Overview

SirForms allows you to create and run sets of linked, interactive screens for data entry, retrieval and update. A complete set of screens, known as a form, is defined using commands. There are commands for defining what variables are on each screen, how they are displayed and edited, how the screen is to look, and for linking screens together.

A form can be re-compiled every time it is used or the compiled version of the form can be saved as a `.frm` file.

Once a form has been developed, it can be used by many people for data entry or for querying data. (See Using an existing form.)

SirForms can be run as an independent module or from the main SIR/XS menu.

When SirForms is run, a number of parameters can be specified. At a minimum, the name of the input file containing the form must be specified. The input can be either a text file to be compiled (`.cmp`) or a pre-compiled form (`.frm`). If the input file is not specified, you are prompted for an input file to be compiled.

SirForms commands are created with a standard text editor. A Default Form can be generated for a database and can be used without any modifications to create new records, to modify or delete any existing records and to search the database for records meeting particular criteria.

When SirForms compiles the commands, if there are any errors, a message is displayed and errors are listed either to the specified listing file or to the default `SirForms.err`. If any changes are made to the database that affect a compiled form, a warning is issued that the form is out of revision and must be re-compiled.

If 'journalling' is set on for a database, SIR/XS take copies of any database updates as they happen through SirForms. If the database becomes corrupted or damaged, then a backed up version can be restored and the journal updates applied to bring the database or tabfile back to the current level without re-entering data. See Check Database for a further discussion of recovery.

Form Structure

A form definition consists of a set of commands, each of which has numerous clauses. Each command may continue on as many lines as necessary and it is usual practice to specify each clause of a command as a continuation line. To continue a command to the next line, put the continuation character "-", at the end of a line.

A Form Definition starts with the FORM command. This specifies the name of the form, the name(s) of any database(s) or tabfile(s) to be used, any defaults that apply to the entire form, security specifications and defines any temporary variables for use in the form. The entire Form Definition is terminated by the `ENDFORM` command.

The FORM command is followed by commands defining one or more screens. A screen is a display on the execution window which is a fixed 30 rows by 80 characters text style screen window. The window is cleared between each screen.

A form can contain any number of screens; it must contain at least one screen definition.

Definition of a screen starts with one of the commands MENU,RECORD,TABLE. Menu screens are independent of any database record or table. They can act as a table of contents with a choice as to where to go next. Menu Screens can also be used to display or enter data into temporary variables. Record and Table screens relate to database records or tables. These screens can be used to display data and can be used to enter new data and modify existing data.

Within each screen, further commands are used to describe the screen. The most common of these is the FIELD command. This displays individual variables and allows the user to enter new data. Default formats and edit rules from the data dictionary are applied automatically. There are numerous clauses on this command to extend the edit rules and alter the position or format of the data.

Definition of a screen is terminated by the appropriate END command (ENDMENU, ENDRECORD or ENDTABLE).

Screens are linked together with the CALL command which passes control from one screen to another. This can be done under user control or automatically at given points in the form.

All the data quality control and schema information defined in the data dictionary (such as Valid Values, Variable Ranges, data type, data size) is applied automatically within a form; this information does not have to be specified again.

Comments can be added to a form definition. Comment lines begin with an asterisk "*". The whole line is treated as a comment.

If the same commands or partial commands are used repeatedly, synonyms for these can be defined, which can simplify maintenance and development. Synonyms are frequently used for video attributes which are common to many fields, edit checks across fields and standard `CALL` keywords.

Parts of forms can be held on separate files and included into other forms definitions. This technique can be used where the same screen or menu occurs in many forms, or can be used with a standard set of synonyms to ensure consistency of usage across all forms.

A form has a structure similar to the following (The indentation is for readability only) :

```
FORM form_name
  MENU menu_screen_name
     CALL record_screen_name
     CALL table_screen_name
  ENDMENU
  RECORD record_screen_name
     FIELD field_name
     FIELD field_name
  ENDRECORD
  TABLE table_screen_name
     FIELD field_name
     FIELD field_name
  ENDTABLE
ENDFORM
```

A very simple form definition might be:

```
form COMPANY -
     database COMPANY -
     password COMPANY -
     security HIGH,HIGH -
     labels 20 -
     root MAINMENU
  menu MAINMENU
    call EMPLOYEE
    call OCCUP
    call REVIEW
  endmenu
  record EMPLOYEE
    generate
  endrecord
  record OCCUP
    generate
  endrecord
  record REVIEW
    generate
  endrecord
endform
```

Specification Levels

There are levels within a form for specification of common field attributes and specifications concerning fields can be set at any level and apply to all of the fields within that level. These levels are:

```
FORM
  SCREEN (RECORD, TABLE or MENU)
     PAGE
       GROUP
         FIELD
```

For example, to highlight data on all of the fields for one particular record, specify the `DATA HILITE` clause at the `RECORD` level.

Higher level specifications act as defaults for all lower levels. These can be overridden for a particular field or group of fields by specifying the clause at that level.

Names

Names are normal SIR/XS names. Standard names are 1 to 32 characters long with no spaces. The first character must be alphabetic. Characters can be letters, digits or four characters ($, #, @,_). Names are translated to upper case so uppercase and lowercase letters are equal.

If you wish to use a name which does not conform to these rules, enclose the name in curly braces {} as per other SIR/XS references to non-standard names. (Note: for compatability with previous versions the SIR/XS FORMS module also supports the use of double quotes "" as delimiters on input for non-standard names.)

A non-standard name can contain blanks or use lowercase letters.

Expressions can be used at many points in commands, typically in IF or COMPUTE clauses. An expression is a set of named variables, constants or functions which eventually resolves to a single value. For example, `SALARY*1.1` is an expression which multiplies salary by 1.1. see COMPUTE commands.

Default form

It is very easy to create a default form from within the main SIR/XS menu. (See Default Form.)

To create a default form, start SIR/XS and log on to the database. For the example `COMPANY` database, the name of the database is `COMPANY`, password `COMPANY`, read security is `HIGH`, write security is `HIGH`. When you have logged on:

Select the `Database` menu option

Select `Write schema`
Enter a file name, for example, `COMPANY.cmp` and turn on the `Forms Definition` option. Execute the dialogue.

The default form is now in the specified file. You can run the default form immediately without making any changes or you can edit the file.

Commands

The following commands can be used:

FORM

Begins the definition of a form. Defines databases, tabfiles, local variables and security. Local variables can be referenced at any point in the form.
RECORD
Begins a record screen set of commands which can reference database variables from one record type.
TABLE
Begins a table screen set of commands which can reference variables from one table.
MENU
Begins a menu screen that contains only calls to other screens and temporary variables. It cannot reference a record or table.
HELP
Begins a help screen containing text. This is a display only screen.
PAGE
Defines a page of fields within a screen. A page refers to a physical screen of data.
GROUP
Defines a group of fields within a page.
FIELD
Defines a field. Fields are always in a screen and may be in pages and/or groups.
GENERATE
Defines a default set of fields from the schema.
CALL
Passes control from one screen to another.
COMPUTE
Calculates a variable.
IF
Conditionally calculates a variable.
LOOKUP
Finds a record from which to retrieve data or to verify a value or expression.
AT
Sets the cursor position on the screen.
TEXT
Displays a text string.
DRAW
Draws a line or box.
ENDxxxxxx

Defines the end of a previous group of fields. Commands are `ENDFORM`, `ENDRECORD`, `ENDTABLE`, `ENDMENU`, `ENDHELP`, `ENDPAGE` and `ENDGROUP`.

Security

The security levels defined in the database schema are applied in SirForms for access to the database, record types or variables that have security levels defined. The database password and security passwords must be specified before access to the database is permitted.

Further security control can be defined to regulate the activities or functions a user may perform while using the form. You can restrict the ability of a particular form to read existing data, write new data, delete data, etc. By default, virtually all activities are allowed. Permission is denied for an activity on a form or screen by adding the prefix `NO` to the name of the activity. For example, to disallow all changes to the database, add the `NO` prefix to the `WRITE` activity.

```
FORM COMPANY .... ACTIVITIES NOWRITE ....
```

The `ACTIVITIES` clause on the `FORM` command sets the permissions for all the screens in the form. Activities can be specified at a screen and further restrict what can be done at that screen. If `USERS GROUP ACTIVITIES` are specified on the `FORM` command, the specifications are in effect for the entire form. Permissions that have been denied at a higher level cannot be granted to a user group at a lower level of the Form Definition.

The `MODIFY/NOMODIFY` applies to fields and can be specified at the individual `FIELD`, `GROUP` and `PAGE` levels. The `ACTIVITIES` clause specifies the restrictions which apply to anyone using the form.

If several people use the same form, activities can be allowed or disallowed for different users or groups of users. The various users of a form can be split into **user groups**. Each group has a name and optional password and a set of permitted and prohibited activities. These are specified as clauses on the `FORM` command. For example:

```
FORM COMPANY  -
      DATABASE COMPANY PASSWORD COMPANY SECURITY HIGH, HIGH -
      LABELS 20 ROOT MAINMENU -
      USERS -
       GROUP SUPER / DOALOT ACTIVITIES NONEWCASES -
       GROUP CLERK / DOSOME ACTIVITIES NOMODIFY
```

When a user accesses this form, they are prompted for a group name and password. The `NONEWCASES` on the `ACTIVITIES` clause for `SUPER` means that this group cannot create new cases, however all other activities are allowed. The `CLERK` group cannot modify existing data.

Help

Help is provided from the schema for users who need information about a particular field. The default help text consists of the variable name, label, input format, value labels, storage requirements and missing value(s).

Additional help can be provided as part of the form as a separate screen of text or as a line displayed in the Status area. A line of help text for a field can be displayed in the Status Area with the `HELP` clause on a `FIELD` command:

```
FIELD BIRTHDAY -
  PROMPT 'Date of Birth (Enter as MMM DD, YYYY)' -
  TYPE DATE 'Mmm DD, YYYY' -
  HELP 'For example: May 24, 1997'
```

If a longer description is required, a separate screen can be defined as an alternative to the default Help screen. Specify the name of the Help screen on the HELP clause. Define a help screen with the text of the help to be displayed. Help screens can contain `DRAW`, `TEXT` and other visual enhancement commands. For example, to specify a Help screen for the Social Security Number field:

```
FIELD SSN -
  PICTURE 'DDD-DD-DDDD' -
  HELP HELPSSN
```

The standard Help scripts from SIR/XS give extensive information on how to use SIR/XS and may be inappropriate for forms users. Help scripts are simply HTML text files and can be edited or altered as necessary for particular projects. The `index.htm` file in the `Forms` sub-directory in the help system is the starting point for SirForms help.

Error Messages

Error messages are displayed on the status line with a number in parentheses following the message. A full listing of the error messages is contained in Messages. Error messages can be altered to reflect the needs of specific projects by specifying an `ERROR` clause for the error message number at any level (form, screen, field). This replaces the default error message for everything within that level. For example:

```
FIELD SSN-
   PICTURE 'DDD-DD-DDDD' -
   HELP HELPSSN -
   ERROR 16 'Social security number is in the wrong format'
```

In this example, error message 16 "Input does not match picture", is replaced by the message "Social security number is in the wrong format" when a user makes an error.

This does not alter the html help page for message 16.

Using Forms

This covers how to use a form after it has been developed. It details how to enter data, how to locate records, how to move around the screens, and the commands that can be used.

Beginning a SirForms Session

You can run SirForms independently or from the SIR/XS menu.

See Running SirForms for details.

Passwords

Databases are protected from unauthorised access by passwords and security levels. A form may already have the passwords specified or may prompt for them. There are four possible pieces of information which may be required about the database. These are :

`Database Password`

This is the password for the database. If this is not specified correctly, you cannot access the database.

`Read Security`

This assigns a level of security for reading data. If particular records or variables have a higher level of security, you cannot read them.

`Write Security`

This assigns a level of security for writing data. If particular records or variables have a higher level of security, you cannot update them.

`Prefix`

If the database is not in your directory, SirForms needs to know where to find it. The prefix is the path name to locate the database.

The form itself may have restricted access to particular operations. This is done through a Group Name and Group Password. If this form has restrictions by group, supply a group name and password to perform those particular operations.

All of these parameters can be specified on the execution statement or can be entered in response to prompts. If a particular prompt does not apply, simply skip it.

Entering Commands

Every SirForms screen has two separate areas; one for data and another, at the bottom of the screen, for status messages and commands.

SirForms commands cyan be entered in one of three ways:

1. Function keys. Some common commands are performed with Function keys. Function keys such as `F1` are 'mapped' to specific commands; i.e. pressing the key performs the command. You can alter this mapping to suit your own style of working.

   The standard use of `Shift-F1` is to display "key help". This is a list of keys mapped to functions. While this is displayed, you can update the mapping by using the menus on the screen. The mapping for SirForms defaults to SirForms.kmp. If this file does not exist, when you start SirForms for the very first time, you are prompted to set up some essential key maps. Whenever a specific key is mentioned in this documentation it assumes that no re-mapping has been done.

2. Command Line. Commands can be entered at the command line but first you must position to the command line by either pressing the `Ctrl-Enter` (Command function key) or entering an = (equals sign) as the first and only character of a field. If the field is long enough then a command can be entered in the field by preceding it with an = (eg `=WRITE`).
3. Special Characters. Certain characters may be entered as the first and only character in a data field and these then act as a command. Enter the single character command (which clears the field) and press `Enter`. The command is executed leaving the field unchanged with the original value restored.

See Commands for a complete list of keys, commands and special characters.

Help

For help, either press `F1` or go to the command line and type `HELP`. SirForms uses the same Help as the rest of SIR/XS and brings up the specified HTML browser.

Ending a SirForms Session

To terminate the session immediately, go to the command line and type `EXIT`. Alternatively use `Shift-Esc(Stop)` or `Esc (Cancel)` repeatedly to return through any previous screens to the root screen and then to end the forms session.

Status Area

There are a number of status fields below the command line. There are two or three codes which indicate the mode this screen is in. Change the mode at the command line with the commands `FIND` or `SEARCH`, `VERIFY` or `NOVERIFY` and `EDIT` or `NOEDIT`. The default is `FIND, NOVERIFY, EDIT`, though these may be different in a particular form.

`F | S`

> This indicates whether the screen is in `Find` mode or `Search` mode. `Find` mode means you are finding a set of records directly by giving high level keys and is the default. You can create, update or delete records in `Find` mode. `Search` mode means you are searching for sets of records based on values. In `Search` mode you can use both key and non-key data to locate records, specifying the information to search by. When a record is retrieved, it can be updated or deleted. You cannot create new records in `Search` mode.

`V`

> If the second code is a "V", the screen is in `Verify` mode. `Verify` is used to validate existing data. It is used in conjunction with either `Find` or `Search` mode. In `Verify` mode, you re-enter data and SirForms checks it against the data already stored. This means you are re-keying existing data to check the accuracy of data entry (known as double key verification).

`E | N`

> `E` means the screen is in `Edit` mode and that in-line editing is enabled. This is the default. `N` means the screen is in `Noedit` mode and you must rekey the whole of any field to change data within it.

`Screen:` displays the name of the screen, and, if applicable, the page number and number of pages for the screen.

`Cir/Rec:` contains information about the case and record data being displayed. `Init` appears when the screen is in `Find` mode and is waiting for keys to be specified. `Search Init` appears when the screen is in `Search` mode and is waiting for a search specification. When a record or row has been retrieved, the two fields in parentheses show the type of lock and record status. These are primarily of interest when multiple people may be accessing records in concurrent mode. If you are using SirForms in concurrent mode, see Master.

`Status:` is where messages are displayed.

Entering Data

Each data field on the screen has a prompt or description of the information and a data area to enter or to display the data. When a screen is initially displayed, fields may already have data in them or the fields may be waiting for data to be entered. Each data area that is waiting for data is filled with a character that indicate the number of characters in the field (periods ..... by default).

To enter data into a field, type in the data. When you have finished, press `Enter`. To skip over fields, use the `Up Arrow` and `Down Arrow` keys. `Up Arrow` goes back a field, `Down Arrow` goes forward a field. Pressing `Enter` without typing in any data, also skips forward over a field.

Field Editing Operations

When you are positioned to a data entry field, you may edit the contents of the field. `Left Arrow` and `Right Arrow` position the cursor within the field and `Enter` terminates data entry and moves to the next field. For the specific keys to use for the other field editing functions listed below, use `Shift-F1` (key help). The following field editing functions are supported:

`Insert/Overstrike`

> If you begin typing immediately at the beginning of the field, the field is cleared. If you use the left and right arrow keys, you are editing the field in *Insert* mode. When you type a character, it is inserted at that point. Overstrike mode overwrites characters. Use `Ins` to toggle between Insert and Overstrike mode.

`Delete Character`

> Delete a character with the `(Del)` key or delete the previous character with the `(BkSp)` key.

`Un-delete Character`

> Restore the just deleted character with the `(UndeleteChar)` key.

`Delete Field`

> Clear the whole entry in the field with the `(DelField)` key. Cursor is left in the field.

`Un-delete Field`

Restore a cleared field with the `(UndelField)` key.

`Store Blank`

Store a blank in a field. Equivalent to entering a single space bar character in the field. Cursor is advanced to the next field.

`Store Undefined`

Stores a system undefined value in the field. Cursor is advanced to the next field.

Long Character Fields

Character (string) fields, may be longer than the display space on the screen. These fields scroll left and right as you type or as you use the left and right arrows.

No Data

If you move past a field with `Down Arrow` , or do not type anything and just press `Enter`, the data remains unaltered. If nothing is entered on a new record, the field is set to Undefined (the System Missing Value). If you enter blanks or edit a field and end up with blanks, this is different from Undefined and may be a valid missing value. `(DelField)` makes a field Undefined; `(BlankField)` stores blanks in a field

Field Help

For information about a field or about a field error message, use the `Alt-F1 (Default Field Help)` key. SirForms displays information about the field from the data dictionary. This includes the name of the field, its input format, its data type, the range of acceptable values, what its legal missing values are and what the codes mean. After reading the help information, return to the Record screen by pressing `Enter`. There is also the *Ctrl-F1 Custom Field Help* key. The information may appear in the status line at the bottom of the screen or it may appear on a separate screen. If there is no custom help screen or status line message, the same default help is displayed.

Flow of Control

The normal flow of control is from one field to the next field. This may happen automatically when a field has been completely entered (known as Autotab) or you may have to press `Enter` to move to the next field.

`Up Arrow` moves to the previous field, `Down Arrow` and `Enter` move to the next field. Fields may be organised in **Groups**. The *Next Group* key goes to the first field in the next group; the *Previous Group* key goes to the first field in the previous group; the *Top of Group* key goes to the first field in this group.

The *Top of Page* key goes to the first data field on this screen. The `Ctrl-R Restart` key clears the whole screen to enter new key field values.

Moving from Screen to Screen

A form usually consists of multiple screens. The data for one record may not fit on a single screen and may be split into **Pages**. If a screen is on multiple pages, the `Page Down - Next page` key moves to the next page; the `Page Up - Previous page` key moves to the previous page. When you move past the last field on a page, the next page is usually displayed automatically (this can be set by the screen designer). Similarly, if you move back past the first field on the second or subsequent page, the previous page is displayed. Screens for different records are linked together through menus or under program control.

A Menu is like a table of contents that tells you what other screens you can select. Page Down or a response of "Y" at the appropriate prompt, goes to that screen. This is known as "calling" a screen. Page Up or Esc returns to the calling screen. Another screen may be called from the lower level screen and screens may be nested as deeply as necessary.

Certain screens may be defined as **Stop** screens. The `Shift-Esc Stop` key returns to a higher level stop screen. This avoids going back through each level of called screen in a hierarchy of screens. (The very first screen is always a stop screen.)

Accessing Records and Rows

Key fields identify the record or row and are usually grouped at the top of the screen. The key fields are the first fields entered. When you enter a key for an existing record, the data for that is displayed. If you enter a key which does not exist, a new record is created.

Creating New Records

To create a new record or row, enter all the key fields. After the last key, a message that a new record has been created is displayed and the cursor moves to the first data field.

Enter the data fields. If the data for a field is invalid, an error message is displayed on the status line and the cursor returns to the beginning of that field for another entry.

The *Write* key or command writes the data to the database and clears all the fields. To write the data and leave the data on the screen, use `Up Arrow` to move to the key portion of the screen. This allows you to enter a new key value and use the existing fields as a starting place for the new record.

If you issue a command to move away from a screen where data has been modified, SirForms may ask whether to write this record. (When a form is created, it can specify that data is automatically written when you leave the screen.) Reply "OK" or "Y" to write the data.

Locating or Updating a Specific Record

To locate a specific record, enter data in all the key fields. The matching record is then retrieved. If no record in the database matches the key, a message that a new record has been accessed is displayed. The *Restart* key or command redisplays the screen without creating a new record.

If you omit a key field, the first record that matches whatever keys are entered is retrieved. If all key fields are blank, the first record in the database is retrieved.

There are function keys and commands to browse through sets of records. *First* retrieves the first record; *Last* retrieves the last record; *Next* and *Previous* go through the set of records one at a time in the specified direction. If no more records are available in the set, a message is displayed that the end of file has been reached.

Updating a Record

Once a record is displayed, you can modify the data fields. The `CANCEL` command restores the original data "undoing" whatever changes were made. When you have finished updating the record, `RESTART`, `WRITE`, or moving away from the screen, writes the changed data.

Deleting Records

Once a record is displayed, the `DELETE` command marks it for deletion. The record is deleted when you `RESTART` or `WRITE` the screen or you move away from the screen. To nullify the deletion, use `UNDELETE` before taking one of these actions.

If the screen has been specified to write records automatically, it is possible to create new records erroneously. If a record key (all the key fields) is entered which is not on the database, a new record is created. When you move away from the screen or restart, that record is written automatically. Use the `DELETE` command to flag the record for deletion in this instance.

Marking A Key Level

When going through records using `NEXT` and `PREVIOUS`, SirForms is controlling the set of records to access. If the top level key or partial key is entered, FORMS automatically restricts the set to records defined by that key. `MARK` allows you to go outside of this range. This is particularly useful when you do not know an exact key, but know roughly the value to start at. Mark a key field and enter a key value; this positions to that record or the first record with a higher key if that key does not exist. If key fields have not been Marked, a new record is created for keys which do not exist.

To mark a key field, position the cursor to the keyfield and use the `(Mark)` key. Forms responds with the message A "key level tagged". Then enter the start key value. To enter a different start key, mark the key again.

For Example: Use the Employee record and suppose 46 exists, but 45 does not. First mark the Identification Number and enter 45. Note that Employee 46 is displayed and that you can use `(Next)` and `(Prev)` to browse forwards or backwards from that point.

Now do not mark Identification Number and enter 45. Note that a new employee, 45, is created and if you use `(Next)` or `(Prev)` an error message 152 A "Command is not legal at current time" is issued.

Search Mode

Search mode locates records based on search specifications. These can consist both of key and non-key data. You cannot create new records in Search mode.

To set the search specifications, set Search Mode with the `SEARCH` command at the command line. This displays a screen with all fields blank. Then enter values in the fields to be searched. When you have finished entering the search specification, use `FIRST, NEXT, PREVIOUS` or `LAST` to retrieve matching records. While searching for a record to match the search specification, you may see a ".....searching" message displayed on the Status line.

If there are no more records that match the search pattern, a message is displayed. Terminate the search with `RESTART`.

The search specifications in individual fields are used to search for records with values which match all of the specified values.

The `WHERE` command defines search specifications that include logical conditions such as an `OR` condition, a `NOT` condition or tests for greater than or less than. For example, to search for employees who earn more than $3000;

WHERE (SALARY GT 3000) The `WHERE` command can be used in addition to individual field specifications. Only one `WHERE` clause is active at one time. Enclose the conditions on the `WHERE` in parentheses.

The hash sign (#) represents the field where the cursor was positioned when you went to the command line. For example, to search for female employees, position the cursor at the field GENDER and then go to the command line and enter

WHERE (# = 2)

Search Patterns

Values specified in fields are matched to be equal. To match on part of a field, use the percent sign (%) as a "wildcard" or A "match anything" symbol. For example, to retrieve the records for anyone with a last name ending in FORD, enter %FORD in the name field. String searches are case sensitive.

The wildcard symbol can appear more than once in the field. For example, %ABC%, finds names that contain ABC anywhere in the name.

Patterns can be used in a `WHERE ... LIKE` command. For example, to search for all the people whose names start with Fred:

```
WHERE (NAME LIKE '%Fred?')
```

Enclose the pattern including any symbols in single quotes. A pattern can consist of any character plus the following symbols:

`?` any character
`%` beginning of line
`$` end of line
`[...]` list of characters (i.e.,any of these characters)
`[!...]` negated list of characters (i.e., all but these characters)
`*` zero or more occurrences of the previous character or list
`+` one or more occurrences of the previous character or list

To specify a list of characters, specify either single characters or ranges of characters separated by a dash. The list must be enclosed in square brackets { [ ] }. Any character in the brackets is treated as the literal character not as the symbol.

Note that the pattern matching symbols used in where are not the same as the wildcards used in field.

The pattern matching features of the `WHERE ... LIKE` clause are extensive and are identical to those of SQL pattern matching.

Verify Mode

`Verify` mode is used to validate existing data. `Verify` mode is set with the `VERIFY` command at the command line. When in `Verify` mode, use the normal Find or Search mode techniques to retrieve records. When a record or row is located, the response area of each field is filled with the letter "V". To verify the row or record, enter values into the data fields. These are checked to see if the data that you have entered matches the data that was previously entered in that field. If it does, the cursor moves to the next data field. If the value does not match the data in the field, an error message is issued, and the previously entered data is displayed in the field. If the original entry is incorrect, modify it, press `Enter`, and continue processing.

To stop using `Verify` mode, use the `NOVERIFY` command at the command line.

Printing Screens

You can copy a screen to a file for later printing. The copy can contain the complete screen, just the data or just the screen without the data. The output file must be specified when you launch SIR FORMS; use the "`Print=filename`" parameter on the execution statement. To print a screen, use the `PRINT` command or key. `PRINT LABELS` leaves off the data; `PRINT DATA` leaves off the labels.

This is a text representation of the screen not a graphic. If you wish to save a "picture" of a screen, use your normal screen save process. (eg Alt PrtSc in Windows).

Keys and Commands

The following is a list of SirForms actions with function keys, symbols (one special character typed in a field) and commands. Some actions can only be performed by entering the command (e.g. SEARCH), and some can only be done with keys (e.g. delete a character).

To use the command symbols, enter the single character command (which clears the field) and press `Enter`. The command is executed leaving the field unchanged with the original value restored.

Key	Sym.	Command	Action
Shift-F1			Get help on keys
Esc	<	RETURN	Go back a screen
Shift-Esc	.	STOP	Go back to stop screen
Left Arrow			Moves left in a field for editing. Returns to a previous screen at a CALL prompt
Right Arrow			Moves right in a field for editing. Moves to chosen screen at a CALL prompt
Down Arrow			Moves past a field
Up Arrow			Moves back a field
Del			Deletes the character at the cursor position.
Bksp			Deletes the character before the cursor position
shift-Del			Delete the field
ctrl-Z			Restore the previously deleted field
ctrl-U			Make a field Undefined
ctrl-B			Make a field blank
ctrl-F1	?		Get help on a field
Alt-F1	!		Get dictionary help on a field
Pg Dn	>		Call Screen or Page Down
Shift-Enter	=		Go to Command Line
Ctrl-M	@		Mark a key level
Ctrl-Left Arrow	(	FIRST	Retrieve first record or row
Ctrl-Right Arrow	)	LAST	Retrieve last record or row
Shift-Right	]	NEXT [n]	Retrieve next record or row. [Skip forward n.]

Arrow			
Shift-Left Arrow	[	PREVIOUS [n]	Retrieve previous record or row. [Skip back n.]
Ctrl-R	*	RESTART	Clears the current screen to specify new keys to retrieve.
	^	TOP	Moves cursor to first field on page
		CANCEL	Restores original data.
		DELETE	Marks record or row for deletion.
		DFIRST	Deletes current and retrieves first.
		DLAST	Deletes current and retrieves last.
		DNEXT n	Deletes current and retrieves next.
		DPREVIOUS n	Deletes current and retrieves previous.
		DRESTART	Deletes current and restarts screen.
		DRETURN	Deletes current and returns to calling screen.
		EXIT	Finishes SirForms session.
		FIND	Sets Find on.
		NOVERIFY	Turns Verify off.
		PAGE n	Moves to a page.
		PRINT	Prints screen. DATA prints data only. LABEL prints labels only.
		SEARCH	Sets search mode.
		STATUS	Displays status.
		UNDELETE	Removes delete flag from current record or row if it is marked for deletion.
		UPDATE	Writes record and positions to first field on record
		WHERE	Sets a search condition.
	#	WRITE	Writes record and clears screen.

General Clauses

Various clauses and specifications are common to several commands. The command specifications include a set of clauses labeled `General Clauses`. This documents these general clauses. The video options apply to a number of clauses.

The general clauses are:

ACTIVITIES
AT
AUTOTAB
BOTTOM
CLEAR
DATA
ERROR
LABELS
PAD
PAGESIZE
PROMPT

ACTIVITIES

`ACTIVITIES permissions`

Specifies permissions for the commands and actions that are allowed. If the `ACTIVITIES` clause is omitted, all activities except `AUTOCASEDELETE` are allowed. Activities can be specified on the `FORM, RECORD` or `TABLE` commands and the `USER` clause. Field level activities (`MODIFY` or `NOMODIFY`), can be specified at the Page, Group and Field levels.

Permissions on the `FORM` level are the defaults for all screens in the form. Permissions allowed at the `FORM` level can be disallowed for individual screens; disallowed permissions cannot be reinstated at lower levels. `ACTIVITIES` restricted for a `USER GROUP` at a given level cannot be allowed at a lower level.

To disallow any permission, prefix the permission keyword with "NO", for example, `NOREAD`. The following permissions can be specified:

`READ`
    read cases and records
`WRITE`
    write cases and records
`MODIFY`
    modify fields in existing records
`FIND`
    use FIND mode
`SEARCH`
    use SEARCH mode
`DELETE`
    delete records
`NEWCASES`
    create new cases
`NEWRECORDS`
    create new records
`NEWROWS`
    create new rows
`OLDCASES`
    access existing cases
`LOCKCHANGE`
    change LOCK status of screen
`OLDRECORDS`
    access existing records
`AUTOUPDATE`
    automatically update records without prompt
`AUTOCASEDELETE`
    automatically delete cases when all records for that case are deleted
`VERIFY`
    use VERIFY mode
`CHANGE`

	change data in VERIFY mode
`ALL`	all activities are allowed
`NONE`	no activities are allowed

AT

`AT [ row ] [, column ]`

`AT` specifies the starting position for the display element with which it is used. When used as a clause on the `GROUP` command, it specifies the starting position of the first display element within the group.

`Row` and `column` specify screen coordinate numbers. The top row on the screen is row 1 (one). The leftmost position on the screen is column 1 (one). The maximum values for row is 30 and for column is 80. Forms uses the bottom two or three rows for the status and command areas. The `BOTTOM` clause is used to set the number of rows used for the bottom area. `AT` may not position elements in the bottom area.

Row and column positions can be specified in either absolute or relative terms. Relative positions are relative to the current default position. The default position is one row greater and in the same column as the starting position of the previous displayed element. The initial position for the first screen element is at (1,1). Relative positioning is specified by prefixing the row or column with a plus or minus sign. `AT -10,+40` specifies a placement ten rows higher and forty columns to the right of the current default position.

The stand-alone `AT` command resets the current default location.

An asterisk (*) may be used instead of row or column numbers to indicate the maximum row or column. For example:

`AT 1 , 1`

positions to the upper left corner.

`AT * , 1`

positions at the last row, column 1.

`AT , 12`

positions at column 12 of the current line.

`AT -1 , 1`

positions at the beginning of the previous line.

`AT , +10`

positions 10 columns from the beginning of the previous field.

AUTOTAB

`AUTOTAB` specifies that, when the cursor moves past the last character of a field, it automatically goes to the next field. With `NOAUTOTAB` (the default), the user must explicitly move away from the field. Similarly, with `AUTOTAB` specified, if the user moves back past the first character of a field, the cursor goes to the previous field.

BOTTOM

`BOTTOM {2 | 3}`
Specifies that either 2 or 3 lines are used for the status area at the bottom of the screen. 3 is the default.

CLEAR

```
  CLEAR 'character'
```

Specifies the character used to delineate a data field which is not yet filled in. A period (.) is the default.

Example: To delineate data fields with an underscore, specify:

```
CLEAR '_'
```

DATA

`DATA { HILITE | VIDEO } video_options`
Specifies the video treatment for the data area of fields.

`HILITE`

specifies the video options for the data portion of a field when the cursor is positioned at that field.

`VIDEO`

specifies video options for the data portion of the field at all other times.

ERROR

`ERROR message_number [VIDEO video_options] 'error_text' [message_number [VIDEO video_options] 'error_text' ... ]`

Specifies alternate text for messages. Each message consists of the message_number followed by text. See messages for a full list of default messages.

Error messages apply to the level (`FORM, SCREEN, PAGE, GROUP, FIELD`) at which they are defined and to all inner levels unless overridden by another error message defined at the inner level. Video_options can be specified for error messages. The default `VIDEO` option is `BELL`. The `NOBELL` option turns off the 'bell' or 'beep' when an error message is displayed.

Example: To change the messages for errors 15 and 47 to
"This field must be two digits." and "Jobcode must be between 10 and 60.", and have the messages displayed in inverse and have a bell sound, specify:

```
FIELD JOBCODE -

  ERROR 15 VIDEO BELL INVERSE -

           'This field must be two digits.'-

        47 VIDEO BELL INVERSE -

           'Jobcode must be between 10 and 60.'
```

LABELS

`LABELS n`

Specifies that value labels, "n" characters long, are displayed next to all fields for which value labels are defined. If the LABELS clause is omitted, value labels are not displayed.

Example: To display 20 characters of value labels next to all data fields, specify:

```
FORM TESTFORM -
       LABELS 20
```

PAD

`PAD [ n ] [, m ]`

Specifies the display width of prompt (n) and data (m). By default, the display width of prompts are half the screen width and the display width of data is the width of the field plus 1.

Explicit prompts (specified on a `PROMPT` clause) are not truncated, even if the width of the explicit prompt exceeds the value specified for n. Similarly, data areas are not truncated, even if the width of the data item exceeds m. (Use the `WIDTH` clause on the `FIELD` command to alter the data width of a field.)

Prompts and character data are left justified and padded with blanks. Numeric data is right justified with leading zeros removed.

Example: Value labels are displayed one column to the right of a field's data area. If there are several such fields of varying width, and you want to align the value labels, specify a width for the data items which is the maximum for any data item. To reserve 15 columns for all data areas and align the value labels, specify:

```
PAD  , 15
```

PAGESIZE

`PAGESIZE [ r ] [, c ]`

Specifies the size of the display area (`r` = number of rows, `c` = number of columns). If you specify sizes smaller than the execution window (30,80), the display takes up the top left portion of the window. If you specify a display area larger than the execution window, you will not be able to see your display and the screen formatting may produce undesirable results.

Do not use this clause with the execution window under Windows or Motif.

The specified row size does not include the bottom display area as defined by the `BOTTOM` clause.

The default useable size is 27 by 80 (window size less the status area).

PROMPT

`PROMPT { HILITE | VIDEO } video_options`

Specifies the video treatment of the prompt area of a field. (See video treatment for a description of video_options.)

`HILITE`

> specifies the video options for the prompt portion of a field when the cursor is positioned at that field.

`VIDEO`

> specifies video options for the prompt portion of the field at all other times.

video_options

Video options are specified by the keywords AT0 through AT16 or by synonyms.

AT1 through AT6 are essentially 'monochrome' options which use a default foreground of black with a default background of white.

AT8 through AT16 are colour options which control both foreground and background and, if any of these are specified, they turn the monochrome defaults off so that, if a foreground colour is not specified, it is black; if a background colour is not specified, it is white.

The colour foreground is dim by default. The colour background is bold by default. Multiple options can be specified together. When using colours, specify blue, green red and bold to get white.

AT0 No effect
AT1 BOLD - Blue on default background
AT2 INVERSE - Swaps foreground and background (only if colours not specified)
AT3 No effect
AT4 UNDERLINE - Underlines with a black line (works with colours)
AT5 DIM - Dim foreground Grey on default background
AT6 GREY - Dim background default foreground on Grey
AT7 No effect
AT8 FGBLUE - Foreground Blue
AT9 FGGREEN- Foreground Green
AT10 FGRED - Foreground Ref
AT11 FGBOLD - Foreground Colour Bold
AT12 BGBLUE - Background Blue
AT13 BGGREEN - Background Green
AT14 BGRED - Background Red
AT15 No effect
AT16 BGDIM - Background colour dim

Examples: To highlight the data field where the cursor is positioned, in bold inverse, specify either of the following:

```
 DATA HILITE INVERSE BOLD
DATA HILITE AT2 AT1
```

To display all data fields in inverse, specify either of the following:

```
DATA VIDEO INVERSE
DATA VIDEO AT2
```

To display all fields in inverse and to make them bold when being prompted, specify either of the following:

```
DATA HILITE BOLD VIDEO INVERSE
DATA HILITE AT1 VIDEO AT2
```

Compiler Directives

There are a number of commands which deal with input such as putting comments in the input text, including text from another file, creating synonyms for any text string and controlling the way the output from the compiler is listed. The compiler directive commands can be used at any point in the input file.

Comments

Comments are indicated by an asterisk (*) as the first non-blank character on a line. For example:

```
* This is a Comment Line
```

INCLUDE

```
INCLUDE 'file_name'
```

Includes a file of commands at that point in a Form Definition. The Included file can contain any valid set of commands. `INCLUDE` is useful when developing screens separately or when a set of commands is common to several Form Definition Files.

- For example, to include a set of standard synonyms:

```
INCLUDE 'SYNONYM.cmp'
```

SYNONYM

```
SYNONYM name text
```

Defines a name that is replaced during compilation with the text. The `SYNONYM` command creates a short mnemonic name for a long, complex expression. The synonym definition must occur before it is referenced in another command. Once a synonym has been defined, it can be used anywhere in the form definition.

For example: suppose the `CALL` for a number of screens is to have the following clauses:

```
CALL screen AUTO CALL RETURN WRITE NOPROMPT NODATA
```

A synonym called `AUTOSCREEN` could be created and used as follows:

```
SYNONYM AUTOSCREEN -
  AUTO CALL RETURN WRITE NOPROMPT NODATA
...
CALL screen AUTOSCREEN
```

Listing Commands

These commands control the titles and paging of a listing of the compilation. In order to get a listing of the compilation, use the `LISTING = filename` parameter on the execution statement. These commands do not affect anything other than the listing.

When compiling large forms, you are advised to send the output to a listing file, and review this for syntax or compilation errors.

EJECT

Causes a page eject to occur at this point.

FOOTING

```
FOOTING 'footing_string'
```

Causes a text string to be printed at the bottom of every output listing page. `FOOTING` can be respecified as many times as necessary.

MAINTITLE

```
MAINTITLE 'main_title_string'
```

Causes a text string to be printed as the first line of each output page.

SPACE

```
SPACE [n][,m]
```

Prints n blank lines. If fewer than n lines remain on the page, a page eject is performed. If m is specified, and at least n+m lines remain on the page, n lines are skipped and printing continues. A page eject is performed if fewer than n+m lines remain. This allows a group of commands to be printed together.

- **Example:** To print a blank line before printing the next 10 lines making sure all 10 lines fit on a page:

```
SPACE 1,10
```

SUBTITLE

```
SUBTITLE 'subtitle_string'
```

Causes a string of text to be printed as the second line of each output page. The `SUBTITLE` command can appear as many times as necessary.

FORM

The `FORM` command specifies the name of the form, the database(s) and tabfile(s) that the form is to access, plus security, options and defaults that apply to the entire form.

Any temporary variables are specified on the `FORM` command.

The only commands that can precede the `FORM` command are compiler directive commands.

The last command of a Form Definition must be an `ENDFORM` command.

Syntax

`FORM name`

The `FORM` command is required and the name of the form must be supplied. All other clauses are optional. The following clauses can be specified on the `FORM` command as required:

AUTO WRITE
DATABASE
TABFILE
DECLARE
MESSAGE
ROOT
USERS

The general clauses can be specified on the `FORM` command as required.

Example:

```
FORM EXAMPLE -
     DATABASE COMPANY -
              PASSWORD COMPANY -
              SECURITY * , * -
              PREFIX 'C:\MYFILES\' -
     LABELS 20 -
     DECLARE  INTEGER TOTIN -
              STRING XNAME-
     ROOT MAINMENU
```

This example specifies that the database is COMPANY, password COMPANY in a directory C:\MYFILES\.

The `SECURITY` clause is specified as asterisks which means that users are prompted to enter read and write passwords when they run the form. If the passwords are coded on the `FORM` command, anyone can use the form without knowing the passwords.

The `LABELS 20` clause specifies that the first 20 characters of any value labels defined in the database schema are displayed together with the field.

The `DECLARE` clause defines all temporary variables used in the form. Temporary variables can be referenced on any screen.

The `ROOT` clause declares the name of the first screen to display when execution begins, in this case MAINMENU.

AUTO WRITE

`[NO]AUTO WRITE`
Specifies that any new records are written to the database without asking for confirmation.

`NOAUTO WRITE` specifies that the user is asked to confirm that the record should be written. `NOAUTO WRITE` is the default.

DATABASE

```
DATABASE name
         PASSWORD password [SECURE]
         SECURITY rpassword, wpassword [SECURE]
         PREFIX  'prefix' [SECURE]
         READ | WRITE
```

Multiple SIR/XS databases can be accessed by a form. A separate database clause is required for each database to be accessed. The database name must be specified. The other information may be specified as part of this clause, on the execution statement, or be prompted for interactively at run time.

`PASSWORD`

> Specifies the database password. If this clause is omitted, the password is taken as blank. An "*" specifies that the password is prompted for when the form is compiled.

`SECURITY`

> Specifies the database read and write security passwords. Specify the read password first, then the write password separated by a comma. If this clause is omitted, the passwords are taken as blank. An "*" specifies that the password is prompted for when the form is compiled.

`PREFIX`

> Specifies the directory path where the database is located if it is not in the current database directory. Omitting the clause or specifying an "*", specifies that the database is in the database directory. This is the current directory or the directory set with the `PREFIX` execution parameter.

`SECURE`

> `SECURE` specifies that the `PASSWORD, SECURITY,` or `PREFIX` supplied are used only for compilation. When the compiled form is executed, the database directory is used. This is the current directory or the directory set with the `PREFIX` execution parameter.

`READ | WRITE`

READ specifies that the database is opened for read-only access so others can share access to the database without using MASTER. WRITE specifies that the database is opened for read and write operations and is the default.

The following example specifies access to the COMPANY database with secure passwords:

```
DATABASE  COMPANY -
          PASSWORD COMPANY SECURE -
          SECURITY HIGH, HIGH SECURE
```

TABFILE

```
TABFILE name
        FILENAME 'filename'
        IDENT group/password.user/password
        READ | WRITE
```

Multiple tabfiles can be accessed by a form. A separate tabfile clause is required for each tabfile to be accessed. The tabfile name is the only required clause.

`FILENAME`

> Specifies the operating system file name of the tabfile. The default filename is the name of the tabfile.

`IDENT`

> Specifies a user group, user and passwords. See tabfiles for details on the security aspects of user groups and users on tabfiles.

`READ`

> Specifies that the tabfile is opened for read-only access. This allows shared access to the tabfile.

`WRITE`

> Specifies that the tabfile is opened for read and write operations. This is the default.

DECLARE

```
DECLARE STRING  variable,  ....
        REAL    variable,  ....
        INTEGER variable,  ....
```

`DECLARE` specifies all the temporary variables used in the form. Temporary variables are global to the entire form and are available for use by any screen in the form. Declare numeric variables as `REAL` or `INTEGER` and character variables as `STRING`. If the temporary variables are displayed on a screen, use the clauses on the `FIELD` command to specify how the temporary variables are formatted.

Example: To declare the numeric variables `CHEM` and `PRESSURE` and the string variable `COMPOUND`, specify:

```
DECLARE REAL CHEM PRESSURE -
        STRING COMPOUND
```

See Variable Names.

MESSAGE

```
MESSAGE ERROR VIDEO video_options
WARNING VIDEO video_options
```

The `MESSAGE` clause establishes the default video options for all error and warning messages. (See video treatment for the video_options.)

`ERROR`

Specifies the video treatment of error messages.

`WARNING`

Specifies the video treatment of warning messages.

- **Example:** To specify that error messages are in bold:

```
MESSAGE ERROR VIDEO BOLD
```

ROOT

`ROOT screenname [CALL options]`

Specifies the name of the screen which is used when execution begins. If `ROOT` is not specified, the first screen defined in the form is used.

`ROOT` is a special CALL command and all the `CALL` options can be specified on the `ROOT` clause. Since defaults are passed down from one called screen to another, any options set on the `ROOT` call are in effect for the entire form.

USERS

```
USERS GROUP  name  [/password]
             ACTIVITIES permissions
             INCLUDE name [/password] ...
```

Specifies the names of the groups allowed to use the form, their passwords and the types of activities they are allowed. If `USERS GROUP` is specified, everyone using the form must supply a matching group name. SirForms prompts for the relevant group names and passwords before permitting access to the databases and tabfiles. Group names and user names can be supplied on parameters on the execution statement.

If multiple groups access the same form, the `USERS GROUP` clause controls permissions for specific operations. All activities are controlled at the `GROUP` level. To identify a number of different users within a group, use the `INCLUDE` clause. This means that more information has to be supplied in order to be recognised as a member of this group. If you specify a group name, password, and a user name and password, the user has to supply four names before being allowed to access the form.

The keyword `USERS` is only specified once, regardless of the number of `GROUP` names specified.

`GROUP`

> Specifies the name of the group, followed by an optional password.

`ACTIVITIES`

> Specifies the permissions for the user group. Permissions disallowed on the `FORM` command cannot be reinstated at a lower level.

`INCLUDE`

> A user group can consist of more than one person. Individual names, with optional individual passwords, can be specified on the `INCLUDE` clause. However, the user must still supply the group name and permissions are set at the group level.

Example: To define two groups, `LOOK` with members `Anne` and `Alan` and `DO` with members `Betty` and `Bob`, where `LOOK` is restricted to read access, and `DO` can do anything, specify:

```
USERS GROUP LOOK ACTIVITIES READ NOWRITE -
            INCLUDE ANNE, ALAN  -
      GROUP DO   ACTIVITIES ALL -
            INCLUDE BETTY BOB
```

Screens

A screen definition starts with one of the RECORD, TABLE or MENU commands. Within a screen there are then commands to compute data, to display fields, to control the appearance of the screen and to link to other screens.

A screen in this sense is not necessarily a display which fits on one window. A screen can be broken up into pages which do correspond to a single window. If more lines are specified on a screen than fit on the window, then the screen will be displayed in pages or pages can be defined explicitly. Screens do not scroll.

The sequence of the commands within the screen definition determines the order in which the cursor moves and the sequence in which the commands are executed (the "flow of control"). If screen positioning clauses are used, this may not necessarily be the "natural" order of fields as they appear on the screen.

The command names the screen and screen names must be unique within a form.

The set of commands for a screen must end with the appropriate `END` command; `ENDMENU, ENDRECORD` or `ENDTABLE`.

The main difference between the screens defined by `RECORD`, `TABLE` and `MENU` commands is that record screens access data from a single record in a database whereas table screens access data from a single row in a table. Menu screens do not access any external data. All screen types have access to any temporary variables defined in the form.

The syntax for the commands is the same except that `AUTOWRITE` only applies to `RECORD` and `TABLE` screens.

MENU

```
MENU screenname
```

A menu screen starts with the `MENU` command followed by the screen name and finishes with `ENDMENU`. Within a menu screen, other commands can be used, however `FIELD` commands can only refer to temporary variables not to database or table variables.

Menu screens may consist simply of choices as to which screens to go to or a menu can be a complete data entry screen providing that it only refers to temporary variables. These can then be passed to record or table screen to write the data.

See syntax.

RECORD

```
RECORD screenname [/database.record]
```

A record screen starts with the `RECORD` command and ends with `ENDRECORD`.

Specify the command `RECORD` followed by the name of the screen and, optionally, the name of the record and the name of the database. If a record name is not specified, the screen name must be the record name. If a database name is not specified, the first database defined on the `FORM` command is the default.

The same record type can be associated with multiple screen definitions in the same form definition to allow different ways of viewing the same data. Each screen name must be unique.

Record screens access and display one record at a time. Once control has been passed to a record screen, a record can be retrieved or a new entry created. The user can enter the key fields to locate the record, or can find the appropriate record or row with commands such as `FIRST, LAST, NEXT` or `PREVIOUS`. If the record or set of records to be retrieved has been determined in a previous screen, this information can be passed down with the `CALL` command.

Within a screen, commands can reference any variable from that record plus any temporary variables. Commands can also reference variables from the CIR on record screens in a case structured database.

See syntax.

TABLE

```
TABLE screenname [/tabfile.table] [ INDEXED BY indexname ]
```

A table screen starts with the `TABLE` command and ends with `ENDTABLE`.

Specify the command `TABLE` followed by the name of the screen and, optionally, the name of the table and the name of the tabfile. If a table is not specified on the `TABLE` command, the screen name must be the table name. If a tabfile is not specified, the default tabfile is the first tabfile defined on the `FORM` command.

An index can be specified for a table and the index variables are treated as the key fields. The index variables must appear first on the screen. Only one index can be specified for a table screen. If an index is not specified, the default access mode is `Search` and the table is processed sequentially.

The same table can be associated with multiple screen definitions in the same form definition to allow different ways of viewing the same data. Each screen name must be unique.

Table screens access and display one row at a time. Once control has been passed to a table screen, a row can be retrieved or a new entry created. The user can enter the key fields to locate the row, or can find the appropriate row with commands such as `FIRST`, `LAST`, `NEXT` or `PREVIOUS`. If the row or set of rows to be retrieved has been determined in a previous screen, this information can be passed down with the `CALL` command.

Within a screen, commands can reference any variable from that table plus any temporary variables.

See syntax.

Syntax

The clauses for `RECORD, TABLE` and `MENU` commands are:

Clauses:

ACCEPT `(condition) 'error message',[OPTIONAL] ...`
REJECT `(condition) 'error message',[OPTIONAL] ...`
[NO]AUTO WRITE
FINAL COMPUTE `variable=expression, ...`
INITIAL COMPUTE `variable=expression, ...`
KEEP
PRELOAD
RECEIVING `variable, ...`
STOPSCREEN
USERS GROUP `name ACTIVITIES permissions`

General Clauses

```
ACTIVITIES permissions
AT [ r ] [, c ]
[NO]AUTOTAB
BOTTOM n
CLEAR 'c'
DATA HILITE | VIDEO options
ERROR number [ VIDEO options ] 'error text' , ....
PAD [n][,m]
PAGESIZE [r][,c]
PROMPT HILITE | VIDEO options
```

ACCEPT

`ACCEPT (condition) 'error message' , ... [ OPTIONAL ]`

`ACCEPT` specifies one or more checks to be performed before the user "moves away" from a screen. Moving away from a screen means retrieving a new record on this screen, calling another screen or returning to the screen that called this one.

If an `ACCEPT` expression is not true, the specified message is displayed and the cursor returns to the first input field. If a message is not specified, a default message is used. If both `ACCEPT` and `REJECT` are specified, the `ACCEPT` is actioned first and records must meet all specified criteria in order to be accepted.

Only one `ACCEPT` clause can be specified on a command.

Specify the keyword `OPTIONAL` after all conditions to allow the user to override the warning. `OPTIONAL` applies to all the specified conditions.

Specify the `BOOLEAN` clause on the `FIELD` command to test individual fields. This is the preferred practice for testing individual fields since the cursor is then positioned on the field in error.

Example: To leave the screen only when `DATE2` is equal to or greater than `DATE1` specify:

```
ACCEPT (DATE2 GE DATE1) 'Invalid dates .. Please re-enter'
```

AUTO WRITE

`[NO]AUTO WRITE`

`AUTO WRITE` specifies that a new record is written when the user moves away from the screen. If `NOAUTO WRITE` is specified, a prompt is issued for permission to write the record.

If the user issues an explicit `WRITE` command, the setting of `AUTO WRITE` has no effect.

Moving away from a screen means retrieving a new record on this screen, calling another screen or returning to the screen that called this one.

FINAL

```
FINAL COMPUTE   variable = expression, ...
      IF (expression)  variable = expression, ...
```

Specifies computations which are done when the user moves away from a screen. Moving away from a screen means retrieving a new record on this screen, calling another screen or returning to the screen that called this one.

Regardless of the number of variables to be computed, only specify the keyword `FINAL` once. The computations are performed before any `ACCEPT` or `REJECT`.

Variables can be computed with `COMPUTE` or conditionally computed using `IF`. Multiple `COMPUTE` and `IF` statements can be specified. Use a comma to separate expressions. An expression can be enclosed in parentheses.

Example: To compute the sum of `VAR1, VAR2` and `VAR3` on exit from the screen, specify:

```
FINAL COMPUTE TOTAL = VAR1 + VAR2 + VAR3
```

INITIAL

```
INITIAL COMPUTE  variable = expression, ...
        IF (expression)  variable = expression, ...
```

Specifies computations which are done when the user first moves to the screen. Regardless of the number of variables to be computed, only specify the keyword `INITIAL` once. The `INITIAL` computations are performed before the first command in the screen and before a record or row is accessed.

Variables can be computed using `COMPUTE` or conditionally computed using `IF`. Multiple `COMPUTE` and `IF` statements can be specified. Use a comma to separate expressions. An expression can be enclosed in parentheses.

Example: To set the global variable `TIME` to the current time when a screen is accessed, specify:

```
INITIAL COMPUTE TIME=NOW(0)
```

KEEP

Specifies that once this screen is accessed, it remains in the screen cache. See parameters for information on the `NUMSCR` parameter on the execution statement.

PRELOAD

Specifies that the screen is put in the screen cache as soon as the form is loaded.

RECEIVING

`RECEIVING variable, ...`
Specifies a list of variables in the screen that are to receive values from the calling screen. These values are specified on the `PASSING` clause on the `CALL` statement in the calling screen. The variables in the Passing/Receiving list are matched by position. That is, the first variable in the passing list corresponds to the first variable in the receiving list, the second variable in the passing list corresponds to the second variable in the receiving list, etc.

Values that are "received" are `DISPLAY` fields and cannot be modified.

When creating a new record, `RECEIVING` forces a value into the variable. If an existing record is retrieved, all the record variables are set to the values from the database. This means that received data is overwritten and the only effect is to make the variable a `DISPLAY` field.

A screen may be called from more than one place. If a `CALL` does not pass any data, `RECEIVING` has no effect.

REJECT

`REJECT (condition) 'error message', ... [ OPTIONAL]`
Specifies one or more checks performed before the user "moves away" from a screen. Moving away from a screen means retrieving a new record on this screen, calling another screen or returning to the screen that called this one.

If a `REJECT` expression is true, the specified message is displayed and the cursor returns to the first input field. If a message is not specified, a default message is used. If both `ACCEPT` and `REJECT` are specified, the `ACCEPT` is actioned first and records must meet all specified criteria in order to be accepted.

Only one `REJECT` clause can be specified on a command.

Specify the keyword `OPTIONAL` after all conditions to allow the user to override the warning. `OPTIONAL` applies to all the specified conditions.

The `BOOLEAN` clause on the `FIELD` command can be used to test individual fields. This is the preferred practice for testing individual fields since the cursor is then positioned on the field in error.

Example: To stay on this screen and not move away when `DATE2` is equal to or greater than `DATE1` specify:

```
REJECT (DATE2 GE DATE1) 'Invalid dates .. Please re-enter'
```

STOPSCREEN

`STOPSCREEN`

Specifies that this is a "Stop" screen. The `STOP` command passes control back through any number of levels of calls to a stop screen. Multiple stop screens can be specified. A user can skip back over a series of screens instead of using the `RETURN` command in each screen. The root screen is a stop screen by default.

USERS

`USERS GROUP name ACTIVITIES permissions`
Specifies whether particular user groups can access this screen and what activities are allowed.

Regardless of how many groups are defined, only specify the keyword `USERS` once. The keyword `GROUP` is followed by the name of the group allowed to access the screen and the permissions allowed. To restrict activities on this screen for everyone regardless of the group, omit the keyword `GROUP` and the groupname. This specifies that all groups are allowed access to the screen to perform the given permissions. If `USERS` are specified for a screen, specify all groups that are permitted access to the screen; any groups omitted from the `USERS` clause are allowed no activities. The `ACTIVITIES` clause specifies the permissions for the group. (See activities for permissions).

If a `USERS` clause is specified on a screen, then for a group to be allowed to perform an activity:

- The `USERS GROUP` must be specified at the `FORM` level and the activity must be allowed. This allows access to the Form and to that activity in general.
- The group must be specified on the `USERS GROUP` on this screen.
- If `ACTIVITIES` are specified for the screen, the activity must be specified.

Example: Allow only group `LOOK` to access the Menu screen `SUBMENU`.

```
MENU SUBMENU USERS GROUP LOOK
```

Page and Group

If record, table or menu screen contains more than one display screen, then each screen is then referred to as a page. Specify pages with the PAGE command and end a page with an `ENDPAGE` command.

If fields do not fit on a screen, they are automatically split into pages. There is no difference between explicitly specified pages and automatically generated pages. Attributes and security can be specified at the page level. When a screen has multiple pages, this is indicated in the `Status` area during use of the Form. The user can tell which page is being displayed, as well as the total number of pages in the screen. Specify groups of fields with the GROUP command and end a group with an `ENDGROUP` command. A group of fields can be treated as a unit for positioning, logical control, security and for video attributes. The way users move around the screen can affect computations and the execution of other commands in the form. Commands, including computations, are performed as the flow of control "passes through" the command. If the user skips from one group to the next, the flow of control does not "pass through" the fields and other commands in the group and thus any computations are skipped.

PAGE

There are no required clauses on the `PAGE` command. The clauses that can be specified are:

Clauses:

```
[NO]AUTO LOOP | PAGE
IF (expression)
```

General Clauses:

```
ACTIVITIES permissions
AT [r][,c]
[NO]AUTOTAB
BOTTOM n
CLEAR 'c'
DATA HILITE | VIDEO options
ERROR number [ VIDEO options ] 'error text'
LABELS n
PAD [n][,m]
PAGESIZE [r][,c]
PROMPT HILITE | VIDEO options
```

GROUP

There are no required clauses on the `GROUP` command. The clauses that can be specified are:

Clauses:

IF `(expression)`

General Clauses:

```
ACTIVITIES permissions
AT [r][,c]
[NO]AUTOTAB
CLEAR 'c'
DATA HILITE | VIDEO options
ERROR number [ VIDEO options ] 'error text'
LABELS n
PAD [n][,m]
PROMPT HILITE | VIDEO options
```

AUTO

```
AUTO {LOOP | PAGE }
```

Controls the flow of control at the top or bottom of a page. `AUTO LOOP` specifies that control stays on the current page and loops when the user moves up from the top most field or down from the bottom most field on the page. The user must specifically move to the previous or next page of a screen when `AUTO LOOP` is specified for a page. `AUTO PAGE` is the default and causes the flow of control automatically to move to the previous or next page when the user moves up from the top most field or down from the bottom field on a page.

IF

```
IF (expression)
```

`IF` specifies a logical expression that controls whether the `PAGE` or `GROUP` command is executed. If the expression is true, the command is executed. If it is false, the whole `PAGE` or `GROUP` is skipped.

```
FIELD PREGNANT IF (GENDER =2)
GROUP IF (GENDER = 2 and PREGNANT = 1)
  FIELD ....
  FIELD ....
ENDGROUP
```

This allows input of data into a variable `PREGNANT` only if `GENDER` equals 2. The following group of fields is skipped completely if `GENDER` is not 2 or `PREGNANT` is not 1. The fields within a group are always displayed whether or not they are executed.

FIELD

The `FIELD` command displays the current value of variables on the screen and provides extensive capabilities for editing and validating data values.

A `FIELD` command without any clauses displays the data at a default position on the screen using the dictionary definitions to control the prompt, the data format and the edit rules.

When defining custom screens and additional edits, much of the work done by the screen designer is in specifying the various clauses to do with this command. Clauses specified at the Form, Screen, Page and Group levels to apply to all fields within that level.

It is used in record screens for record variables, in table screens for table columns and in any screen for temporary variables. The same variable can be referenced by multiple `FIELD` commands on a screen.

The command is executed when the flow of control moves to it. The sequence of the commands determines the flow of control and this is the sequence followed by the cursor on the screen. The key fields for a record, or the index fields for a table, must be the first fields referenced on a screen. Field commands do not have to correspond to the sequence of variables in a row or record. All of the fields on a record or table do not have to be on a screen.

Syntax

Clauses:

The only required clause on the `FIELD` command is the name of the variable.

```
FIELD variable
BOOLEAN ( expression ) [OPTIONAL]
[NO]DATA [AT r , c] [HILITE | VIDEO video_options]
DEFAULT expression
DISPLAY ['prompt' | VARDESC | VARLABEL | VARNAME] [VIDEO video_options]
DSPEDIT options
DSPTYPE options
FINAL COMPUTE variable=expression, ...
HELP helpscreen | 'help_string'
IF ( expression )
INITIAL COMPUTE variable=expression, ...
INSERT | OVERSTRIKE
LEFT | RIGHT
LENGTH min, max
LOOKUP options
NOECHO
PATTERN 'pattern'
PICTURE 'picture'
PMTEDIT options
PMTTYPE options
[NO]PROMPT {'string' | VARDESC | VARLABEL | VARNAME } HILITE | VIDEO
video_options
RANGE (n,m) ['error message'] .... [OPTIONAL]
REQUIRED
TYPE options
VALUELABELS (n [,m]) 'string'
WIDTH d,w
```

General Clauses:

```
ACTIVITIES permissions
AT [r][,c]
[NO]AUTOTAB
CLEAR 'c'
ERROR message_number [ VIDEO video_options ] 'error message'
LABELS n
PAD [n][,m]
```

BOOLEAN

`BOOLEAN ( expression ) [OPTIONAL]`
Specifies a logical expression that must be true for data to be accepted into the field. After the field has been entered, the expression is evaluated and, if false, an error message (Error 57) is issued and the cursor remains at the field.

If `OPTIONAL` is specified, and the logical expression is false, an error message is displayed to alert the user. The user can modify or accept the field before moving to the next field. If `OPTIONAL` is not specified, the user cannot accept a field which fails the test.

Example: To specify that an entry for `BIRTHDAY` is acceptable only if it falls within the last week, specify:

```
FIELD BIRTHDAY BOOLEAN  -
      (BIRTHDAY GE (TODAY(0)-7) AND BIRTHDAY LE TODAY(0))
```

DATA

`[NO]DATA [ AT r,c] [ HILITE | VIDEO video_options ]`
Specifies attributes for the data area of a field. `NODATA` suppresses the data area of the field. If `NODATA` and `NOPROMPT` are specified nothing is displayed and the cursor position remains unchanged.

`AT`

specifies the starting position (row and column) of data area of field. See AT for syntax.

`HILITE`

specifies the video treatment when cursor is in field.

`VIDEO`

specifies the video treatment of field at other times. See video treatment for video_options.

Example: Highlight the field with bold video when the cursor is positioned in the field. Position the data area below the prompt area.

```
FIELD BIRTHDAY DATA HILITE BOLD AT +1
```

The `"AT + 1"` option on the `DATA` clause positions the data area only. The data area is positioned on the next line, and the column position is not changed. If the `"AT + 1"` was specified as a clause on the `FIELD` instead of as a sub-clause on `DATA`, it would position both the prompt and the data.

DEFAULT

`DEFAULT expression`
Specifies the default value for a field on a new record if the user skips the field. Any arithmetic expression, function or constant can be used.

Example: Set the default for `BIRTHDAY` to today's date.

```
FIELD BIRTHDAY DEFAULT TODAY(0)
```

DISPLAY

```
DISPLAY  ['prompt string' | VARDESC | VARLABEL | VARNAME]
         [VIDEO video_options]
```

Specifies that the field is display only. It cannot be modified nor can new data be entered. The options on `DISPLAY` control the prompt and are the same as the `PROMPT` clause.

`prompt string`

Specifies a string that is used as the prompt.

`VARDESC`

Specifies that the variable name and label are used as the prompt string.

`VARLABEL`

Specifies that the variable label is used as the prompt string. This is the default.

`VARNAME`

Specifies that the variable name is used as the prompt string.

`VIDEO`

Specifies the video treatment of the prompt field. Because the cursor never goes to a display field, only the `VIDEO` clause is applicable. See video treatment for video_options.

DSPEDIT

```
DSPEDIT
   CAPITALIZE
   EDIT 'edit_template'
   LEADING 'leading_string'
   LEFT | RIGHT [FILL 'fill_character']
   LOWERCASE
   MASK {COMMA | DOLLAR | PERCENT}
   MONEY
   SCALE n
   STRIP 'string'
   TRAILING 'trailing_string'
   TRUNCATE n
   UPPERCASE
```

Specifies display edits. `DSPEDIT` alters the way the field is displayed. If an error occurs in the execution of the `DSPEDIT`, an error message is issued and the field is filled with X's. Use the `TYPE` clause to increase the field length to accommodate extra characters.

If the user edits the field, the displayed field, including any characters added or taken out by the display edit, become the input. This must be dealt with, either by the user clearing the field when editing or by an appropriate `PMTEDIT` clause.

`CAPITALIZE`

> Specifies that the first character and every character following a blank, is changed to upper case. Other characters are unchanged.

`EDIT`

> Specifies an "edit template". Specify characters to display and use the circumflex (^) to represent a character of data to achieve the desired format. Characters are displayed as they appear in the template intermixed with data. Strings are left-justified, numerics are right-justified.
>
> If the template specifies more data characters than exist, excess characters are filled with the `FILL` character. If the template specifies less data characters than exist, an error message is issued and the field filled with X's.

`LEADING`

> Specifies a character string prefixed to the data.

`LEFT`

Specifies that the data characters are left-justified. Empty positions are filled with the `FILL` character.

`RIGHT`

Specifies that the data characters are right-justified. Empty field positions are filled with the `FILL` character.

`FILL`

Specifies a single "fill" character. Enclose the character in quotes. Blank is the default.

`LOWERCASE`

Specifies that the text is converted to lowercase. When `LOWERCASE` is used with `CAPITALIZE`, all characters are `LOWERCASE`d then `CAPITALIZE`d.

`MASK`

Specifies a mask applied to numeric values. Specify up to three keywords: `COMMA` inserts commas every three digits to the left of the decimal point. `DOLLAR` prefixes the number with a dollar sign ($). `PERCENT` appends a percent (%).

`MONEY`

Specifies the `DOLLAR` and `COMMA` mask. A dollar sign ($) is prefixed to the number and commas are inserted every three digits to the left of the decimal point.

`SCALE`

Specifies a constant "n" by which the data value is multiplied.

Note: When using a variable defined as scaled in the schema, specify a `SCALE 1` in order to have the scaling computation performed.

`STRIP`

Specifies a set of characters removed from a field before it is displayed. Each character is stripped out as a single character. Enclose the list of characters or each individual character in single quotes.

`TRAILING`

Specifies a character string appended to the data before it is displayed. Enclose the characters in single quotes.

`TRUNCATE`

Specifies that the field is truncated to "n" characters.

`UPPERCASE`

Specifies that all characters are converted to uppercase.

Example: The following template can be used to display a social security number in the format ddd-dd-dddd:

```
FIELD SSN -
      DSPEDIT EDIT '^^^-^^-^^^^'
```

DSPTYPE

```
DSPTYPE
   DATE      'date_format'
   EREAL     n  ,m
   INTEGER   n [,m]
   REAL      n  ,m
   STRING    n
   TIME      'time_map'
   ZINTEGER  n  [,m]
```

Specifies the type of a field for display. Specify one of the following:

`DATE`

Date variable described by the date format. Specify date formats in uppercase. By default, dates are displayed using the format defined in the schema. To display a temporary variable as a date, declare the variable as numeric (real or integer).

```
FIELD BIRTH DSPTYPE DATE 'MM/DD/YY'
```

`EREAL`

Real number n columns wide with m decimal places displayed in exponential format. Specify the field width (using `TYPE`) at least six positions greater than the number of decimal places.

`INTEGER`

Integer number n columns wide, zero filled to m columns wide.

`REAL`

Real number n columns wide with m decimal places.

`STRING`

String variable n characters wide.

`TIME`

Time variable described by the time format. Specify time formats in uppercase. To display a temporary variable as a time, declare the variable as numeric on the `DECLARE` clause of the `FORM` command.

`ZINTEGER`

Zero filled integer number n columns wide. Value m is accepted for compatibility with `INTEGER`, but has no effect. `ZINTEGER` formats integers as zero-filled numbers. The field is zero-filled to the left up to n characters.

FINAL

```
FINAL  COMPUTE           variable = expression, ...
       IF ( expression ) variable = expression, ...
```

Specifies one or more variables computed when the user leaves the field.

Specify multiple `COMPUTE` and `IF` statements if required; only specify the keyword `FINAL` once. If multiple `COMPUTE` and `IF` clauses are used, use commas as delimiters between clauses.

Use `COMPUTE` to simply compute a variable.

Use `IF` to compute a variable under particular conditions. When the `IF` clause is used, variables are computed only when the expression following the `IF` clause is true. The expression can be enclosed in parentheses if desired.

HELP

```
HELP screen_name | 'help_string'
```

Specifies a text string or the name of a help screen to display when help for a specific field is requested by the user.

When the user requests help, if a screen name is specified, that help screen is displayed; if a text string is specified, the string is displayed on the status line at the bottom of the screen.

IF

`IF ( logical_expression )`

Specify an `IF` clause to control whether the cursor moves to the field for data entry. If the expression is true, the cursor moves into the field and data can be entered. If the expression is false, the cursor skips over the field and data cannot be entered.

Example: Allow monthly rent to be entered if `OWNHOME` is not equal to 1.

```
FIELD MONRENT IF (OWNHOME NE 1)
```

INITIAL

```
INITIAL COMPUTE           variable = expression, ...
        IF ( expression)  variable = expression, ...
```

Specifies one or more variables computed when the cursor moves to the field.

Specify multiple `COMPUTE` and `IF` statements if required; only specify the keyword `INITIAL` once. If more than one clause is used, use commas as delimiters between clauses.

Variables can be computed using the `COMPUTE` clause or conditionally computed using the `IF` clause. When the `IF` clause is used, variables are computed only when the expression following the `IF` clause is true.

INSERT | OVERSTRIKE

Specifies that insert or overstrike mode is in effect for this field. Insert specifies that new characters are inserted before the character at the cursor position; overstrike specifies that the character at the current cursor position is overwritten. The user can toggle between these modes. This clause sets the initial mode. Insert is the default for multiple character fields; overstrike is the default for single character fields.

LEFT | RIGHT

Specifies that the cursor is initially at the leftmost character or the rightmost character of the field. Left is the default.

LENGTH

`LENGTH min , max`

Specifies the minimum and maximum length (in characters) of a field. If the user enters fewer characters than the minimum or more characters than the maximum, an error message is issued.

LOOKUP

```
LOOKUP {lookup expression}
      [NOT]ON screen  [/database.record | /tabfile.table]
      BACKWARDS  n | FORWARDS  n
      ERROR  nnn 'message'
      IF ( expression )
      IN destination_expression
      LOCK locktype
      OPTIONAL | REQUIRED
      REPORT 'message'
      RETURNING expression, ...  TO variable, ...
      USING caseid  [ * ] | [ key, ...]
      VIA             *   |   key, ...
```

Uses another screen to locate a record. Use `LOOKUP` to establish the presence or absence of a record or to retrieve data from the record that is looked up. Use `LOOKUP` to verify the field data by testing stored value(s) in a record. `LOOKUP` can perform a number of functions:

- It can test whether a specified record is present. Specify the keys of the record with `USING` or `VIA` clauses. If keys are not specified, the first record of that type is used.
- It can test whether a specified record is not present. Specify the keys of the record with the `USING` or `VIA` clauses.
- It can test the presence or absence of one of a set of records or of the nth record. Specify a partial key and the data is searched (forwards or backwards) from the point where the specified keys start.

- It can test the current field to be equal to a variable in another record.

- It can test an expression composed of fields from this record against an expression composed of variables from another record.

- It can retrieve data from another record into variables in this record or into temporary variables.

`lookup expression`

Specifies a value which is tested against the retrieved data (specified by the `IN` clause). By default, the field value is used. For example, if this field were `DOSAGE,` and this has to be multiplied by the doses per day to check the value on a reference record, specify:

```
FIELD DOSAGE -
      LOOKUP DOSAGE*QTY ON DRUGREF -
      USING -1,DRUGID IN MAXDOSE
```

`[NOT]ON`

Specifies the screen which defines the record or table to be used. This clause (`ON` or `NOTON`) is the only required clause. The clause references the *screen name.* If the screen name is not the same as the record or table, the record name or table name MUST be quoted as part of the screen name. Specify the database or tabfile if they are not the default.

`ON` checks that the record exists and returns an error message if a record matching the search criteria is not found.

`NOTON` checks that no records matching the search criteria exist. `NOTON` returns an error message if a record matching the search criteria is found. For example, when adding new people, the social security number could be tested to be `NOTON` an inverted list of current social security numbers.

`BACKWARDS n|`
`FORWARDS n`

Sets the direction of a record search. If a full key is not specified in the `USING` or `VIA`, the direction of search may be relevant. The default direction is `FORWARD.` If a number is specified, the nth record matching the search criteria is selected.

`ERROR nnn`

Specifies alternative text for error messages.
59 is "Lookup Failure"; 60 is "Lookup Warning".

`IF`

Specifies an expression that controls whether the `LOOKUP` clause is executed. If the expression is true, the `LOOKUP` is performed. If it is false, the `LOOKUP` is not performed.

`IN`

Specifies an expression which is tested against the current field or the source expression. If a match is not found, an error message is issued. Use variables from the screen being looked up.

If a full key is specified with a `VIA` or `USING`, `IN` tests on that record. If a partial key is specified, the set of records is searched for a match. If a number is specified on `FORWARDS` or `BACKWARDS`, the nth matching record is used.

`LOCK`

Specifies the lock the called screen uses when more than concurrent operations (more than one user accessing the database) are envisaged. There is no reason ever to have a lock type other than 'CR' (concurrent read) for lookups.

`OPTIONAL |`
`REQUIRED`

If the lookup fails and the lookup is `REQUIRED`, the value is not accepted and must be re-entered. Specify `OPTIONAL` to issue a warning message, letting the user continue and accept the field. `REQUIRED` is the default.

`REPORT`

Specifies a constant (a string), which is displayed when the lookup is successful. To display individual values from the lookup record, use `RETURNING` to bring those values into fields that can be referenced and displayed on this screen.

`RETURNING`

Specifies the value(s) (variable names or expressions) from the lookup record that are transferred to the fields in the `TO` list. The `RETURNING` and the `TO` list are positionally matched. The first variable in the `RETURNING` list is moved to the first variable in the `TO` list, the second to the second and so on. Variables referenced in the `RETURNING` list must be available to the lookup record; variables in the `TO` list must be available to the current record.

`USING`

Specifies that the `LOOKUP` uses a different case in a case structured database. Use `VIA` to lookup records in the current case. Specify the case id value, optionally followed by the key fields. Specify an asterisk (*) to use the key fields of the current screen. Do not specify `USING` for caseless databases nor in conjunction with the `VIA` clause.

`VIA`

Specifies the key fields for a lookup within this case on a case structured database, for any lookup on a caseless database or a lookup to an indexed `TABLE` screen. Specify values for the key fields. Specify an asterisk (*) to indicate that the key fields of the current screen are to be used. If the specified key(s) are not explicit to one record, but define a set of records, the first record which matches the `IN` clause is used. If there is no `IN` clause, the first record in the set is used. (This can be changed from the first to the last or the nth with the `FORWARD` and `BACKWARDS` clauses).

Lookup examples:

A) This checks that a record (`LOCREC`) exists with a case id of -1 that has a key of `LOCATION` and gives an error message if the record is not there. The lookup screen is also called `LOCREC` and is in the form definition solely to be used as a lookup. It has just the keys defined and is never displayed.

```
FIELD LOCATION -
      LOOKUP ON LOCREC USING -1, LOCATION-
      ERROR 59 'Invalid Location Code - Please Re-enter'
..
RECORD LOCREC / COMPANY.LOCREC
  field ID
END RECORD
```

B) This checks for the opposite condition from A, that the record does not already exist and gives an error message if this is not a new location.

```
FIELD LOCATION LOOKUP NOTON LOCREC USING -1, LOCATION -
      ERROR 59 'Location Code Already Exists - Please Re-enter'
```

C) This checks the record is present and that `STATUS` (a field on the location record) equals 1.

```
FIELD LOCATION 1 LOOKUP ON LOCREC USING -1, LOCATION-
      IN STATUS -
      ERROR 59 'Location Not Found or Not Active'
```

D) This returns the location status to a temporary field `XSTATUS` (which has been `DECLARE`d on the `FORM` command) and then displays the result.

```
FIELD LOCATION LOOKUP ON LOCREC USING -1, LOCATION-
      RETURNING STATUS to XSTATUS -
      ERROR 59 'Invalid Location Code - Please Re-enter'
FIELD XSTATUS DISPLAY 'Location Status'
```

If the screen which referenced the record `LOCREC` had a different name, say `XLOCREC`, for some reason, the lookup clause must name both the record and screen name:

```
RECORD XLOCREC / COMPANY.LOCREC
FIELD LOCATION LOOKUP ON XLOCREC / COMPANY.LOCREC
```

NOECHO

`NOECHO`

`NOECHO` suppresses the display of input for a field as it is entered and subsequent displays of the data portion of the field.

PATTERN

```
PATTERN 'edit pattern'
```

`PATTERN` specifies an edit for validating input data. If the value matches the pattern, the value is accepted. If it does not match, an error message is issued (Error 113) and the value is not accepted. An edit pattern is specified as characters that must match exactly and the percent sign (%), which indicates that from zero to 'n' characters are not tested. Character codes, such as those on the PICTURE clause, are not used. Both `PATTERN` and `PICTURE` clauses can be specified for the same field.

Example:

```
PATTERN '%-%'
PATTERN '%-%-%'
PATTERN 'E%/%E'
```

The first example tests that the input has a "-" somewhere in it. The second tests that there are two "-", possibly separated by other characters. The third example tests that the input begins and ends with an "E" and has a "/" somewhere in the middle.

PICTURE

`PICTURE 'picture string'`
`PICTURE` specifies a picture string for validating user input. The data is accepted when the value entered by the user matches the picture. If it does not match, an error message (Error 16) is issued and the data is not accepted. Any picture specifications are added to the default help screen for the field.

The picture must be the same length as the input field. When changing the length of the field to match the picture, the `TYPE` clause must precede the `PICTURE` clause.

The picture is composed of characters that must be matched exactly and character codes for each position in the input string. The character codes are:

```
a any letter
d any digit
n any letter or digit
s a numeric value (0-9, decimal point, +, -, E)
u any uppercase letter
l any lowercase letter
x any character
```

Using uppercase or lowercase for the codes determines whether or not blanks are also allowed. Codes in lowercase specify that blanks are not allowed. Codes in uppercase allow either the specified character type or a blank. For example, "A" indicates any letter or a blank; "D" indicates any digit or a blank; and so on. "x" matches anything except blank; "X" matches anything.

Example: The following `PICTURE` clause validates input of a social security number. The expected picture is 3 digits, a dash, 2 digits, a dash, and 4 digits (eg., 123-54-6789).
`PICTURE 'ddd-dd-dddd'`

PMTEDIT

```
PMTEDIT
    CAPITALIZE
    EDIT 'edit_template'
    LEADING 'leading_string'
    LEFT | RIGHT [FILL 'fill_character']
    LOWERCASE
    MONEY
    SCALE n
    STRIP 'strip_string'
    TRAILING 'trailing_string'
    TRUNCATE n
    UPPERCASE
```

The `PMTEDIT` clause specifies permanent edits (**P**er**M**anen**T EDIT**s) that are applied to the value in the variable. The value entered is edited according to the specified options and the edited value is stored in the database. The editing also updates the data on the screen. These edits are applied to the field whenever it is modified.

If the user inputs a field once and then edits it for input again without clearing it completely and re-entering it, this may cause problems. `PMTEDIT` updates the field and, once the field is edited and re-input, all characters are read again. A `PMTEDIT` such as `LOWERCASE` is no problem because making a field lowercase twice does not impact it; however clauses such as `EDIT, LEADING, SCALE` and `TRAILING`, all update the field. If the field is edited again, these clauses are applied again and may give unexpected results.

Any edits applied to a numeric field must produce a resulting legal numeric value (digits, a plus or minus character and/or a decimal point).

`CAPITALIZE`

> Specifies that the first character of the string and every character following a blank is capitalised. Other characters are unaffected.

`EDIT`

> Specifies a template that is used to edit the data. The circumflex (^) represents one character of the data value. Any other characters are inserted exactly as they appear in the template. The number of circumflex determines how many characters are going to be able to be entered.

`LEADING`

Specifies a character string that is prefixed to the data.

`LEFT`

Specifies that the entered characters are left-justified. Any unoccupied field positions are filled with the `FILL` character.

`RIGHT`

Specifies that the entered characters are right-justified. Any unoccupied field positions are filled with the `FILL` character.

`FILL`

Specifies the fill character. Blank is the default fill character.

`LOWERCASE`

Specifies that all characters are converted to lowercase. When `LOWERCASE` is used with `CAPITALIZE`, all characters that are not capitalised are lowercase.

`MONEY`

Specifies that the dollar sign ($) and commas associated with a numeric field are removed. If `MONEY` is not specified, dollar sign and commas are illegal in a numeric field. Increase the width of the field with the `TYPE` clause to allow for these characters.

`SCALE`

Specifies a constant "n" by which a numeric value is multiplied. If the field is defined as a scaled variable in the schema, a scale factor of 1 specifies that the dictionary scaling is applied.

`STRIP`

Specifies characters to be stripped from the field. Each character is treated as a single character and is stripped from the string wherever it occurs.

`TRAILING`

Specifies a character string that is appended to the field.

`TRUNCATE`

Specifies that the field is truncated to "n" characters.

`UPPERCASE`

Specifies that the field is converted to uppercase.

Example: A field for entering a social security number may be defined with hyphens or slashes. The following strip clause removes these:

```
field SS pmtedit strip '-/'
```

PMTTYPE

```
PMTTYPE
    DATE     'date_format'
    EREAL    n , m
    INTEGER  n
    REAL     n , m
    STRING   n
    TIME     'time_map'
```

`PMTTYPE` changes the type and width of a field. When a variable is defined in the database or in a table, it can have a standard external format. For example, an integer might be defined as 4 characters long. The `PMTTYPE` allows for additional field positions which are not stripped out by one of the edit clauses.

(The `DSPTYPE` changes the width of the display field but not the number of digits. This allows for additional characters such as dollar signs or other edit characters.) The `TYPE` clause changes both the `DSPTYPE` and the `PMTTYPE`.

Fields cannot be changed from numeric to string or vice versa with this clause. To specify a `DATE` or `TIME` type for a temporary variable, declare the variable as real on the `DECLARE` clause of the `FORM` command.

`DATE`

> Specifies a date integer described by a date format.

`EREAL`

> Specifies a real number n columns wide with m decimal places. `EREAL` specifies that numeric data is displayed in scientific format. The width must be at least six positions greater than the number of decimal places.

`INTEGER`

> Specifies a integer number n columns wide

`REAL`

> Specifies a real number n columns wide with m decimal places

`STRING`

Specifies a string variable n characters wide

`TIME`

Specifies a time variable described by a time format

PROMPT

```
[NO]PROMPT  [ 'prompt-string' | VARDESC | VARLABEL | VARNAME
              HILITE | VIDEO video_options
              AT [r] [,c]  ]
```

`PROMPT` specifies how the prompt for this field is displayed. `NOPROMPT` suppresses the prompt for the field. If both `NOPROMPT` and `NODATA` are specified, nothing is displayed and the cursor position remains unchanged.

`prompt string`

Specifies the prompt.

`VARDESC`

Specifies that the variable name and label are the prompt.

`VARLABEL`

Specifies that the variable label is the prompt. This is the default.

`VARNAME`

Specifies that the variable name is the prompt.

`HILITE`

Specifies the video attributes when the cursor is at this field.

`VIDEO`

Specifies the video options when the cursor is not at this field (See video treatment for video attributes).

`AT`

Positions the prompt (See AT).

RANGE

```
RANGE (value1 [,value2])  ['error message']
      (value3 [,value4])  ['error message'] ...  [OPTIONAL]
```

`RANGE` (or `RANGES`) specifies ranges for edit checking. Ranges must be within any valid ranges specified for the variable on the data dictionary. The values specified can be constants or the keywords `LO` or `HI` meaning the lower and upper valid value respectively. Variable names or expressions are not allowed.

If the value entered is within a range, the value is accepted and the cursor moves on to the next field. If the value entered is within a range with a message, the message is issued and the user has a chance to correct or accept the value. If the value entered is outside all ranges, an error message is issued and the value is rejected unless `OPTIONAL` is specified. This allows the user to accept a value outside any of the ranges specified here. The user cannot enter a value outside any ranges specified in the data dictionary. Specify `OPTIONAL` once, after the final range.

Multiple ranges can be specified. The ranges must be in ascending order and cannot overlap.

Example: `HEIGHT` is defined in the data dictionary with the range 48 to 84 inches. The `RANGE` is specified to warn the user when values below 60 or above 78 inches are entered.

```
FIELD HEIGHT -
      RANGES    (LO, 60) 'Are you sure they are this short ?'-
                (61, 78)  -
                (79, HI) 'Are you sure they are this tall ?'
```

REQUIRED

`REQUIRED`

Specifies that the field is required. The user cannot skip this field and cannot go on to the next field without entering data. The user cannot clear a required field. If the required field is within a group, the group can be skipped but the user cannot save a record having skipped any required fields. (Error message 20)

Keys are required by default and need not be specified as `REQUIRED`.

Example : To ensure that the variable `NAME` is always entered:

```
FIELD NAME REQUIRED
```

This tests that the user has entered a value. This value may be blanks since blanks is a valid character string and is a valid missing value. To exclude blank names as well:

```
FIELD NAME REQUIRED -
      BOOLEAN (EXISTS(NAME)) -
      Error 57 'You must enter a name for an employee' -
      Error 20 'You must enter a name for an employee'
```

TYPE

```
TYPE
    DATE     'date_format'
    EREAL    n   , m
    INTEGER  n [ , m ]
    REAL     n   , m
    STRING   n
    TIME     'time_map'
    ZINTEGER n [ , m ]
```

Specifies the type and width of the field. The default is the data dictionary definition. `TYPE` does not affect the way the variable is stored in the database; it is always stored according to the data dictionary. The following types can be defined:

`DATE`

Specifies a date variable described by a date format. If a `DATE` type is specified for a temporary variable, that variable must be declared as real on the `DECLARE` clause on the `FORM` command.

`EREAL`

Specifies a real number n columns wide with m decimal places. `EREAL` specifies that numeric data is displayed in scientific format. The width must be at least six positions greater than the number of decimal places.

`INTEGER`

Specifies an integer number n columns wide, zero filled to m columns wide.

`REAL`

Specifies a real number n columns wide with m decimal places.

`STRING`

Specifies a string variable n characters wide.

`TIME`

Specifies a time variable described by a time format. If a `TIME` type is specified for a temporary variable, that variable must be declared as real on the `DECLARE` clause of the `FORM` command.

`ZINTEGER`

Specifies a zero filled integer number n columns wide. Value m is accepted for compatibility with `INTEGER` but has no effect. `ZINTEGER` formats integers as zero-filled numbers.

VALUELABELS

`VALUELABELS ( value [ , value ] ) [ 'string' ], . . .`
Specifies value labels for variables. If a variable is a temporary variable or does not have value labels, the `VALUELABELS` clause provides a means of defining them. If a variable already has value labels defined in the database, the `VALUELABELS` clause provides alternatives.

Example: To specify value labels for a temporary variable named `AGECAT` (age category):
For values 1 through 3, display "Children".
For values 4 through 6, display "Young Adults".
For values 7 through 9, display "Adults".

```
FIELD  AGECAT -
       VALUELABELS (1 ,3) 'Children' -
                   (4 ,6) 'Young Adults' -
                   (7 ,9) 'Adults'
```

WIDTH

`WIDTH d,w`

Specifies the display width d, and the total width w. The display width is the width seen on the screen, the total width is the maximum number of characters that can be entered.

For string fields only, if the total width is larger than the display width, the user can scroll horizontally through the field with the right and left arrows. This allows entry, editing and display of long strings in a limited area of the screen.

CALL

The `CALL` command passes control from one screen to another. `CALL` can be invoked automatically or under user control. `CALL` is used to determine the flow of control through a form.

See syntax.

You can specify text for the prompt for a `CALL` or allow the default which is the name of the screen to be CALLed.

Calls can be made conditionally by adding an `IF` clause to the `CALL` command.

The call can be automatic without offering the user a choice with `IF (...) AUTO CALL`

Control is returned to the calling screen either when the user enters the `RETURN` command in the called screen or because of options specified on the `CALL` command. On return from a `CALL`, there are a number of options as to how the calling screen behaves. This control makes `CALL` a very powerful command. It is possible to `CALL` a screen which performs actions and then returns without the user ever seeing a different screen.

Record screens and table screens have one single active record; to display data from multiple records on a single screen, use the `CALL` or `LOOKUP` clauses to get the data.

A screen can be called from a number of different screens.

Moving between screens

When control is passed to another screen with `CALL`, control moves away from this screen and, if a record or row has been updated, it is written. If the `AUTO WRITE` option is not on this screen, the user is asked whether it is "OK" to write this screen. (A `LOOKUP`, by contrast, does not move away from a screen, and does not force a write of this screen.)

Passing Keys

A called screen can be left blank for the user to enter keys of records or the key fields can be passed specifying the record(s) that the called screen is to reference.

The `VIA` clause passes values from the calling screen that are used as the keyfields in the called screen. This means that only records with the specified key values are accessed, and the user does not have to re-enter key values on the called screen.

In a case structured database, the `USING` clause accesses another case by passing a value for the case id (plus any other keys), to the called screen. In a case structured database, a `CALL` from a record screen without a `USING` clause, automatically restricts the called screen to records within the same case. The case identifier is used as the key for accessing other records from any record type. For example, the command `CALL OCCUP` from the `EMPLOYEE` screen, retrieves `OCCUP` records only for the employee whose record has been referenced.

Automatic record access

A `CALL` to a record or table screen where there are no fields to be displayed is an invaluable technique for creating or updating records automatically. The effect of this is that the record or row is accessed, updated if necessary, and control returned, without the user ever being aware that a screen was called.

It can be used to take data from menu screens and write it to the database automatically; to maintain control records; to assign sequential numbers to new input, to maintain inverted lists; etc. For example, suppose that an inverted list is needed on `NAME`. The `CALL NAMEREF` writes a record and returns without showing the user anything.

```
FORM ....  AUTO WRITE  error 65 video nobell '' -
.......
RECORD PERSON
   FIELD ID
   FIELD NAME REQUIRED
   CALL NAMEREF USING -1,NAME,ID -
        AUTO CALL RETURN NODATA NOPROMPT
.......
RECORD NAMEREF
   FIELD ID NODATA NOPROMPT
   FIELD NAME NODATA NOPROMPT
   FIELD XID NODATA NOPROMPT
ENDRECORD
```

Using CALL for control

When a `CALL` returns to the calling screen, the `ONRETURN` clause specifies a command that is automatically executed. This means that a dummy `CALL` is sometimes useful to take advantage of this capability. For example, to RESTART a record screen automatically;

```
FORM .....  -
     DECLARE INTEGER DUMMY -
     ......
RECORD PERSON
     FIELD ...
     ......
* Restart automatically when the cursor gets here
     CALL DUMMY NODATA NOPROMPT AUTO CALL RETURN -
          ONRETURN RESTART -
          PASSING 1
```

```
ENDRECORD
MENU DUMMY RECEIVING DUMMY
    FIELD DUMMY NODATA NOPROMPT
ENDMENU
```

A similar technique can be used to issue any of the user commands (`EXIT, RETURN, TOP,` etc.) automatically. An `IF` clause can be used on the `CALL` command to determine whether the `CALL` is performed.

CALL Stack

A "stack" is maintained and each time a `CALL` is issued an additional level is created on the stack with information about every variable. When control returns to the calling screen, the stack level is removed. Avoid sequences of nested `CALL` commands which potentially never return.

If the application needs to pass control between screens at the same level, do not specify `CALL`s where each screen calls the other screens. Use a menu which calls all the other screens and pass control through this menu each time. (With correct specifications, the control menu screen can be hidden from the user if so required.)

Locked records normally only happen when operating in a concurrent environment and another person has accessed the record. It is possible to lock a record from yourself because of a `CALL` or `LOOKUP` of a record currently in the stack. With case structured databases, there is a Common Information Record or 'CIR' for each case. If a `CALL` or a `LOOKUP` has a `USING` clause, this expects to access a new case and to retrieve the CIR. If the case is already referenced in the stack, it causes a locked record on the CIR.

CALL

The only required clauses on the `CALL` is the called screen_name. Available clauses are:

Clauses:

CALL `screen_name`
AUTO `[CALL [IF ( expression ) ]] [DELETE] [RETURN ] [WRITE]`
CLEAR `'c'`
DATA `[AT r , c] [HILITE | VIDEO options]`
FINAL `COMPUTE var=exp, ...`
HELP `screen name | 'help text'`
IF `( expression )`
INITIAL COMPUTE `var=exp, ...`
LOCK `lock_options`
MODE `[FIND | SEARCH | VERIFY ] [EDIT | NOEDIT]`
NODATA
NOPROMPT
ONCALL `FIRST | LAST`
ONRESTART `EXIT | RETURN | STOP`
ONRETURN `EXIT | FIRST | LAST NEXT | PREVIOUS | RESTART RETURN | STOP | TOP UPDATE | WRITE`
PASSING `expression, ...`
PROMPT `'prompt' HILITE | VIDEO options`
USING `caseid, [ * | key, ...]`
VIA `* | key, ...`

General Clauses:

```
ACTIVITIES permissions
AT [r][,c]
PAD [n][,m]
```

`screen_name`

> The `CALL` references another screen that is included in this form. Use the screen name *not* the record or table name.
>
> If the name is specified in quotes, this is taken as a command string to pass to the operating system via an Escape

.

AUTO

```
AUTO  [CALL [IF (expression)]]
      [DELETE]
      [RETURN]
      [WRITE]
```

The `AUTO` clause specifies actions that are automatically performed without user intervention when the `CALL` statement is executed.

`CALL`

Specifies that the called screen is gone to automatically without an user decision. The call happens when the cursor moves through this command. To hide the choice from the user completely, also specify `NODATA` and `NOPROMPT`.

`IF`

Specifies a logical expression that controls whether the `AUTO CALL` is executed. If the expression is true, the `AUTO CALL` is performed. If it is false, the `AUTO CALL` is not performed. If the `CALL` data field is displayed on the screen, the user can still choose to execute the `CALL`.

`DELETE`

Specifies that the record or table row referred to by the called screen is deleted when the user moves away from the screen. In conjunction with `AUTO RETURN`, this deletes records automatically.

`RETURN`

Specifies that control is automatically returned to this screen when the user moves through the last field or group on the screen being called. If this is not specified, the user stays on that screen and can enter multiple records. Control is passed back when the user issued a `RETURN` command.

`WRITE`

Specifies that the called screen is written automatically when the user moves away from the screen.

Example: To specify that `INDEX` is called without user intervention and control is passed back without user intervention. (might be to write a record automatically.)

```
CALL INDEX NODATA NOPROMPT -
     AUTO CALL RETURN WRITE
```

CLEAR

`CLEAR 'c'`

Specifies the single character used in the `CALL` data field. A question mark (?) is the default.

DATA

```
[NO]DATA
    [AT [ r ] [, c ]]
    [HILITE | VIDEO  video_options]
```

Specifies attributes for the data area of the `CALL`. `NODATA` specifies that the data area is not present and the user does not have the opportunity to select whether or not to perform the `CALL`.

`AT`

Specifies where on the screen the `CALL` data field is displayed.

`HILITE`

Specifies the video treatment of the `CALL` data field when the cursor is in it.

`VIDEO`

Specifies the video treatment of the `CALL` data field when the cursor is not in it.

`video_options`

See video treatment.

FINAL

```
FINAL COMPUTE   variable = expression, ...
      IF (expression) variable = expression, ...
```

Specifies variables which are computed after the `CALL` returns to this screen. The variables to be computed are variables referenced by this screen, (not by the called screen). The final compute is equivalent to the final compute on a `FIELD` command. Multiple `COMPUTE` and `IF` statements can be specified; specify the keyword `FINAL` once. Variables can be computed using the `COMPUTE` clause or conditionally computed using the `IF` clause. When the `IF` clause is used, variables are computed only when the expression following the `IF` clause is true. If multiple `COMPUTE`s and `IF`s are specified, commas are used as delimiters between specifications.

```
CALL INDEX -
     FINAL COMPUTE    MFLAG = 1, -
           IF (DFLAG = 2) DDATE = TODAY(0)
```

HELP

`HELP screen_name | 'help_string'`
Specifies a text string or the name of a help screen to display when help is requested by the user when positioned at the `CALL` field.

When the user requests help, if a screen name is specified, that help screen is displayed; if a text string is specified, the string is displayed on the status line at the bottom of the screen.

IF

```
IF ( expression )
```

`IF` specifies a logical expression that controls whether the `CALL` command is executed. If the expression is true, the `CALL` can be performed. If it is false, the `CALL` is not allowed. The `IF` clause is evaluated before any of the other clauses.

```
CALL NEXTOFKIN IF (RELATIVE = 1)
```

INITIAL

```
INITIAL  COMPUTE   variable = expression, ...
         IF (expression),  variable = expression, ...
```

The `INITIAL` clause specifies one or more variables computed before the screen is `CALL`ed. The variables to be computed are variables referenced by this screen (not the called screen). The initial compute is equivalent to the initial compute on a `FIELD` command. Multiple `COMPUTE` and `IF` statements can be specified; specify the keyword `INITIAL` once. Variables can be computed using the `COMPUTE` clause or conditionally computed using the `IF` clause. When the `IF` clause is used, variables are computed only when the expression following the `IF` clause is true. If multiple `COMPUTE`s and `IF`s are specified, commas are used as delimiters between specifications.

LOCK

`LOCK` specifies the lock the called screen uses when more than concurrent operations (more than one user accessing the database) are envisaged.

MODE

```
MODE  [FIND      |  SEARCH ]
      [VERIFY    |  NOVERIFY ]
      [EDIT      |  NOEDIT ]
```

`MODE` specifies whether the called screen is in `FIND` or `SEARCH` mode, `VERIFY` or `NOVERIFY` and `EDIT` or `NOEDIT` mode. The default is `FIND, NOVERIFY, EDIT`. If the default has been changed, and a mode is not specified, the called screen takes the mode of the calling screen.

`FIND | SEARCH`

Sets `FIND` or `SEARCH` mode.

`VERIFY | NOVERIFY`

Sets or clears `VERIFY` mode.

`EDIT | NOEDIT`

Sets `EDIT` mode. Fields can be edited in `EDIT`. In `NOEDIT` mode, fields must be completely re-entered to change them.

ONCALL

`ONCALL FIRST | LAST`
Specifies that either the `FIRST` or `LAST` record in the set of records available to the called screen is retrieved and displayed.

ONRESTART

`ONRESTART EXIT | RETURN | STOP`
Specifies an action which is performed if the called screen is `RESTART`ed by the user or by an `ONRETURN RESTART` clause of a `CALL` command.

`EXIT`

Terminates the SirForms session.

`RETURN`

Returns control to the calling screen.

`STOP`

Returns control to the last stop screen which the user passed. The root screen is always the highest level stop screen.

ONRETURN

```
ONRETURN
  EXIT
  FIRST
  LAST
  NEXT
  PREVIOUS
  RESTART
  RETURN
  STOP
  TOP
  UPDATE
  WRITE
```

Specifies a command which is executed when control is returned to the calling screen. Only one command may be specified. This is equivalent to the user entering this command when control returns to this screen, except that it is done automatically.

`EXIT`

Terminates the SirForms session.

`FIRST`

Gets the first record for this screen.

`LAST`

Gets the last record for this screen.

`NEXT`

Gets the next record for this screen.

`PREVIOUS`

Gets the previous record for this screen.

`RESTART`

Displays an initial screen with the cursor at the first field.

`RETURN`

Returns control to the screen that called the current screen.

`STOP`

Returns control to a stop screen.

`TOP`

Displays the current record with the cursor at the top of the screen.

`UPDATE`

Writes the record to the database, leaving the screen unchanged. The record is automatically written when a screen is CALLed. `UPDATE` only has effect if the data in this record is altered as a result of the `CALL`.

`WRITE`

Writes the record to the database and restarts the screen.

PASSING

`PASSING expression, ...`

Specifies a list of values which are passed to the CALLed screen. Values can be variable names, constants or expressions. The values passed are received by specifying the `RECEIVING` clause in the screen being called. If the passing list is longer than the receiving list, additional entries in the passing list are ignored. Keys may appear on the `PASSING` clause but cannot be received into keys on the CALLed screen. Pass keys with the `VIA` or `USING` clauses of the `CALL` command.

```
CALL INDEX USING -1,NAME,ID -
     PASSING GENDER
 .
 .
 .
RECORD INDEX RECEIVING SEX
```

PROMPT

`[NO]PROMPT [HILITE | VIDEO video_options]`

Specifies the treatment of the prompt for the `CALL`. `NOPROMPT` suppresses the display of a prompt. If `NOPROMPT` and `NODATA` are specified, nothing is displayed and the cursor position is not altered. `AT` specifies the row and column position where the prompt display commences. (See AT.)

The prompt is displayed followed by the data area which displays a "?" or the specified `CLEAR` character. If a `PROMPT` is not specified, the name of the CALLed screen is displayed.

`HILITE`

Specifies the video treatment of the prompt when the cursor is in the field.

`VIDEO`

Specifies the video treatment of the prompt when the cursor is not in the field. (See video treatment.)

USING

`USING expression,...`

`USING` specifies that the `CALL` uses a different case. Specify the case id value, optionally followed by the key field values. Specify an asterisk (*) to indicate that the key fields of the current screen are to be used. `USING` references a different case and is not used for databases without case structures or in conjunction with the `VIA` clause.

Example: To call a screen `INDEX` which references a new case with a case id of -1 and two other keyfields:

```
CALL INDEX USING -1,NAME,ID
```

VIA

```
VIA expression, ...
```

`VIA` specifies the key fields for a `CALL` within this case, to another record on a caseless database or to a table screen. Specify values for the key fields. The values can be arithmetic expressions, constants or names of fields in the calling screen. The values are displayed in the called screen.

Specify an asterisk (*) to indicate that the key fields of the current screen are to be used. The asterisk can be followed by other values.

If one or more of the lower level keys are omitted, all records with the specified keys are accessible.

Do not specify a case id on the `VIA` clause. Do not use `USING` and `VIA` on the same `CALL` command.

Example: To access associated records in the `REVIEW` Record screen from the `OCCUP` screen. The `VIA *` option specifies that the caseid and all keys of the Record screen `OCCUP` are used in accessing records in the `REVIEW` screen. The user can now retrieve or create records which "belong" to the particular `OCCUP` record.

```
CALL REVIEW VIA *
```

Escape

```
CALL {'command' | expression}
     NOCLEARSCREEN
     [CALL options]
```

The `CALL` command can be used to "escape" SirForms and run another process. After the completion of the escaped process, control returns to SirForms.

Specify a constant as a quoted string or a string expression which is the command that is passed to the operating system. If the expression is simply a variable name, enclose it in parentheses () to avoid confusion between an expression and a screen name.

`NOCLEARSCREEN`

> Specifies that the screen is not cleared when an 'escape' call is made. The screen is cleared by default.

`CALL options`

> Specify any of the CALL options as with the standard form of CALL.

Example, to exit SirForms and print a file, specify a CALL such as:

```
CALL 'PRINT output.lst' -
      prompt 'Print Output File on Printer ?'
```

LOOKUP

`LOOKUP` uses another screen to locate a record. If the specified record is not found, an error message is issued. The `LOOKUP` can reference or retrieve data from the specified record. See syntax.

The stand-alone `LOOKUP` command is very similar to the `LOOKUP` clause of the `FIELD` command.

`LOOKUP` can perform a number of functions:

- It can test whether a specified record is present. Specify the keys of the record with `USING` or `VIA` clauses. If keys are not specified, the first record of this type is used.
- It can test whether a specified record is not present. Specify the keys of the record with the `USING` or `VIA` clauses.
- It can test the presence or absence of one of a set of records or of the nth record. Specify a partial key and the data is searched (forwards or backwards) from the point where the specified keys start.
- It can test an expression composed of fields from this record against an expression composed of variables from another record.
- It can retrieve data from another record into variables in this record or into temporary variables.

Syntax

The `LOOKUP` command must specify the `ON screen_name` clause. Available clauses are:

```
LOOKUP {lookup expression}
   [NOT]ON screen_name[ /database.record | /tabfile.table]
   BACKWARDS | FORWARDS  n
   ERROR nnn 'message'
   IF ( expression )
   IN  expression
   LOCK locktype
   REPORT 'message'
   RETURNING expression, ...  TO variable, ...
   USING caseid  [ * ] | [ key, ...  ]
   VIA             *   |   key, ...
```

`lookup expression`

> If a lookup expression is specified, it is tested against the expression specified on the `IN` clause. If a lookup expression is not specified, do not specify an `IN` expression.

`[NOT]ON`

> Specifies the screen which defines the record or table to be used from the `LOOKUP`. `ON` or `NOTON` is the only required clause. `ON` checks that a record matching the search criteria exist. `NOTON` returns an error message if a record matching the search criteria exists.

`screen_name`

> Specifies the name of another screen in this form. If the name of the screen is different from the record or table name to which it refers, specify the extended form of the name with the record name and the database name if necessary.

`BACKWARDS n|`
`FORWARDS n`

> Sets a record search direction. When the `USING` or `VIA` specifies a partial key, the direction of search can be set. The default direction is `FORWARD`. When a number is specified, the nth record matching the search criteria is selected.

`ERROR`

> Specifies the error messages (and video treatment) to replace the default error messages. Error 59 is 'Lookup Failure' and error 60 is 'Lookup Warning'.

`IF`

Specifies an expression that controls whether the `LOOKUP` command is executed. If the expression is true, the `LOOKUP` is performed. If it is false, the `LOOKUP` is not performed.

`IN`

Specifies an expression using variables from the screen being looked up which is tested against the source expression. If they are not equal, an error message is issued. If the `USING` or `VIA` clause specifies a complete key, that record is tested. If the `USING` or `VIA` clause, specifies a partial key, the defined set of records is searched for a match.

`LOCK`

Specifies the lock the called screen uses when more than concurrent operations (more than one user accessing the database) are envisaged. There is no reason ever to have a lock type other than 'CR' for lookups.

`REPORT`

Specifies a string that is displayed when the lookup is successful. To display values from the lookup record, use `RETURNING` to copy the values to fields that are displayed on this screen.

`RETURNING`

Specifies the value(s) (variable names or expressions) from the `LOOKUP` record that are transferred to the fields in the `TO` list. The `RETURNING` and the `TO` list are positionally matched. The first variable in the `RETURNING` list is moved to the first variable in the `TO` list, the second to the second and so on. Variables referenced in the `RETURNING` list must be available to the `LOOKUP` record; variables in the `TO` list must be available to the current record.

`USING`

Specifies that the `LOOKUP` uses a different case. Specify the case id value, optionally followed by other key field values. Specify an asterisk (*) to indicate that the key fields of the current screen are to be used. `USING` specifies a different case and thus cannot be used for databases without case structures and cannot be used in conjunction with the `VIA` clause.

`VIA`

> Specifies the key fields for a `LOOKUP` within this case, to another record on a caseless database or to a table screen. Specify values for the key fields. Specify an asterisk (*) to indicate that the key fields of the current screen are to be used. If the key is not unique for one record, but defines a set of records, the first record in the set is used. (This can be changed with the `FORWARD` and `BACKWARDS` clauses).

Since the stand-alone lookup is not positioned on an input field, the test cannot be "required" since control has to continue to the next field. Use the lookup clause on the field command, to enforce the checking and re-entry of fields. The stand alone lookup is normally used to retrieve data.

Example: This returns the location status to a temporary field `XSTATUS` (which has been `DECLARED` on the `FORM` command) and then displays the result.

```
LOOKUP ON LOCREC USING -1, LOCATION -
       RETURNING STATUS to XSTATUS -
       ERROR 59 'Invalid Location Code - Please Re-enter'
FIELD XSTATUS DISPLAY 'Location Status'
```

COMPUTE

COMPUTE assigns the value of an expression to a variable. IF is similar to COMPUTE and assigns a value to a variable but only when a particular condition is true.

These are stand alone commands where the values are computed each time the command is traversed. These commands do not affect the current cursor position.

`COMPUTE` and `IF` are also options on the `INITIAL` and `FINAL` clauses of the MENU, RECORD, TABLE, CALL and FIELD commands. These options use expressions and conditions in exactly the same manner.

COMPUTE

`COMPUTE variable =` expression

The variable specified on the left side of the equal sign must be defined in the record or table accessed by the current screen or in the DECLARE clause of the `FORM` command.

- **Example:** To compute the average of three variables:

```
COMPUTE AVERAGE = (VARX + VARY + VARZ) / 3
```

IF

`IF` (condition) `variable =` expression

The variable specified on the left side of the equal sign must be defined in the record or table accessed by the current screen or in the DECLARE clause of the `FORM` command. The variable is only calculated if the condition is true.

Example: To give a $1,000 raise to all technical writers (position = 10):

```
IF (position eq 10) salary = salary + 1000
```

Conditions

An `IF` condition may include the following operators:

`EQ or = or IS`	Equal to
`NE or ><`	Not equal to
`LT or <`	Less than
`LE or <=`	Less than or equal to
`GT or >`	Greater than
`GE or >=`	Greater than or equal to
`LIKE`	Matches a specified character pattern
`EQ NULL`	Is missing values
`NE NULL`	Is not missing values

The logical operator `NOT` can be used to test for the opposite of any condition. Parentheses ( ) can be used in the `IF` clause to explicitly denote the order of evaluation. The default order of precedence is to evaluate expressions first, then to use the relational operators to determine whether the test is true, then to test compound conditions.

EQ, NE, LT, LE, GT, GE

These operators test the relationship between two values. If the specified condition is true, the field is computed. `NE` is provided as a convenient shorthand; it is identical to `NOT EQ`.

LIKE Pattern Matching

A pattern is a partial string where symbols are used to indicate how that position is to be treated. (see Pattern Matching for information on `LIKE` pattern matching.)

compound conditions

An `IF` condition may consist of a number of conditions connected by the logical operators `AND,` `OR,` and `XOR`.
`AND` means both conditions must be true;
`OR` means either condition must be true;
`XOR` means either condition must be true but not both.

Expressions

Expressions are a combination of variables and operators which produce a new value.

There are numeric expressions and string expressions. Adding two numeric variables produces the sum. Concatenating two string variables produces a longer string. Numeric and string variable types cannot be mixed. There are functions to convert numbers to strings and vice versa.

Numeric expressions consist of numeric variables, constants, all of the normal arithmetic operators (+, -, /, *, **), numeric functions, and parentheses. A numeric constant starts with a number and may have a decimal point.

String expressions may use quoted strings, variables which are strings, the "+" character to join strings, string functions, and parentheses to denote the order of operation. Enclose strings in single quotes.

A function is a keyword followed by one or more arguments enclosed in parentheses. Arguments may be variable names, constants or expressions. The function operates on the arguments and returns an appropriate value. For example, `MOD` returns the remainder from an integer division. The following order of precedence applies to expressions:

> Expressions in parentheses
>
> Functions
>
> Exponentiation (**)
>
> Multiplication (*) and division (/)
>
> Addition (+) and subtraction (-)
>
> Expressions of equal precedence are evaluated from left to right.

For example:

```
2*5-4                    (6)
2*(5-4)                  (2)
MOD(5,2)                 (remainder; 1)
SALARY*52/12             (monthly salary from  weekly (approx))
'ABC' + 'DEF'            ('ABCDEF')
SBST('ABCDEF',4,3)       ('DEF')
TIMEC(NOW(0),'HH:MM:SS') (current time in display format)
```

Functions

Functions can be used in expressions in `IF` and `COMPUTE`. A function is a keyword followed by one or more arguments enclosed in parentheses. Arguments may be either variable names, constants or expressions. The function operates on the arguments and returns an appropriate numeric (n) or string (s) value . The available functions are:

`ABS`

```
n = ABS(n)
```

Absolute value of n.

`ACOS`

```
n = ACOS(n)  or ARCOS(n)
```

Arc cosine of n.

`AINT`

```
n = AINT(n)
```

Same as `TRUNC`. Returns the integer portion of n after truncating any fractional part.

`ALOG`

```
n = ALOG(n)
```

Natural logarithm of n.

`ALOG10`

```
n = ALOG10(n)
```

Base 10 logarithm of n.

`AMOD`

```
n = AMOD(n,m)
```

Remainder of integer division of n by m.

`ASIN`

`n = ASIN(n) or ARSIN(n)`

Arc sine of n.

`ATAN`

`n = ATAN(n)`

Arc tangent of n.

`CDATE`

`n = CDATE (date_string,'format')`

Convert a date string to a date integer using the date format specified in quotes.

`CEILING`

`n = CEILING(n)`

The smallest integer greater than or equal to n.

`CNT`

`n = CNT(n1,n2,...)`

The number of arguments that are not missing or undefined.

`COUNT`

`n = COUNT(n)`

The number of record type (n) in the current case.

`COS`

`n = COS(n)`

Trigonometric cosine of n.

`CTIME`

`n = CTIME (time_string,'format')`

Converts a time string to a time integer using the time format specified in quotes.

`DATE`

`s = DATE(0)`

Current date as a string in the format `MM/DD/YY`.

`DATEC`

`s = DATEC(n,'format')`

Converts a date integer to a date string using the date format specified in quotes.

`DATET`

`s = DATET(n1,n2)`

The current date and time are available in a string in the format: '`WWW,MMMDD,YYYY,HH.MMAP`' where `WWW` is the day of the week, `MMM` is the first three letters of the month name, `DD` is the date, `YYYY` is the year `HH.MM` is the wall clock time and `AP` is AM or PM. Select a portion of the string by specifying the beginning and ending positions as n1 to n2. For example to return the `HH.MM` (the time in hours and minutes as a string), specify `DATET(16,20)`

`EXISTS`

`n = EXISTS(expr)`

Returns 0 if expression contains missing values; 1 if expression contains a non-missing value.

`EXP`

`n = EXP(n)`

Exponentiation (base e) of n.

`FIX`

`n = FIX(n)`

Truncation of n.

`FLOOR`

`n = FLOOR(n)`

The greatest integer less than or equal to n.

`FORMAT`

`s = FORMAT(n,w,d)`

Converts a number (n) to a string. Specify the length of string (w) to right justify the number. If w is not specified, the number is left justified. Specify any decimal places (d) required.

`GROUP`

`s = GROUP(0)`

Returns group name of current user.

`INT`

`n = INT(varname)`

Integer value of a categorical, date or time variable.

`JULC`

`s = JULC(n)`

Converts a date integer to a string in the format '`MMM DD, YYYY`'.

`JULN`

`n = JULN(m, d, y)`

Converts separate numeric month, day and year to a date integer.

`LEN`

`n = LEN(s)`

Length of string s.

`LOG10`

`n = LOG10(n) or LG10(n)`

Base 10 logarithm of n.

`LOG`

`n = LOG(n) or LN(n)`

Natural logarithm of n.

`LOWER`

`s = LOWER(s)`

Convert a string to lower case.

`MAX`

`n = MAX(n1,n2,...)`

The largest value that is not missing or undefined.

`MIN`

`n = MIN(n1,n2,...)`

The smallest value that is not missing or undefined.

`MOD`

`n = MOD(n,m)`

Remainder of integer division of n by m.

`NOW`

`n = NOW(0)`

Current time as a time integer.

`NUMBR`

`n = NUMBR(s)`

Converts a string to a number.

`PATTERN`

`n = PATTERN(s1,s2)`

Returns 1 if pattern s2 is found in string s1 and 0 if not found. A percent sign is the wildcard character. With no wildcard, an exact match is done. For example, ('ABC','A') does not match, ('ABC','A%') does match.

`RAND`

`n = RAND(0) or RANF`

Random uniform number between 0 and 1.

`SBST`

`s = SBST(s,n1,n2)`

Extracts a substring of string s, starting from position n1 for a length of n2.

`SIGN`

`n = SIGN(n1,n2)`

The absolute value of n1 with the sign (positive or negative) of n2.

`SQRT`

`n = SQRT(n)`

Square root of n.

`SRST`

`n = SRST(s1,s2)`

Returns the starting column in s1 of substring s2 if s2 is found in s1 and 0 if not found. The value is positive if s2 is surrounded by special characters or blanks; negative if s2 is surrounded by letters or numbers.

`STR`

`s = STR(varname)`

String value of a categorical, date or time variable.

`SYSTEM`

`n = SYSTEM(n),`

Returns various system status information. Set **n** to one of the following codes. The value returned indicates the following meaning:

n	Meaning
101	cir is new (0 no, 1 yes)
102	cir is old (0 no, 1 yes)
103	cir is changed (0 no, 1 yes)
104	cir is in process loop (0 no, 1 yes)
105	cir is marked for deletion (0 no, 1 yes)
106	record is new (0 no, 1 yes)
107	record is old (0 no, 1 yes)
108	record is changed (0 no, 1 yes)
109	record is in process loop (0 no, 1 yes)
110	record marked for deletion (0 no, 1 yes)
111	number of cases in database
112	number of record type in case
113	number of record type in database
114	current database update level

121	current mode (0 find, 1 search)
122	case lock
123	record lock
124	verify mode (0 no, 1 yes)
125	table row is new (0 no, 1 yes)
126	table row is old (0 no, 1 yes)
127	table row has changed (0 no, 1 yes)
128	table row in process loop (0 no, 1 yes)
129	t able row marked for delete (0 no, 1 yes)
130	table row is lock_type (1-6)
131	direction at field (-1 back,+1 forward)
132	direction at CIR (-1 back,+1 forward)
133	direction at record (-1 back,+1 forward)

`TAN`

```
n = TAN(n)
```

Trigonometric tangent of n.

`TANH`

```
n = TANH(n)
```

Hyperbolic tangent of n.

`TIME`

```
s = TIME(0)
```

Current time as a character string in the format `HH:MM:SS`.

`TIMEC`

```
s = TIMEC(n,'format')
```

Converts a time integer to a time string using the time format specified in quotes.

`TODAY`

```
n = TODAY(0)
```

Current date as a date integer.

`TRIM`

```
s = TRIM(s) or TRIMR(s)
```

Trims trailing blanks from string s.

`TRIML`

`s = TRIML(s)`

Trims leading blanks of string s.

`TRIMLR`

`s = TRIMLR(s) or TRIMRL(s)`

Trims leading and trailing blanks of string s.

`TRUNC`

`n = TRUNC(n)`

Same as `AINT`. Returns the integer portion of n after truncating any fractional part.

`UPPER`

`s = UPPER(s)`

Converts string s to upper case.

`USER`

`s = USER(0)`

Returns current user name.

Visual Control

There are a number of commands to enhance the appearance of a screen. The DRAW command draws lines and boxes.

After drawing the box, the cursor is positioned just inside the top left corner of the box.

The TEXT command can be used to place text any where on the screen.

The AT command positions the cursor.

The default screen position for display data on the screen is one line down and in the same column as the previous data. The `AT` clause can be used on any command that relates to displaying information to position the information according to the coordinates specified.

Positioning can be absolute or the positioning can be relative to the current position. Numbers such as `AT 10,15` positions the information at a specific point on the screen (e.g. row 10, column 15). A "+" or a "-" such as (+3,+0) indicates relative movement from the current position (e.g. 3 rows down, in the same column.

AT

`AT [ r ] [, c ]`

`AT` positions the cursor on the screen. This determines where the next element is displayed. The default position is one row down at the same column.

`AT` specifies row (r) and column (c) for the position. Row and column positions can be specified in either absolute or relative terms. Absolute positions are specified in the form "n" where n is the absolute row or column. `AT 1,1` positions the cursor to the upper left corner of the window.

Relative positions can be specified in the form "+n" or "-n" to indicate row and column movement from the previous cursor position.

An asterisk (*) indicates the maximum row or column.

```
AT 5 , 1
```

Positions at row 5, column 1.

```
AT , 12
```

Positions at column 12 of the current line.

```
AT -1,1
```

Positions at the beginning of the previous line.

The `AT` command positions the cursor independently of any screen display command. This may be useful in setting a default display position in the middle of a list of `FIELD` commands. `AT` can always be specified as a clause on an individual command.

DRAW

```
DRAW ['c']
   FOR [r][,c]
   FROM [r][,c]
   TO [r][,c]
   CENTER [HORIZONTAL | VERTICAL]
   END symbol
   FILL ['fill_character']
   NOLINES
   START symbol
```

General Clauses:

```
AT [r][,c]
VIDEO video_options
```

`DRAW` draws lines (both vertical and horizontal) and boxes. After drawing a horizontal line, the cursor is positioned at the row and column position of the first character of the line. After drawing a vertical line, the cursor is positioned one character to the left of the top character of the line. After drawing a box, the cursor is positioned at the top left inner corner of the box.

`DRAW 'c'`

> Specifies a character in quotes immediately following the `DRAW` command to alter the character used to draw the line or box.
>
> For example, to draw a box using asterisks, specify:
> `DRAW '*' FOR 2,28`

`FOR r,c`

> Specifies a relative destination point for drawing. The line or box begins at the current position and continues for as many characters as are specified for row and column. The current position counts as one row and/or column, so the row and column specifications are one less than the total dimension desired.
>
> If only the row coordinate is specified, a vertical line is drawn. If only the column coordinate is specified, a horizontal line is drawn. A `DRAW` with no `FOR` specified is one row deep and one column wide ,i.e. one character.
>
> For example: The first command draws a box three lines deep and 29 columns wide; the second draws a vertical line for 4 rows down; the third draws a horizontal line for 20 columns across.

```
DRAW FOR 2,28
DRAW FOR 3
DRAW FOR ,19
```

`FROM r,c`

Specifies the absolute starting point for drawing the line or box. If `FROM` is not specified, the `DRAW` commences at the current cursor position. When drawing a box, `FROM` specifies the upper left coordinate. When drawing a line, `FROM` specifies the left column or top row. For example, to draw a box that begins on line 12, column 2 and continues to line 20, column 30, specify:

```
DRAW FROM 12,2 TO 20,30
```

`TO r,c`

Specifies an absolute destination point for drawing the line or box. When a box is drawn, `TO` specifies the bottom right coordinate. When a line is drawn, `TO` specifies the right column or bottom row. Specify an asterisk to denote maximum row or maximum column as defined by `PAGESIZE`.

For example, the first command draws a box starting at the current position and ending in the lower right corner; the second draws a box around the current page; the third draws a box from line 12, column 2 to line 20, column 25.

```
DRAW TO * , *
DRAW FROM 1 , 1 TO * , *
DRAW FROM 12,2 TO 20,25
```

`CENTER [HORIZONTAL| VERTICAL]`

Specifies that the box is centred in the screen. `HORIZONTAL` causes the box to be centred only horizontally; `VERTICAL` causes the box to be centred only vertically. If neither option is specified the box is centred in the middle of the screen.

`END | START symbol`

Specifies symbols used for the junction of vertical and horizontal lines when drawing a box. The symbols at the junction of the vertical line and the horizontal line are different from the standard horizontal and vertical line characters and differ depending on which corner is being drawn. The available symbols are:

TLCORNER	Top Left Corner
TRCORNER	Top Right Corner
BLCORNER	Bottom Left Corner
BRCORNER	Bottom Right Corner
TTEE	Top TEE
BTEE	Bottom TEE
LTEE	Left TEE
RTEE	Right TEE
INTERSECTION	Intersection
VLINE	Vertical Bar
HLINE	Horizontal Bar

`FILL`

Specifies the character to use to fill a box. FILLing a box means that any previously displayed information in that portion of the screen is overwritten. A `DRAW` without the `FILL` clause does not affect data in the middle of the box. `FILL` without a fill character fills the box with blanks. Specify `VIDEO INVERSE` on the `DRAW,` to `FILL` with `INVERSE FILL` characters. The fill character can be any character from the keyboard.

`NOLINES`

Specifies that blanks are used as the line drawing character. When `NOLINES` is used with `VIDEO INVERSE`, the line or box is drawn with inverse blanks.

TEXT

```
TEXT 'text_string'
     CENTER
     LINESET
     UNDERLINE ['underline_character']
```

General Clauses:

```
AT [r][,c]
VIDEO video_options
```

`TEXT` specifies a text string which is displayed. Each `TEXT` command displays the text at the current cursor position and increments the position by the standard single line. The text string to be displayed is the first and only required parameter. This may be followed by any of the specific or general clauses.

For example:

```
TEXT 'Welcome to the Administration System' at 3,30
TEXT 'Please contact ext. 123 for help' at +2 ,+5
```

`CENTER`

Centres the specified string in a line.

`LINESET`

The `TEXT` command without `LINESET` displays any character that can be entered on the keyboard. `LINESET` specifies that the symbols associated with lines and boxes are used. With `LINESET`, the specified text string is a set of codes with the following meanings.

CHARACTER	MEANING
A	Top left corner
B	Top right corner
C	Bottom left corner
D	Bottom right corner
E	Top Tee
F	Bottom Tee
G	Left Tee
H	Right Tee
I	Intersection
J	Vertical line, thin
K	Vertical line, thick
M	Horizontal line, thin
N	Horizontal line, thick

Codes must be uppercase. Any character other than the code characters is displayed on the screen. For example, the following creates a small rectangular box:

```
TEXT 'AMMMMMMMMMMMMB' LINESET
TEXT 'J            J' LINESET
TEXT 'CMMMMMMMMMMMMD' LINESET
```

`UNDERLINE`

Specifies that the text is underlined with hyphens (-) on a new line. Specify the underline character to use. For example, to underline the text with an equal sign instead of a hyphen:

```
TEXT  'MAIN MENU'  -
      CENTER AT 2 VIDEO BOLD UNDERLINE '='
```

GENERATE

The `GENERATE` command in record or table screens uses the record or table schema to give the equivalent of default `FIELD` commands for every field. `GENERATE` in a Menu screen produces the equivalent of a default `CALL` to every previously defined Record or Table screen. Commands are not physically created and thus cannot be edited.

The `EXCLUDE` and `INCLUDE` clauses allow fields to be nominated which are affected or not by the `GENERATE` and allow specific `FIELD` commands to be combined with a `GENERATE` command. When a `FIELD` command is specified for a variable, the variable is `EXCLUDED` from the `GENERATE` otherwise it appears on the screen twice. Key fields must be the first fields in a screen definition.

Syntax

The syntax for the `GENERATE` command is:

Clauses:

```
GENERATE EXCLUDE | INCLUDE
         COMMON
         DATAVARS
         SCREENS screen, ...
         SORTIDS
         VARS    variable, ...
```

General Clauses:

```
ACTIVITIES permissions
AT [r][,c]
CLEAR 'c'
DATA HILITE | VIDEO options
LABELS  n
PAD [n][,m]
PROMPT HILITE | VIDEO options
```

The general clauses on the `GENERATE` command apply to all of the fields generated by the command. The `AT` clause refers to the position of the first field.

`EXCLUDE`

> Omits classes of variables or specific variables from the `GENERATE`. When `EXCLUDE` is specified, any variable (screen) not `EXCLUDED` is `INCLUDED` automatically.

`INCLUDE`

> Includes classes of variables or specific variables in the `GENERATE`. When `INCLUDE` is specified, any variables (or screens) not `INCLUDE`d are `EXCLUDED`.

`COMMON`

> Specifies all common variables.

`DATAVARS`

> Specifies all data fields.

`SORTIDS`

Specifies all key fields (sort ids).

`SCREENS`

Specifies screen names to be `INCLUDE`d or `EXCLUDE`d in menu screens.

`VARS`

Specifies a list of named variables.

For example, to specify particular options for three fields and then bring in all the remaining fields in the `EMPLOYEE` record:

```
RECORD EMPLOYEE / COMPANY.EMPLOYEE
   TEXT  'This is the Demographic Record' UNDERLINE CENTER AT +2
   FIELD ID        PROMPT  'Employee ID:' DATA VIDEO INVERSE
   FIELD NAME PROMPT  'Name of Employee' REQUIRED
   FIELD SSN    PICTURE 'ddd-dd-dddd'
   GENERATE  EXCLUDE VARS ID  NAME  SSN
ENDRECORD
```

Help

Help screens display information. This can be associated with a particular field through the HELP clause on the `FIELD` command or may be called up through a CALL command.

Within the Help Screen Definition, specify visual appearance commands such as `TEXT` and `DRAW` and PAGE commands as necessary.

Syntax

```
HELP screen_name
     KEEP
     PRELOAD
ENDHELP
```

General Clauses

```
AT [r][,c]
BOTTOM n
PAGESIZE [r][,c]
```

`HELP` defines a text screen which cannot contain any commands other than `TEXT` and `DRAW` and `PAGE` and `ENDPAGE`.

`KEEP`

> Specifies that the screen remains in the cache buffer if caching has been enabled.

`PRELOAD`

> Specifies that the screen is put in the cache buffer prior to the execution of the form.

Example: The first `DRAW` command draws a box beginning at the screen origin 1,1 through the extreme point 20,79. The second `DRAW` command draws a smaller box within the first. The `TEXT` command displays the specified text.

```
.
CALL HELPSCRN PROMPT 'Select if further HELP required'
.
HELP HELPSCRN
  DRAW TO 20,79
  DRAW FROM 5,30 FOR 2,30
  TEXT 'THIS IS A HELP SCREEN' VIDEO BOLD
  TEXT '  ....  help text ....'   at 10,20
  TEXT '  ....  more help text ...'
ENDHELP
```

Running SirForms

When SirForms is run, a number of parameters can be specified. At a minimum, the name of the input file containing the form must be specified using either `CMP=` for source or `FRM=` for compiled. If this is omitted, a `CMP` source input is prompted for.

Separate parameters with slashes (/) or spaces.

Execution parameters

The following parameters can be specified:

```
COMPILE | CMP = filename with source text
EDIT = YES | NO
ERRORFILE = filename for any compilation errors
EXPERT
FORM | FRM = filename of compiled form
GRPNAM = groupname
GRPPAS = password
LISTING = filename
NOEXECUTE
NUMSCR = n
PREFIX = database path
PRINT = filename
PS = n
PW = database password
RS = read password
TMPLTSCR = filename
USRNAM = username
USRPAS = password
WD = n
WS = write password
```

`CENY=yyyy`

Sets the year used for calculation of the default century when two digit years are input. When a two digit year is input, the system compares the input year with this parameter. If the parameter is less than the input, the century is set to the specified century. If the parameter is greater than or equal to the input, the century is set to the specified century plus 1.

The default is 1920.

To illustrate the calculation, assume the default setting of 1920. Then any input year in the range 00 - 19 is assigned a century of 20nn; any input year in the range 20 - 99 is assigned a century of 19nn.

All variables defined as dates are held internally as a number of days since the start of the Julian calendar on Oct 15 1582.

`COMPILE | CMP`

Specifies an input file to compile. `CMP` is the abbreviation.

`EDIT`

Specifies whether fields can be edited character by character (`=YES`) or whether a complete field must be re-entered (`=NO`). `YES` is the default

`ERRORFILE`

Specifies the name of the file to contain any errors found in compilation. If an `OUT=` is specified, errors are written to that along with other messages. If neither parameter is specified and compilation errors occur, messages are written to `SirForms.err`.

`EXPERT`

Disables the listing of expanded error messages and warnings during form compilation.

`FORM | FRM`

Specifies the name of the file containing the compiled version of the form. If this parameter is specified together with the `COMPILE` parameter, the form is saved in compiled format on the file specified.

If the `FRM` parameter is used without the `CMP` parameter, the file specified is accessed and executed. If neither `FRM` nor `CMP` is specified, SirForms prompts for the file name of the source file. `FRM` is the abbreviation.

`GRPNAM`

Specifies the `GROUP NAME.` If the form has group names specified and `GRPNAM` is not supplied, it is prompted for.

`GRPPAS`

Specifies the password assigned to the group name specified with the `GRPNAM` parameter.

`LISTING | LIST | OUT`

Specifies the file where the output listing is written. If `LISTING =` is not specified, error messages are written to the error file or to the screen depending on whether these are compile or run time errors. `LIST` and `OUT` are alternates.

`NOEXECUTE`

Specifies that the form is not executed after compilation.

`NUMSCR`

Specifies the maximum number of screens to be held in the screen cache. The default value for "n" is 4. Use the `PRELOAD` and `KEEP` clauses on the screen commands (`MENU, RECORD, TABLE,` and `HELP`) to control screen caching.

`PRINT`

Specifies a file for output from the user `PRINT` command. If the user then issues a `PRINT` command, all pages in the current screen are written to this file as text.

`PREFIX`

Specifies the database prefix which is the full path pointing to the location of the database.

`PW`

Specifies the database password.

`RS`

Specifies the `READ` security level password.

`TMPLTSCR`

Specifies the template scratch file. Specify a scratch file to avoid conflicts with other SIR/XS modules which use a template scratch file.

`USRNAM`

Specifies a user name defined within the group specified with the `GRPNAM` parameter.

`USRPAS`

Specifies the user password assigned to the user name specified with the `USRNAM` parameter.

`WS`

Specifies the `WRITE` security level password.

Examples of running SirForms

In these examples, the command `SirForms` and the parameter separator is a blank. The following are typical examples of executing SirForms.

```
SirForms CMP=MYFORM.cmp LISTING=OUTPUT.LST
```

This compiles a form from `MYFORM.cmp`, produces a listing on `OUTPUT.LST` and executes the compiled form if there are no errors.

```
SirForms CMP=MYFORM.cmp FRM=MYFORM.frm NOEXECUTE
```

This compiles a form, stores a compiled version of the form on `MYFORM.frm` and does not execute the form.

```
SirFormsFRM=MYFORM.frm
```

This executes a previously compiled and saved form from `MYFORM.frm`.

Key Mapping

In SirForms, various functions are performed with special keys. The physical keys that accomplish any given task can be altered by the user through keymapping.

When you start SirForms, it checks to see if a keymap file is present. If the appropriate file is not found, the system immediately prompts you to supply a minimum set of keys. These are `Enter`, `Backspace`, `Escape` and the four arrow keys (up, down, left and right). At each prompt, simply press the appropriate key on your keyboard. It also asks for one special key called `Key Help`. Press the appropriate key and type in the name of this physical key, for example "Shift-F1". After this initial mapping, pressing the `keyhelp` key displays a list of keys and allows you to alter the mappings.

The keymap file supplied with the software uses "Shift-F1" as the `keyhelp` key and this can be modified to reflect any new key mapping rather than starting from scratch.

Meta Key

Most systems use a combination of certain special keys such as `Ctrl`, `Shift` and `Alt` which, when pressed at the same time as standard keys, extend the number of keys available. Some systems use the concept of a `MetaKey` which performs a similar function but is pressed before the standard key. The most common example of this is the `Gold` key on DEC systems. If you use this style of key, define it as part of the initial keymapping. If you do not use this style of key (most users) do not define a meta key.

Changing Key Mapping

Once you have a keymap file, press `Key Help` to display a list of keymappings. Use the menus on the displayed list to alter existing mappings or add new ones.

To define a new mapping, position to the function with the arrow keys. Press **M** to associate a new key with that function. Then, responding to the prompts, press the key you want to allocate to the function and then enter the name of the physical key, for example "Ctrl-M". You can only allocate special keys to functions, you cannot map standard character keys.

One physical key can only be associated with one function but a function can have more than one physical key associated with it. To remove an existing mapping, position to the function with the arrow keys and press **U**. This removes the first physical key mapped to the function. Repeat the process if there are further keys to be de-allocated.

If you exit after changes without first saving the keymap file, you are prompted to save or abandon your changes.

If you wish to have a special keymapping, copy the appropriate keymap file from the home directory to your working directory and then modify it as above.

Concurrency

Overview

This is intended for users who are using a database concurrently with other users. The concurrent environment is controlled by a program called `MASTER`. To use SirForms with concurrency, specify the name of the `MASTER` process on the `MST =` parameter at start-up. In concurrent mode, multiple users can simultaneously enter data and read data from the same database.

A form that has been designed for single user running, runs without alteration in concurrent mode. The defaults provide a reasonable concurrent user environment. If you specify particular features such as locking, which only have meaning in a multiple user environment, these specifications can be left in the form for single user operation and have no effect. The same form definition can run interchangeably in both environments.

When using a form, it appears the same whether or not you are using concurrency. While a database is under `MASTER`'s control, that database can only be updated through `MASTER`. It cannot be updated through the non-concurrent products. The non-concurrent products can read the database, but they cannot update it.

There may be a number of `MASTER` programs on a system, and the one to use is specified through the `MST=` parameter on the execution statement.

When a `MASTER` starts, it does not use any databases, and once the last user has stopped using a database, `MASTER` releases it. `MASTER` could be on a system for hours or days doing no work and taking virtually no resources, waiting for a request from a user program to attach to a particular database.

When using a concurrent product through `MASTER`, information is at the latest possible level. If you use a non-concurrent product in read-only mode while the database is still controlled by `MASTER`, information is as per the last "Permanent Update" before the process connected to the database.

Prefix

`MASTER` is a separate process from an `SirForms` session and `MASTER` may use a different default directory. The database to access may not be in the `MASTER` default directory. It is advisable to specify a full prefix in order that `MASTER` is able to find the database you wish to access. Specify the directory either with the PREFIX parameter on the execution statement or with the `PREFIX` sub-clause on the DATABASE clause of the `FORM` command. It is safest if all forms which accesses a database have the `PREFIX` specified to avoid any problems when using the form from different directories. If a form uses multiple databases in multiple directories, then the prefix information must be specified in the form, not on the execution statement parameters. For example:

```
SirForms/ FRM=MYFORM.frm / MST='sirnt.sirxs.com.au' /
PREFIX='C:\sirxs\'
```

Locking

When multiple users are updating the same database at the same time, `MASTER` uses locks when updating a record to prevent other users from simultaneously updating the same record. Locks apply to individual instances of a record. They do not apply to the database as a whole or to all records of one type.

Locks can be specified on the CALL or ROOT commands in the form definition. During a session, the user can change the lock types specified in the form, provided they have LOCKCHANGE permission. Current lock types are displayed in the status area. The lock types are:

Lock Type	Command	Requests	Allows Other
CONCURRENT READ	CR	Read access	Readers Writers
CONCURRENT WRITE	CW	Read and write access	Readers Writers
PROTECTED READ	PR	Read access	Readers No writers
PROTECTED WRITE	PW	Read and write	Readers No writers
EXCLUSIVE	EX	Read and write access	No

`Concurrent Read` allows read-only access to the record. Other users can still access the record for read and write.

`Concurrent Write` allows both read and write access, while also allowing other users both read and write access. (It is strongly recommended NOT to use this lock type as updates could be lost.

`Protected Read` allows read-only access to the record. Other users can read the data, but no writers are allowed.

`Protected Write` allows read and write access. Other users can read the data but no writers are allowed.

`Exclusive` establishes read and write access to the record. No other users can access the record.

Changing Locks

Operators with `LOCKCHANGE` permission can use the `LOCK` and `UNLOCK` commands to change the lock types during a session. The user enters the command `LOCK` and a lock type. Lock types entered with the `LOCK` command remain in effect while the screen is being used: they do not affect the form definition. If a lock is changed, any other locks on the requested record are scanned for compatibility. Specify `NOLOCKCHANGE` on the `ACTIVITIES` clause on the `FORM` to revoke permission to make lock changes.

The following table shows the allowable changes:

`Current Locks` refer to the locks held on a case or a record by other users at the time the lock change is requested. New Lock Request refers to the lock type entered with the `LOCK` command. For example, if a user requests a `Concurrent Write` (`CW`) lock on a record, the request is granted if there are no other locks or the other locks currently granted are `Concurrent Read` (`CR`) or `Concurrent Write` (`CW`). The request is denied if there is a `Protected Read` (`PR`), `Protected Write` (`PW`), or an `Exclusive` (`EX`) lock established for the record. If a lock request cannot be granted, a message is displayed.

		Current Locks				
		CR	CW	PR	PW	EX
New Lock Request	CR	YES	YES	YES	YES	NO
	CW	YES	YES	NO	NO	NO
	PR	YES	NO	YES	NO	NO
	PW	YES	NO	NO	NO	NO
	EX	NO	NO	NO	NO	NO

LOCK

```
LOCK lock_type or
LOCK [CIR lock_type] [RECORD lock_type] [ROW lock_type]
```

Specify locks in a form definition with the `LOCK` clause. There are four places to do this:

- the ROOT clause of the `FORM` command;
- the CALL command;
- the LOOKUP command;
- the LOOKUP clause of the `FIELD` command.

Specification of lock types is optional. The default lock type for the root screen is `Exclusive (EX)`. For called screens, the default lock type is the same as that of the calling screen. The default for the `LOOKUP` command and clause is `Concurrent Read` (`CR`).

Added to the `ROOT` clause, the `LOCK` option sets the default concurrency mode for the entire form. All screens use the default lock type. For example, to set a lock type of `Concurrent Read` (`CR`) for all the data accessed with a form, specify:

```
form COMPANY ..... root MAINMENU lock CR
```

Specifying `LOCK` and a lock type sets the lock types for both the CIR and the associated record. Lock types can be specified separately for the CIR, the record or a table row. For a separate lock type for the CIR and for records, specify:

```
form COMPANY ..... root MAINMENU lock CIR PR RECORD EX
```

On the `CALL` command, the `LOCK` clause specifies the lock type to apply to the records on the called screen. The lock type specified on the `CALL` command overrides the lock established with the `ROOT` clause. The lock set on the `CALL` applies to all instances of that record type displayed on the screen. For example, to set a lock type of `Protected Write` (`PW`) for a called screen named `OCCUP`, specify:

```
CALL OCCUP .... lock PW
```

The `LOCK` clause on the `LOOKUP` command (or the `LOCK` option to the `LOOKUP` clause of the `FIELD` command) sets the lock type for the records accessed by the `LOOKUP`. The lock type specified on the `LOOKUP` command or clause overrides the lock established with the `ROOT` clause. There is no reason to have any lock type other than "CR" on a `LOOKUP`.

Establishing a Set of Lock Types

Selecting appropriate lock types for an application depends on the type of work being done.

`Read Only`

> If users are only reading data, specify `ACTIVITIES NOWRITE NOLOCKCHANGE` and use the `Protected Read` (`PR`) lock type. This allows multiple users to read a record and prevents any other user from simultaneously modifying or updating a record which is being looked at.
>
> Consider using the non-concurrent product for read only access, specifying the `READ` clause on the `DATABASE` specification on the `FORM` command. The advantage is that this avoids the overhead of inter-process communication involved in using `MASTER`.

`Read & Write`

> If numerous people simultaneously read records, but data input is limited to select individuals, define separate form definitions for readers and writers. Use the `Concurrent Read` (`CR`) lock type for the readers and the `Protected Write` (`PW`) lock type for the writers. For maximum control of data input and updating, use the `Exclusive` (`EX`) lock type. This lock type allows only one writer and no readers.

Locking can be at the case-level or at the record-level. When a case-level lock is required, specify the keyword `LOCK` and a lock type to lock both the CIR and the records for a case. Specify separate lock types for the CIR and records for general access to the case, but a more restricted access to the records.

If you lock at the case level, only one user at a time can access any of the records for a case, regardless of the record types involved.

Master Messages

The following messages may occur during a concurrent session:

Message	Explanation
98	Waiting for locked case. Another user has locked the CIR and the record. If the likelihood is good that the other user will release the case, you may decide to wait. If not, issue a break. Issuing a break causes Message 99 to be displayed.
99	Next case is locked, enter `ABORT, LOCK` or `WAIT`. This message is displayed after a break is issued in response to Message 98. You can `ABORT` the process, change your lock request, or continue waiting.
100	Waiting for locked record. Another user has locked the record. If the likelihood is good that the other user will release the record soon, you may decide to wait. If not, issue a break. Issuing a break causes Message 101 to be displayed.
101	Next record is locked, enter `ABORT, LOCK` or `WAIT`. This message is displayed after a break is issued in response to Message 100. You can `ABORT` the process, change your lock request, or continue waiting.
123	No permission to change `LOCK` status for record or CIR. `LOCKCHANGE` permission has been revoked. See whoever created your form.
150	Illegal lock type specified. Most probably, a misspelled lock type. Check the spelling of the lock type and reenter the command with the proper spelling.

Error Messages

Number	Default Text
1	Illegal COMMAND specified.
2	No such COMMAND.
3	Illegal KEYWORD " mode is NONE.
4	Illegal command syntax.
5	Record does not exist on the database.
6	Leading insert overflow.
7	Edit insert overflow.
8	Field command is illegal in MODIFY mode.
9	End of screen.
10	Trailing insert overflow.
11	Date syntax error.
12	Integer syntax error.
13	Floating point syntax error.
14	Input is too short.
15	Input is too long.
16	Input does not match picture.
17	Leading insert overflow.
18	Edit insert overflow.
19	Biasing error.
20	Field is required.
21	Scaling error
22	Trailing insert overflow
23	Time syntax error.
24	Floating insert overflow.
25	END of screen.
26	End of search.
27	End of find.
28	Record already exists.
29	Record has been DELETED.
30	Record has been STORED.
31	SEARCH record has been found.
32	FIND record has been found.
33	BEGINNING of screen.
34	BEGINNING of page.
35	END of page.
36	Command is illegal during initialisation.
37	End of CASES +" no more RECORDS.
38	End of RECORDS.
39	End of CASES.
40	Command illegal in READ only mode.
41	Command illegal in READ/WRITE mode.
42	Record is READ locked.
43	Record is WRITE locked.
44	Next record is READ locked.
45	Next record is WRITE locked.
46	Command is illegal during menu.
47	Value is not valid or out of range.
48	A legal missing value was stored.

49 Data entry is illegal in read mode.
50 Data entry is illegal with no record.
51 Command is illegal with no record.
52 Command is illegal in read mode.
53 Command name is ambiguous.
54 Range test violation " value too low.
55 Range test violation +"" value too hi.
56 Range test violation +" illegal value.
57 Boolean failure.
58 Beginning of form.
59 Lookup failure.
60 Lookup warning.
61 Check boolean failure.
62 Old records cannot be deleted by the current screen/user.
63 New records cannot be created by the current screen/user.
64 New cases cannot be created by the current screen/user.
65 A new RECORD has been accessed.
66 A new CIR has been accessed.
67 Cannot access new record in read mode.
68 Cannot access new cases in read mode.
69 Illegal print option.
70 No print file was specified.
71 Search mode illegal for current screen/user.
72 Find mode illegal for current screen/user.
73 Field cannot be modified.
74 Field cannot be corrected.
75 CIR cannot be deleted if it owns records.
76 CIR cannot be deleted from current screen.
77 User cannot select write mode.
78 Key level marked.
79 Command is legal only during key initialisation.
80 User break.
81 User does not have permission to read current CIR.
82 User does not have permission to read current record.
83 Case is READ locked.
84 Case is WRITE locked.
85 Next case is READ locked.
86 Next case is WRITE locked.
87 Mark not legal at current time.
88 User does not have write access to the database.
89 Old records cannot be read by the current screen/user.
90 Old cases cannot be read by the current screen/user.
91 Case will be deleted automatically.
92 No field in page to execute .. prompt at command area.
93 A new CIR and RECORD have been accessed.
94 A new record was entered and modified - enter OK to save.
95 A new CIR was entered and modified - enter OK to save.
96 .... Waiting for read locked case.
97 Case is read locked, enter ABORT, NEXT or WAIT.
98 .... Waiting for write locked case.
99 Case is write locked, enter ABORT, NEXT or WAIT.
100 .... Waiting for read locked record.
101 Record is read locked, enter ABORT, NEXT or WAIT.
102 .... Waiting for write locked record.
103 Record is write locked, enter ABORT, NEXT or WAIT.
104 Check boolean failure, enter OK to continue.
105 .... searching.

```
106 New record in empty case, enter OK to delete CIR also.
107 Deleting only record in case, enter OK to delete CIR also.
108 Case is not empty, enter OK to delete the entire case.
109 Accept check failure, enter OK to ignore.
110 Reject check failure, enter OK to ignore.
111 Command is illegal at a call field.
112 Integer value is too large to format as integer.
113 Field does not match specified pattern string.
114 Date value is out of range for a date conversion.
115 Time value is out of range for a time conversion.
116 The system undefined value was stored.
117 String was truncated on store into database.
118 An old record was modified - enter OK to save.
119 An old CIR was modified - enter OK to save.
120 Calls cannot be executed without any permissions.
121 Illegal response .. enter YES (OK) or [NO].
122 Illegal response .. enter NEXT, ABORT or [WAIT].
123 No permission to change LOCK status for record or CIR.
124 Illegal response .. enter ABORT, LOCK or [WAIT].
125 Verification error .. new value does not match old value.
126 No permission to change verify mode status.
127 Accept check failure.
128 Reject check failure.
129 Where command not legal in FIND mode.
130 Operator illegal.
131 OPERAND is illegal.
132 Mixed mode operation.
133 Illegal expression terminator.
134 Character is illegal.
135 Illegal NUMERIC syntax.
136 Illegal STRING syntax.
137 Illegal operation in an expression.
138 Illegal variable name.
139 Illegal variable value.
140 Illegal argument count.
141 Illegal argument type.
142 Illegal array element.
143 Illegal type of result for expression.
144 No permission to update old rows.
145 No permission to insert new rows.
146 No permission to delete old rows.
147 No permission to read old rows.
148 No permission to read column.
149 No permission to modify column.
150 Illegal lock type specified.
151 Value was too large to store in field.
152 Command is not legal at the current time.
```