```

13. Clearing controls

These controls are only relevant on text mode screens.

The POP-UP and QREM clauses are used to control the way a window is cleared from the screen. The POP-UP clause causes the screen image under the window to be saved when it is activated. When the window is cleared using the CLW option (see below) or the CLEAR *window* statement, this image is re-displayed, thus resetting the screen to its prior state.

The QREM clause is used to improve clearing performance. When an ordinary window (i.e. not a POP-UP) is removed using the CLW option or CLEAR *window* statement, the area underneath it is cleared by displaying spaces. This can take some time, especially if the window is large. The QREM clause causes the window to be removed using the clear-to-end-of-line facility, which operates much faster, but also has the effect of clearing the area to the right of the window. This option should therefore only be used when the area to the right of the window is otherwise unused.

14. The SBOX Statement

The **SBOX** statement causes the box to be drawn immediately to the left and right of the first and last used column positions, thus creating a slightly narrower box than would normally be produced. This allows data and text to be displayed from column position two as opposed to column three. If used, the SBOX statement must be coded prior to any text or data item definitions.

This statement is only relevant for text mode screens.

15. The REVSUBKEY Statement

The **REVSUBKEY** statement can only be used in a window that contains a DEPENDENT ON..MATCH clause. It causes the remainder of the key, not mentioned in the DEPENDENT on clause, to be displayed in reverse order in the window within the match table entry.

For example, if you have a table which contains an index with two key fields xxBRANCH and xxDATE with records as follows:

```
BRANCH (PIC 9(2) C)     DATE
1                       1/1/07
1                       2/2/07
2                       1/2/07
```

```
2                       2/2/07
3                       1/2/07
3                       2/2/07
```

If you want to review the two branches 1 and 3 you would require a window with a clause as follows:

```
WINDOW W1 USING XX DEPENDENT ON (Z-BRANCH MATCH Z-KTAB)
```

where Z-BRANCH is a PIC 9(2) C field and Z-KTAB is defined as follows:

```
01    Z-KTAB
  02  FILLER             PIC 9(2) C        * Terminator
                         VALUE -1
  02  FILLER OCCURS 2    PIC 9(2) C
                         VALUE 1
                         VALUE 3
  02  FILLER             PIC 9(2) C        * Terminator
                         VALUE -1
```

A scrolled window with this clause and the relevant fields would show the following information in this order:

```
BRANCH                  DATE
1                       1/1/07
1                       2/2/07
3                       1/2/07
3                       2/2/07
```

To review the entries for the two branches in ascending order looking at the latest entries first, you need to add the **REVSUBKEY statement** to the window. The entries will then appear as follows:

```
BRANCH                  DATE
1                       2/1/07
1                       1/2/07
3                       2/2/07
3                       1/2/07
```