UNIFACE DEVELOPMENT GUIDELINES

PART 1 - EXTERNALS

Author: A J Marston

Date Created: 11 June 1999

Date Changed: 02 January 2002

Version: 04.006.000

NOTICE

Table of Contents

Amendment History

01.000.000 to 03.004.000	April 1995 May 1999 Versions produced in a previous life.
04.000.000	June 1999 Total Rewrite.
04.001.000	18th July 1999 a) Added section 3.9 *Column Buttons*.
04.002.000	15th November 1999 a) Added new Dialog Type 4.11.2 *Select 2*. b) Amended *List 4* to receive hitlist from *Select 2*.
04.003.000	29th November 2000 a) Added new Dialog Types 4.3 *Control*.
04.004.000	22nd April 2001 a) Changed Copyright page. b) Added new Dialog Type 4.6 *Front End*.
04.005.000	26th November 2001 a) Updated *3.1.1 Fonts* for a screen resolution of 1024 x 768. b) Redefined the contents of *3.2.1 The Pull-Down Menu Bar*. c) Added a session panel layout in *3.2.2.1 Session Panel*.
04.006.000	2nd January 2002 a) Added Dialog Type *Multi 4c*. b) Added section *3.8.1.1 HTML Help*

1. INTRODUCTION

1.1 PURPOSE

This document describes a set of standards and guidelines that could be used for the development of computer systems using the UNIFACE 4th Generation Language deployed on a desktop PC running the Microsoft WINDOWS 9x operating system. These Development Guidelines have been split into the following parts:-

⇒ Part 1 deals with the External view of the software (ie: the "look and feel"). This will be of particular interest to the client as it shows which features are currently available, and indicates how they can be used to interact with the computer system.

⇒ Part 2 deals with the Internal workings of the software (ie: how it is actually constructed). This will be of interest to the development team as it gives examples on how the features described in Part 1 can be implemented, and would therefore be of little interest to the client.

⇒ Part 3 documents the Component Templates used in development.

⇒ Part 4 documents an XAMPLE Application which provides a working demonstration of these guidelines.

These guidelines have been put together using the experience gained while developing various systems using UNIFACE. Their purpose is as follows:-

- To identify the various facilities and options that are available to the system designer that can be used to achieve particular objectives.

- To examine the pros and cons of each method so that the optimum one can be chosen for each particular set of circumstances.

- To encourage the production of software that achieves the desired objectives as efficiently as possible and in a user-friendly manner.

- To encourage the production of software that has a consistent “look and feel” between its individual components so that the user does not have to go through a different learning curve with each component.

- To encourage the production of software that has a consistent “look and feel” with other software products that may be encountered by the user so that he can switch from one product to another and feel that he is in familiar, not alien territory. The Microsoft Windows Graphical User Interface (GUI) contains many “controls” that are used in a standard fashion within every software product developed for that platform. UNIFACE provides “widgets” which match a subset of these controls so that the same functionality can be provided.

- To encourage the construction of software using common techniques and, where possible, common components or routines so that they can be easily maintained and enhanced by any member of the development team, and not just the original author.

- To discourage the use of non-standard components, or features that are undocumented and/or unsupported, so that the software can be easily upgraded to subsequent releases of UNIFACE or other supporting software.

1.2 SOFTWARE VERSIONS

This document has been constructed with the following software versions in mind:-

⇒ UNIFACE version 7.2.06

⇒ Microsoft WINDOWS 95/98/ME/NT

There may be differences with the versions of Uniface that are available on other platforms, but they are outside the scope of this particular document. Certain widgets (controls) are not supported on earlier versions of Microsoft Windows, and screens designed for a GUI will not perform very well if deployed on a Character-based User Interface (CHUI).

New versions of UNIFACE may be released from time to time. Each release may contain new features, changes to existing features, or even the deletion of redundant or superseded features. Before any new release can be successfully deployed the contents of this document may need to be reviewed for accuracy.

1.3 DBMS PLATFORM

It is possible, indeed it is the norm, to develop applications within UNIFACE without making reference to the specifics of any particular file management/data storage system. All that is required is that the correct driver for the chosen file management system be incorporated into the final application. It is possible to take an application which was developed on one file system and deploy it using a totally different file system without making any changes to any of the code (other than a simple setting in the assignment file).

The only problem arises with the differences in functionality, capacity and speed of the different file management systems. It may not be possible to switch from one file system to another and achieve the same levels of performance or functionality. This is not a problem that can be solved by changing the design or by changing any code, it can only be solved by using a file management system that was designed to provide those levels of performance and functionality.

The requirements of any particular application should be identified and documented in advance so that they can be compared with the capabilities of the proposed file management system. Any incompatibilities or restrictions that arise from this comparison should also be documented as this may require a switch to an alternative file management system.

Of the many file systems which are currently available (relational databases, network databases, hierarchical databases, indexed files, flat files, etc.) not all of them support the following:-

- the STORE function
- the COMMIT function
- TWO-PHASE COMMITS (for logical updates across multiple databases)
- the ROLLBACK function
- stepped hitlists
- sorted hitlists
- record (row) level locking
- SQL interface
- stored procedures
- database triggers
- database views
- multiple cursors/paths

Other factors that may differ between the various file systems are:

- maximum size of table names and field names
- allowable characters in table/field names
- supported data types
- reserved words

It is possible in some circumstances to bypass standard UNIFACE behaviour and insert special code to take advantage of features that exist only within a particular file system. This practice should be avoided as it may cause problems at a later date should an alternative file management system ever be considered. Examples are the use of the “where” clause instead of the “u_where” clause within entity read triggers, or the use of SQL statements to access the database.

2. SYSTEMS DESIGN

2.1 DATA MODELLING

The systems analysis phase produces a Logical Data Model that is independent of any database management system. This is best catered for by producing an Entity-Relationship (ER) diagram that represents the data requirements of the system as a collection of entities and relationships. An entity may be related to another entity in a one-to-one, one-to-many, or many-to-many relationship, and the links may either be optional or mandatory. Each entity may be included in more than one relationship, and may be the “one” in some and the “many” in others.

Before any development can begin the Logical Data Model must be converted into its Physical form. Like most file systems, UNIFACE is incapable of recognising anything other than one-to-many relationships, therefore all the logical variations must be resolved into this single construct.

The importance of deriving an accurate model, which represents the entire data requirements of the system cannot be stressed greatly enough. UNIFACE is very much a data driven product, not a process driven one, and as such expects that the system can be completely defined in terms of the data (both entities and relationships) that it uses.

It is extremely unwise to take short cuts in the analysis and design phase of a project in the expectation that the details can be resolved at a later stage. Experience has shown this to be a false economy - any day cut out of the analysis phase can add weeks to the development phase. If changes and/or refinements are identified after development has started, the cost of their implementation will increase in direct proportion to the amount of work that has already been completed - the database may have to be re-structured, functions may have to be re-designed and, if already written, will have to be re-worked and re-tested.

2.2 DATABASE DESIGN

The Physical Data Model must be entered into the UNIFACE repository as the Application Model - this is used both to construct the script from which the physical database will be built, and to generate all components (forms, services and reports) which will access the data.

The Application Model will contain the following details:-

- All entity (table) definitions and descriptions
- All field (column) definitions (interface, syntax and layout) and descriptions
- All unique keys (primary and candidate)
- All relationships between entities
- All foreign keys
- Any additional indexes
- Any entity subtypes
- The delete constraints (cascade, restricted, or nullify)
- Database locking requirements (optimistic, cautious, or paranoid)
- Any default trigger code

2.2.1 Database Locking

UNIFACE allows three levels of database locking:-

1) **Optimistic** - the locking of any occurrence is postponed until the last possible moment; that is, when the occurrence is actually stored in the database.

2) **Cautious** - the occurrence is locked at the moment when any database field within the entity is modified.

3) **Paranoid** - the occurrence will be locked at the moment it is read.

Levels (2) and (3) should be avoided as they leave database locks in place while dialog with the user is still taking place. As this could last for extended periods of time it increases the possibility of deadlocks.

In the circumstances where the next available number is obtained from a central database table it would be more prudent to change the locking level to either **cautious** or **paranoid** (by means of a read/lock on that table).

2.2.1.1 Using U_VERSION in a locking strategy

U_VERSION is a special field recognised by the UNIFACE run-time kernel. It may be defined as a field in any entity in the application model. Its benefits are only recognised when using a Cautious or Optimistic locking strategy. This causes the UNIFACE kernel to re-read a row that is being modified, then verify that it has not been changed by another process before applying a lock.

The kernel begins its verification with the first field in the row, comparing each field in order. If the kernel encounters a field named U_VERSION (Data Type = String, Interface Definition = C1) in the row it will check the value of this field for changes. If the value of U_VERSION has not changed UNIFACE assumes that nothing else in the row has changed. Without U_VERSION the UNIFACE kernel will check the value of every field in the row for changes.

If the data source is a relational DBMS the kernel will include the known value of U_VERSION in the "WHERE" clause of the "SELECT" statement. If the value of U_VERSION has changed the row will not be retrieved, and the kernel will not have to perform any comparisons of field values.

If the kernel detects that the row has changed it will stop the process of updating the row. If it has not changed then the kernel will increment the value of U_VERSION and update the row.

It is important to understand that U_VERSION will always contain a printable ASCII character. Thus of the 256 possible values for a single byte only the values between ASCII 32 (space) and ASCII 126 will be used. When the kernel increments the value of U_VERSION and it already contains ASCII 126, it will instead be reset to ASCII 32.

The primary benefit of using U_VERSION is increased performance, but any savings will be minimal unless:

- U_VERSION is placed very close to the beginning of the field definitions for the entity, preferably immediately after the primary key, **and**
- There are a large number of fields, or some very large fixed-length or BLOB fields in the entity definition following U_VERSION.

However, in certain environments the use of U_VERSION can present a risk to application integrity. U_VERSION should NOT be used when:

- The transaction rate is very high and there are common rows being updated simultaneously by many application users, **or**
- An entity will be simultaneously updated by UNIFACE and any other application development language or tool.

Where a single row is updated frequently it is possible that between the time that a process retrieves a row, then checks it for modifications that so many other processes have modified the row that the value of U_VERSION has cycled round to its original value. The process will assume that no changes have been made and update the row, overwriting all the intermediate changes performed by other processes.

Where a row is modified by a non-UNIFACE process which does not increment the value of U_VERSION, and that row has previously been read by a UNIFACE process, any subsequent checking by that UNIFACE process will detect no changes. The row will be updated, and any changes made by the non-UNIFACE process will be overwritten.

As it is doubtful that the performance benefits of using U_VERSION will be noticeable except under extreme circumstances it is therefore recommended that U_VERSION not be used in new UNIFACE applications.

2.2.2 Database Updates

With relational databases the update process is performed in two stages:-

1) **STORE** - which writes the changes to a temporary database area in order to carry out any integrity checks. This temporary area is not accessible to other users, but is accessible to other components run by the same user.

2) **COMMIT** - which transfers the changes from the temporary area to the central database, thus making all changes accessible to other users. This also releases all database locks.

If the **STORE** process detects any errors the **COMMIT** is not performed - instead a **ROLLBACK** will flush all changes from the temporary area, thus resetting all database occurrences to their original values.

It is possible during the execution of a lengthy or complicated function to perform interim stores (a **STORE** without a **COMMIT**), delaying the **COMMIT** until the very end of the function. If there is any user dialog between the first **STORE** and the eventual **COMMIT** this could produce some unexpected and unwanted effects - database locks are issued as soon as the first **STORE** is executed, and the database system may decide for itself that all subsequent records that are retrieved must also be locked. This could eventually lead to locks being applied to significant portions of the database, which will have a detrimental effect on other users, especially if any locks are held for lengthy periods of time due to continued dialog with the user.

The use of interim stores while user dialog is still taking place should be avoided.

It is possible for a logical update to be split across several forms, in which case each form should perform its own STORE, but should leave the COMMIT until all updates have been successful. Provided that there is no intervening user dialog this should not present any problems.

If the available DBMS does not support the COMMIT and ROLLBACK commands then these will be ignored. If a STORE operation fails it is therefore possible to leave the database in an inconsistent state.

If a store is being performed across multiple databases please note that not all DBMS's support two-phase commits (please refer to the relevant DBMS manual for details). If the DBMS does not support this, and there is an error on the commit to the second (or subsequent) database, the updates to the first (or previous) database cannot be rolled back.

Some DBMS's support SQL SavePoints to handle updates which contain large numbers of occurrences (refer to the relevant DBMS manual for details). This option should only be considered in rare circumstances, and may not be available across different database platforms.

2.2.3 Delete Constraints

In a one-to-many relationship the deletion of the “one” entity can be restricted by specifying one of the following delete constraints:-

1) **Cascade** - this will cause all associated occurrences in the “many” entity to be deleted.

2) **Restricted** - this will prevent the “one” occurrence from being deleted if any associated occurrences of the “many” exist.

3) **Nullify** - this will delete the “one” occurrence, and set the foreign key of all associated “many” occurrences to null. This setting should be used in those cases where the relationship has been defined as optional.

The level of constraint must be chosen carefully so as not to leave the database in an inconsistent state following the deletion of the “one” entity in any relationship.

⚠ The processing required by these constraints cannot be carried out unless the “many” entity has been defined in the component within the boundaries of the “one” entity.

2.2.4 Database Views

Some DBMS's support Database Views. These are virtual tables (entities) that do not actually exist as physical tables within the database, but which are constructed from physical tables as and when requested.

When a process accesses a table name which has been defined as a View the DBMS will construct the table (in memory only) from the physical table(s), using any selection criteria provided, and will pass each constructed record to the requesting process in the same way that records from physical tables are passed to the process.

A process does not have to perform any special processing in order to access a View. It merely accesses a table in the normal way, and the DBMS detects that it has been defined as a View, and acts accordingly.

Each View contains a group of fields (columns), as do ordinary tables, but each field can be obtained from any table within the system, or even from other databases. The View definition can include selection criteria for each of these tables, such as "where table.column = value", meaning that any filtering is performed by the DBMS before data is presented to the requesting process.

The advantages of using database Views are:-

- An arrangement of several physical database tables may instead be referenced within a component through a single logical table, which reduces the effort required in creating components that access that arrangement of tables.
- The construction of the View, including any data filtering, is performed by the DBMS on the server device, which may be significantly faster than the client device.
- Network traffic (ie: from the database server to the client device) is kept to an absolute minimum, thus avoiding a potential bottleneck. It is more efficient to have unsuitable records discarded by the DBMS on the database server than it is to send them over the network only to have them discarded by the client.

The disadvantages of using database Views are:-

- Views can only be used when supported by the underlying database. The use of Views may restrict the range of database engines that could potentially be used to support the application.
- Although Views must be defined within the application model before they can be referenced by any UNIFACE component, they must be built into the database using manual procedures. The database scripts created by UNIFACE can only be used to create real tables, not virtual ones.

⚠ Although it may be possible for a process to write to a database View be aware that a View may not contain all the required (not null) fields for each of the physical tables which are referenced. When the DBMS moves the field values from the View to a physical table before performing the write operation, if any required fields have not been supplied with values this will result in a store error.

2.2.5 Database Triggers

Some DBMS's allow processing to be carried out by the database itself as well as the application that manipulates the database. This processing can be defined in the form of triggers that are automatically fired when a particular activity takes place on a specified table (entity). These activities may be any one of the following:-

- before or after an INSERT
- before or after an UPDATE
- before or after a DELETE

This allows particular processing to be defined just once for an entity regardless of the number of forms that access that entity. The processing that may be performed by these triggers could include a mixture of the following:-

- the reformatting of data before it is written to the table
- performing validation against entries on other tables
- creating entries on other tables

The database trigger will be fired by the UNIFACE STORE command, therefore if a trigger fails the only error message that can be communicated to the user is "STORE error, see Message Frame for details". The Message Frame will contain nothing more than the name of the trigger which failed, but not the precise reason for the failure, as shown in the following example:-

```
I/O Function: W, mode: 0 on file/table: REQUEST_PRODUCT length: 204
ORA-06510: PL/SQL: unhandled user-defined exception
ORA-06512: at line 224
ORA-04088: error during execution of trigger "OU_DBA.MAKE_PRODUCT2"
ORACLE Driver Error (-37): Write driver function failed.
```

The disadvantages of using database triggers are:-

1) The UNIFACE application would not have any knowledge of any database triggers.
2) Database triggers could be turned on or off without the knowledge of the application.
3) Processing logic would be held in two separate areas, which could possibly lead to a maintenance problem.

2.2.6 Stored Procedures

Stored procedures are simple programs, or procedures, that are defined within the database and executed in the server. The user can create a procedure that contains several SQL statements or a whole transaction, and execute it with a single call statement. The usage of stored procedures reduces network traffic and allows more strict control to access rights and database operations.

Unlike database triggers a stored procedure must be called from within the UNIFACE application, but cannot be dropped without causing the appropriate result being returned.

Procedures can take several input parameters and return a single row as a result. The resulting row is built using whatever output parameters were specified.

Procedures are owned by the creator of the procedure. Specified access rights can be granted to other users. The procedure has the creator's access rights to database objects when it is run.

⚠ The disadvantages of using stored procedures are:-

1) The UNIFACE application would have to contain code to explicitly activate each procedure.
2) Processing logic would be held in two separate areas, which could possibly lead to a maintenance problem.

2.3 FUNCTIONAL DESIGN

Once the number of functions or transactions has been identified, and their individual requirements analysed, the components which will satisfy these requirements can be specified. In this context a “function” or “transaction” is a single logical process as viewed by the user, and a “component” is a single module within the UNIFACE application. A user function (transaction) may involve any number of components, just as it may involve many occurences from many database tables.

A full description of the techniques of functional analysis will not be included here as it would require its own separate manual.

2.3.1 Basic Principles

The design of individual form components (those which contain screens) within the system should follow these basic principles:-

a) It is absolutely essential that there is a clear, complete, and accurate definition of the problem before any attempt can be made to design the solution.

b) It is equally important that the solution be adequately designed before any attempt can be made to build it.

c) To be consistent with any other application that may be available on the PC any conventions adopted by Microsoft WINDOWS that are also available within UNIFACE will be used wherever possible. This should enable any user to switch between different applications on his/her PC without noticing any major differences.

d) The design should be UNIFACE-friendly. Requests to bypass standard behaviour should be avoided. Experience has shown that to implement non-standard or eccentric requests (eg: to mirror the appearance or behaviour of a different application) is usually difficult, is rarely cost-effective, and can cause major problems when the time comes to upgrade. If there is ever a conflict between the design and the development tool it is always easier, and safer in the long term, to change the design to fit the development tool rather than to bend the development tool to fit the design.

e) Undocumented or unsupported features will not be used - these can be changed or withdrawn by the software supplier at any time without notice. Any problem that appears as a result of their usage on the instructions of the client cannot be treated as a fault.

f) Individual components should be kept as small and simple as possible. It is far better to have a large number of small components than a small number of large components.

g) Individual forms should not be allowed to be larger than the visible screen area. Scrolling forms and split screens should therefore be avoided. If there is more data than can comfortably fit onto a single screen then the overflow can be handled by auxiliary screens. (see Section 4.2).

h) Each form should be limited to a single function (eg: add, enquire, modify, delete) wherever possible. Multi-function forms should either be avoided, or kept to a minimum.

i) Each component should perform its processing on the minimum number of occurrences (eg: Create Customer should only create a single customer at a time - it should be called again for each additional entry). This will prevent the user from accumulating a large number of updates which could cause a delay to other users if applied to the database in a single store operation.

2.3.2 Screen Size and Complexity

It used to be common practice to design forms that filled the whole of the screen area, mainly because it was only possible to display one form at a time, and any area that wasn't used was simply blank. They were also designed to perform as many actions as possible without having to switch to another form as the time it took to repaint another form was incredibly slow.

With the arrival of the Graphical User Interface (GUI) and modern screen technology it is now possible to design forms (windows) of any size, to have several forms visible on the screen at any one time, and to have forms popup very quickly when needed, and to disappear again just as quickly when completed. Not only is it possible to design smaller simpler forms, it is now the preferred method as they the following benefits:-

1) It is easier to design and specify a large number of small components than it is for a small number of large components.

2) A large number of small, simple components can be developed simultaneously by a large programming team, whereas a small number of large, complex components can only be developed by a correspondingly small programming team.

3) If a change is made to a component then the whole component must be re-tested to ensure that a bug has not been introduced into any other area - this obviously takes longer for a large, complex component.

4) Users will usually find it easier to work with small, simple form components rather than large, complex ones.

5) The load time for a small screen containing a small number of database occurrences is faster than for a large screen containing many occurrences as it would require less initial data to be retrieved from the database. If users want fast response times this cannot be achieved with large screens.

6) It is easier for a user to display another page of details by clicking on a button to invoke another component than it is to scroll up or down through the entire contents of a very large screen.

7) If a component accumulates a large number of updates before they are committed to the database this will place a heavy load on the system and increase the possibility of a locking conflict. It is better to update little and often rather than many and seldom.

8) If there is requirement for different levels of security for different processes or functions, this will be easier to implement if each different process is performed within a separate component.

9) There are physical limits to the size of individual components, and a complex component that accesses a very large number of database tables, and includes a large amount of procedure code can hit this limit and refuse to run properly (if at all).

10) It is easier to take a print of a small screen for inclusion in user documentation.

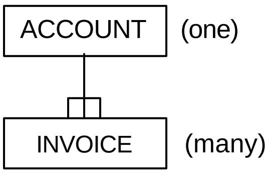
11) If it is desired to make the application available on the World Wide Web then the larger and more complex the component the greater the degree of difficulty.

2.3.3 Form Design

2.3.3.1 Entity Frames

In order to access a database entity within a UNIFACE component it is first necessary to define (paint) a frame for that entity within the component. A frame is a rectangle with variable dimensions, the minimum size being 1 character wide by 1 character high. The size of each entity frame should be large enough to contain the fields that are to be displayed from that entity, plus any inner entities and their fields.

Consider the following Entity-Relationship (ER) diagram:-



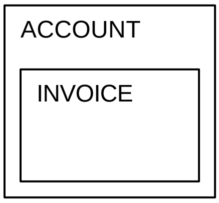
This is the classic One-to-Many relationship.

Each ACCOUNT can have any number of INVOICE entries.

Each INVOICE is owned by a single ACCOUNT.

These entities may be painted on a form in any of the following combinations:-

a)

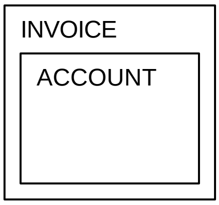


ACCOUNT is the outer entity.
INVOICE is the inner entity.
The relationship between outer and inner is One-to-Many.

The <retrieve> sequence is ACCOUNT then INVOICE (retrieve 1st ACCOUNT, retrieve INVOICES for 1st ACCOUNT; retrieve 2nd ACCOUNT, retrieve INVOICES for 2nd ACCOUNT, etc)

An ACCOUNT may have multiple (or zero) INVOICE entries. Only those INVOICE entries that are related to the current ACCOUNT can be shown at any one time. It is not possible to show INVOICE entries for more than one ACCOUNT within the same screen display.

b)

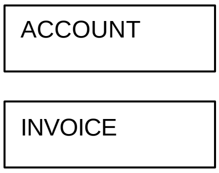


INVOICE is the outer entity.
ACCOUNT is the inner entity.
The relationship between outer and inner is Many-to-One.

The <retrieve> sequence is INVOICE then ACCOUNT.

Each INVOICE can have only one ACCOUNT.
Each ACCOUNT may be attached to more than one INVOICE.

c)



There are no inner entities, only outer entities in parallel.

There are no relationships to be considered within the form regardless of any that have been defined within the Application Model.

Occurrences of one entity can be retrieved independently of occurrences in the other entity.

The following points should be noted:-

1) Any database field should be painted entirely within the frame of the entity to which it belongs. but not within the area covered by any inner entities. Values can then be retrieved from, and written to, the database occurrence without the need for any additional procedure code. If a field is painted within an area in which it does not belong it becomes a “dummy” field within that area, and will require explicit code to move data between the “dummy” field and its “real” counterpart.

2) Database retrieval takes place from the OUTER entity first, followed by its immediate INNER entity (top to bottom, if there are parallel entities), followed by their inner entities, and terminating at the innermost entity. It is not possible to begin the retrieval on an inner entity and then retrieve its associated outer entity.

3) If the INNER entity is related to the OUTER entity (as defined in the Application Model) then only those occurrences of the inner entity that are related to the outer entity will be retrieved.

4) If no relationship has been defined in the Application Model between the outer and inner entity UNIFACE will, by default, retrieve all occurrences of the inner entity. It is possible to override this by specifying additional procedure code to restrict the retrieval, but if any sort of relationship exists it should be defined in the Application Model. If no relationship exists then it is not logical to paint the entities in this way.

2.3.3.2 Retrieve Profiles

A Retrieve Profile is the set of "*where <fieldname> = value*" statements which is used to limit the number of occurrences that are to be retrieved from the database. Only those occurrences that satisfy all the selection criteria defined in the profile will be selected.

Retrieve Profiles can be defined in the following ways:-

- The user is presented with a blank screen. Values are entered into the screen as required, then a manual <retrieve> is invoked.
- The form uses profile values passed to it from the previous form, and performs an automatic <retrieve> before passing control to the user.
- Profile values can be hard-coded into the entity's <read> trigger within individual forms.

Retrieve Profiles have the following characteristics:-

1) When one entity on a form is painted inside another entity, and a relationship between the two entities exists in the Application Model, the retrieve profile of the inner entity is automatically set to the field names (as defined for that relationship) and values passed down from the current occurrence of the outer entity. If there is no relationship defined there is no automatic profile. Additional profile values may be defined within the form to limit the selection further.

2) If no profile is defined (or the values are null) then a serial read (full table scan) will be invoked to select all occurrences from that table. The time taken to complete a full table scan will be directly proportional to the size of the file, therefore online functions should avoid this type of access on large files (unless stepped hit lists are available - see following section) as the response times would probably be considered as unacceptable.

3) If the profile contains a field which has its own index in the database the <retrieve> will be performed in less time as it always quicker to scan an index rather than the data. By default all primary keys are indexed - other indexes may be added as and where necessary (eg: to foreign keys, or other frequently used search items).

4) If an indexed read is not possible the alternative is a serial read (full table scan).

5) Normally a second <retrieve> is not possible until the results of any previous <retrieve> have been <clear>ed. If a component has been designed with separate areas for the entry of retrieve profiles and the display of their results, and has been programmed to use **retrieve/a**, then the results of a second retrieve can be appended to the results of the first.

6) As the <retrieve> starts from the outer-most entity and works its way inwards, the retrieve profile should be restricted to fields which exist on the outer-most entity. Inner entity profiles that cause outer entities to be discarded if there are no qualifying occurrences of the inner entity should be avoided as they generally do not operate with acceptable response times. In these circumstances the designer should consider the use of a database view.

2.3.3.3 Hit Lists

When a <retrieve> is issued for a database table the database engine will construct a hit list containing the unique identities (primary keys) of all occurrences from that database table which satisfy the selection criteria (retrieve profile). If no profile is supplied then all available occurrences will be retrieved, and the size of the hit list will equal the capacity of the database table. The hit list is passed back to the UNIFACE component, which uses it to fetch the data for each occurrence from the database into the component as and when required.

2.3.3.3.1 Stepped Hit Lists

A stepped hit list offers an increase in performance as each “step” is made available to the UNIFACE component as soon as it is available. The size of each “step” may differ from one database system to another. The database handler will not construct the next step until it is actually required by the component, thus postponing the necessary data access until that time. It is not possible to provide the total size of the hit list until the last step has been read in.

⚠ If the underlying database does not support stepped hit lists then the entire hit list must be constructed in a single operation. Serial reads (full table scans) on large-capacity files may therefore result in poor response times.

2.3.3.3.2 Sorted Hit Lists

If the retrieve profile is issued with an “order by” clause to force the occurrences to be presented in a particular sequence, this may incur an overhead as all the entries may have to be scanned before they can be sorted into the specified sequence. If the underlying database supports the “order by” clause it will be able to sort the entries before the hit list (stepped or not) is made available to the form. If this sorting cannot be done by the database it will be done by UNIFACE (within the kernel), but the overhead will be greater and stepped hit lists will not be available.

⚠ If the underlying database does not support sorted hit lists then “order by” clauses on large hit lists may result in poor response times.

✖ It is not possible to specify an “order by” clause for an entity using fields which do not exist on that entity.

ℹ It is possible to sort occurrences using any combination of fields from any entity using the “sort” statement within proc code, but this will require the hit list to be complete. If it is not complete then the operation will be suspended until it is.

2.3.3.4 Network Traffic

In a Client/Server configuration the slowest component tends to be the communications network between the client and the server. The transmission of overly large volumes of data to satisfy a user request can therefore be a major contributor to poor response times.

2.3.3.4.1 Data Filtering

It is technically possible within a UNIFACE component to retrieve a large number of occurrences, examine them, and discard those that do not satisfy the selection criteria. This is one way of trying to deal with selection criteria that covers more than one entity. As this causes data to be transmitted across the network before it is filtered out of the view that is made available to the user, it is one of the primary causes of poor response times. If any data filtering needs to take place it would be far better for this to be performed on the databse server device rather than the client device so that only data that has already been filtered is sent over the network, thus reducing the amount of network traffic to the bare minimum.

This type of data filtering can be performed in a UNIFACE component called a self-contained service (see section *2.3.3.4.3*), or by the database engine itself by using a View (see section *2.2.4*).

2.3.3.4.2 Database Functions

There are occasions when it is necessary to scan a number of database occurrences in order to obtain a single value, such as:

- The COUNT of occurrences that satisfy the selection criteria.
- The MINIMUM, MAXIMUM or AVERAGE values of a column (field) for a selection of occurrences.
- The SUM of all values in a column (field) for a selection of occurrences.

In these cases it would not be very efficient to transmit all the occurrences across the network so that they could be processed by the client - it would be far better if this activity were performed on the server so that the only network traffic would be the result. This can be achieved by using the **selectdb** command within a UNIFACE component, which causes only the result(s) of the command to be passed back across the network. If these functions are not supported within the underlying database they will be performed by the UNIFACE kernel.

A possible alternative to the **selectdb** command would be a stored procedure, as discussed in an earlier section.

2.3.3.4.3 Remote Services

An additional option within UNIFACE is the Self-Contained Service that can be run on a server device instead of the client. A Service is a component that has no visible layout at run time (a hidden form) in order to carry out a function which may involve accessing large amounts of data.

The advantages of running a Service on the same device as the database engine are:-

- No data (apart from passed parameters) is sent across the network.
- The Service runs at the speed of the server device, which may be superior to that of the client.

It would be possible, therefore, to create a Service that filtered out any unwanted data before transmitting the results across the network to the parent form on the client device.

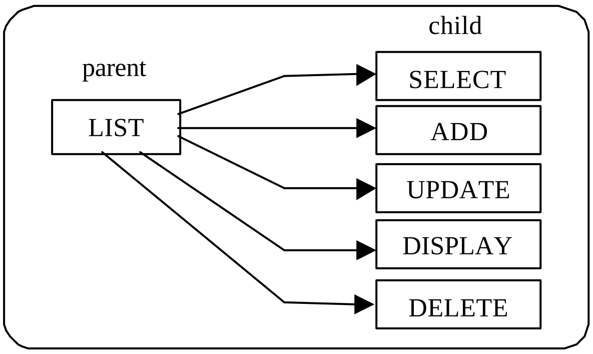
2.3.3.5 Modal and Non-Modal Forms

If a form is **modal** it means that focus cannot be switched from the current active form to another other than by passing control to that form, either by a parent form invoking a child, or by a child form returning control back to its parent. If the user moves the cursor outside the boundaries of the current active window (form) it becomes disabled.

If a form is **non-modal** then the user is allowed to switch focus from one form to another just by clicking the mouse on any other form that is available in the application window.

Instead of creating a single large component which performs all possible functions it is better to split it into a family of smaller components, each performing a single function. This can best be described using a typical family of forms as in the following example:-

This family starts with a parent of dialog type of LIST. This has buttons that pass control to child forms of type SELECT, ADD, UPDATE, DISPLAY and DELETE, as shown below:-



With **modal** forms only one child form can be active at any one time. Once a child form has been activated focus remains with that form until it either exits and returns control to its parent form, or it activates a child of its own. If, after activating a child form the user moves the cursor out of the area encompassed by that form it will not have any effect.

With **non-modal** forms it is possible for the user to activate a child form, then switch focus back to the parent form by clicking the cursor within the boundaries of that form. The user can continue working within the parent form, and may even activate another child form. The title bar of the form that has focus will be highlighted - all others will be greyed out.

Using the example above the user may activate the ADD form from the LIST form, then click the cursor back onto the LIST form in order to activate another child form. This may be repeated until all child forms have been activated. All these forms will be available in the application window, but only one will have focus at any one time. The user will be able to switch focus from one form to another just by clicking the cursor within the boundaries of the required form.

Non-Modal forms have the following characteristics:-

- A **non-modal** form can only be activated by a **non-modal** parent.
- A **modal** form can only create child forms that are **modal**.
- Any **non-modal** form will be automatically terminated when its parent is terminated.

⚠ It could be assumed that an entire application could consist of **non-modal** forms so that the user could switch focus from any form in the system to any other. However, in real life it is not practicable to have more forms active than can fit comfortably within the application window (or the PC's memory) at any one time.

2.3.3.6 Interaction between Non-Modal Forms

The abilities of non-modal forms, coupled with the ability of components to send messages from one to another, provides an interesting choice of possibilities when dealing with a family of forms.

An example would be use the family of forms from the previous section. The LIST form displays multiple occurrences, with the details of a single occurrence summarised on each line. The ADD, UPDATE and DISPLAY forms each deal with a single occurrence, with the details expanded to fill the screen. Using these forms, the following scenarios are possible:-

1)
   - The user selects occurrence 1 in form LIST, then presses the navigation button to activate the DISPLAY form to show details of this occurrence.
   - Without terminating this DISPLAY form the user returns to the LIST form, selects a different occurrence, then activates a new copy of the DISPLAY form for this occurrence.

   There are now two copies of the DISPLAY form, but each contains the details for a different database occurrence. This is possible because each form component is activated with an instance name, which may be the same as the form name (allowing only one copy at a time to exist) or it may be constructed with a unique name (allowing a separate copy to be created for each occurrence).

   This allows the user to have multiple copies of the same form visible at the same time so that data from different occurrences can be compared more easily. In a modal environment this would be much more difficult.

2)
   - The user selects occurrence 1 in form LIST, then activates the DISPLAY form to show details of this occurrence.
   - Without terminating this DISPLAY form the user returns to the LIST form, selects a different occurrence, and the contents of the DISPLAY form changes automatically to match the newly selected occurrence.

   There is only one copy (instance) of the DISPLAY form, and each time the user selects a different occurrence in the LIST form this same instance is re-activated with the primary key of the selected occurrence instead of a new instance being created.

   This allows the user to activate the child instance only once, then change its contents by simply selecting another occurrence in the parent form. In a modal environment the user would have to exit from the child form back the parent, make another selection, then press the navigation button again to invoke the child form.

The choice between these two scenarios does not have to be fixed in the design of the application. The user has the ability to switch between these two options at run time by selecting one of the entries on the pulldown menu bar. The CHILDREN menu contains an option called REFRESH which can either be ON (displayed with a tick) or OFF (displayed without a tick). If the REFRESH option is set ON then scenario 2 will be in effect, which will cause the contents of each child form to be automatically refreshed each time a different occurrence is selected in the parent form. The initial value for this option can be set in the assignment file (please refer to the *Menu and Security System User Guide* for more details).

This ability is available between parent and child forms where the parent is of dialog type LIST. The ability to communicate between grandparents/grandchildren, other sibling instances, or unrelated instances, would have to be specifically built into the design and coded accordingly.

2.3.4 Function Specifications

Before a developer can create a component a Functional Specifications should be created, and it should contain the following:-

a) The Function identity (ie: component name, title).

b) Dialog type (eg: ADD1, LIST1, etc.). This is used to identify the Component Template from which the component will be built.

c) A description of the function's objectives.

d) A screen/report layout (not applicable to services).

e) A CRUD matrix to indicate which entities within the database are accessed, and whether they are **C**reated, **R**ead, **U**pdated or **D**eleted.

f) An ER (Entity Relationship) diagram which identifies the entities and relationships required for the function - although there may already be a single ER diagram which covers the whole database this may be confusing as it contains entities and relationships that are not used within this component. There may be several different paths between a group of entities, and only one path will produce the optimum result. If an entity is referenced more than once then an alias (subtype) name needs to be identified, along with its position within the ER diagram. If a group of entities were accessed by a series of functions then it would be acceptable to create a single diagram and use it for all the functions in the series.

g) The identity of the OUTER entity - this is especially important if the function accesses a large number of entities as a mistake here could produce confusing results.

h) A description of any input or output parameters, if different from the standard dialog type.

i) A description of any input messaging (via the <async interrupt> trigger) or output messaging (via the **sendmessage**/**postmessage** command), if these differ from the standard dialog type.

j) A description of any operations that may be required for this component to be activated by another component.

k) Any special processing considerations (eg: items not covered in, or different from the behaviour identified in the standard dialog type).

2.4 SYSTEM SECURITY

The security requirements of the system need to be considered and defined, otherwise the following defaults will apply:-

- All users will have access to all transactions.

- Within a transaction each user will have the same access to all fields/items.

- Within a transaction each user will have access to all available data on the database.

2.4.1 Function Access

If it is required that within a specified transaction the ability to Create, Read, Update or Delete is only be given to certain users rather than all users, then this multi-function transaction must be split into a series of separate transactions, each performing a single function

The authorisation for specific users to activate specific transactions will then be the responsibility of a separate menu and security system. This will allow the transaction to be selected by the user only if the relevant authority has been granted. If no such authorisation has been granted then the selection will be rejected. The menu system may have the ability to display to the user only those transactions to which access has been granted, which means that the user will not be able to see, let alone select, any transactions for which access has not been authorised.

The validation of particular user/transaction access will be a function of the menu and security system, and not the individual transaction. Once a transaction is called it will assume that all the necessary authorisation validation has been passed, and the user will have access to all operations within that transaction without the need to re-validate.

2.4.2 Field / Item Access

If it is required that within a specified transaction the ability to view or modify the data within specified fields is only be given to certain users rather than all users, then the relevant user/field combinations will need to be defined within the security database.

This will take advantage of the facility within UNIFACE to make a field in a form both invisible and inaccessible to the user with a command that is executed at run time. This will make a single form appear to be different for users with different security provisions, thus reducing the need to create separate versions of the form for different users.

As well as functions in the menu and security system to define which fields in which forms are to be turned off for which users, additional processing will be required within each of these designated forms to access this list of fields so that access can be turned off when required.

Care should be taken to avoid the situation where the validation of a field on a form requires data to be input to another field to which access has been turned off.

2.4.3 Data Access

Where a transaction has been designed to access certain files/tables within the system (eg: customer data) but it is required that certain users can only access a certain subset of that data (eg: only those customers within a particular region) then this validation will need to be performed within the function in question.

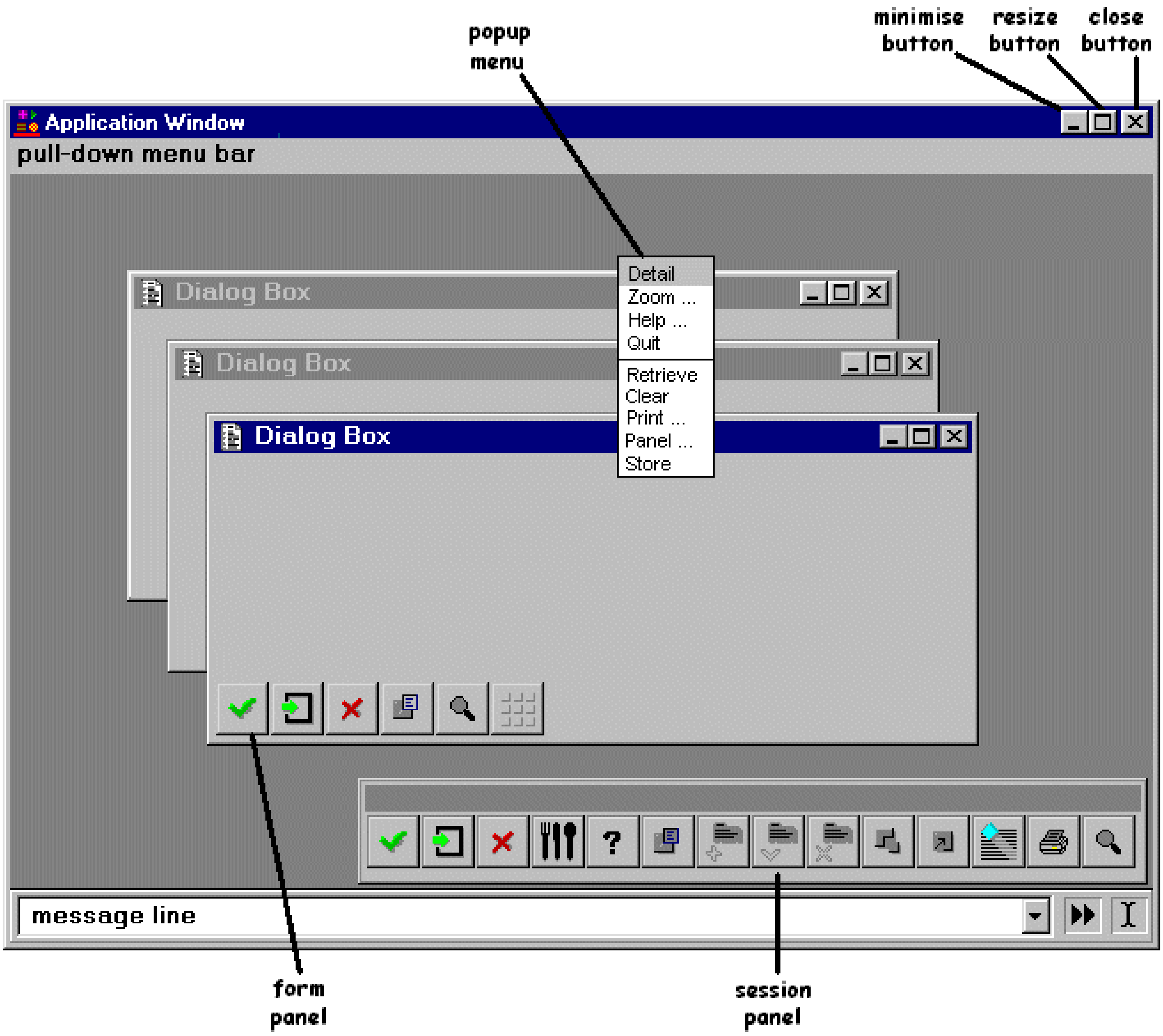
This type of validation cannot be performed by the menu and security system as it relates to the structure and content of the application database, which is unknown to the menu and security system.

It is assumed that the application database will be designed to allow this data subsetting to be defined, thus allowing the necessary validation checks to be performed easily and efficiently.

3. SCREEN COMPONENTS

This section identifies the components (known as “widgets” in UNIFACE) that are available when designing and constructing individual screens (forms). These are a mixture of standard UNIFACE components coupled with conventions adopted by A J MARSTON .

3.1 FULL SCREEN LAYOUT



This represents the complete screen as is visible to the user, with the title bar at the top and the message line at the bottom. Inside this application area are one or more dialog boxes.

Each dialog box represents the screen that is used by a form component. It is possible to create several dialog boxes and have them appear in the application area at the same time, but only one of them can have focus (be processing) at any one time. Focus can be moved from one dialog box to another simply by moving the cursor into another dialog box and clicking with the mouse. When a dialog box loses focus its title bar will be dimmed.

3.1.1 Fonts

Unless specified otherwise it is assumed that a standard SVGA monitor will be used, with a minimum resolution of 800 x 600, and at least 256 colours. The maximum dimensions for each dialog box will vary according to the base screen font (defined in the initialisation file), as shown in the following table:

Font	Width (characters)	Depth (lines)
Courier New 8 point	110	30
Courier New 9 point	110	28
Courier New 10 point	96	26

The following dimensions apply if the screen resolution is increased to 1024 x 768.

Font	Width (characters)	Depth (lines)
Courier New 9 point	143	40
Courier New 10 point	125	37
Courier New 11 point	106	34

Separate fonts can be defined for fields, labels, buttons, message line, etc. These can be changed in the .INI file without having to alter the application, but responsibility for the effects of any such customisation resides with the person who changes the .INI file.

⚠ Courier New is a WINDOWS base font. Only base fonts should be used as these are the only ones that are guaranteed to exist on every PC. It is recommended that the use of additional fonts that are installed with optional products (eg: WORD) be avoided.

3.2 CONTROL AREAS

3.2.1 The Pull-Down Menu Bar

This is an example of a list of menus that could appear on the menu bar, and the items that would appear if one of these menus were to be selected.

MENUS	MENU ITEMS	DETAILS
File	Clear	Clear all data on the current form
	Retrieve	Retrieve occurrences from the database
	Accept	Leave the current form (after applying any outstanding updates).
	Store	Aply updates to the database
	Detail	Fire the <detail> trigger
	Print	Print the current screen
	Write to File	Dump the contents of the current field out to a disk file
	Fetch File	Load the contents of the current field from a disk file
	Quit	Leave the current form (without applying any updates).
Edit	Cut	Cut the selected text
	Copy	Copy the selected text
	Paste	Insert the text selected via the previous Cut/Copy
	Add Occurrence	Create a new occurrence after the current occurrence
	Insert Occurrence	Create a new occurrence before the current occurrence
	Remove Occurrence	Remove the current occurrence
	Erase	Erase all fetched occurrences in the current form
	Profile	Provide a window to enter a profile before finding text
	Find Text	Find text in the current field according to the previous Profile
	Font - Underline	Toggle the UNDERLINE attribute on the selected text
	- Bold	Toggle the BOLD attribute on the selected text
	- Italic	Toggle the ITALIC attribute on the selected text
View	First Occurrence	Go to First Occurrence.
	Prev Occurrence	Go to Prev Occurrence of current entity.
	Next Occurrence	Go to Next Occurrence of current entity.
	Last Occurrence	Go to Last Occurrence.
	Next Frame	Go to Next Occurrence Window.
	Prev Frame	Go to Prev Occurrence Window.
	Zoom	Show the current field data in a bigger window (for those cases where the amount of data is larger than the size of the field on the current screen).
	QuickZoom	As above, but provides the maximum-sized window immediately
	Panel	Toggle the session panel ON or OFF
Application	Sort Ascending	Sort current screen contents on the current field, in ascending sequence (only applicable in forms with multiple occurrences in the screen display).
	Sort Descending	As above, but in descending sequence.
	Refresh Children	If ticked ON all child instances will be automatically refreshed each time a new occurrence is selected in its parent.
	Delete Children	Delete all child instances that are attached to the current instance.

Help	Show Help	Show help text on the current field/form (described in a later section).
	About Program	Will identify the form which is running (described in a later section).
	Keyboard Map	Will show the current keyboard map (described in a later section).
	Message Frame	Display the message frame.
	Debug	(only available to selected users)

ℹ These are the options that are currently set by the A J MARSTON Menu and Security system. It is possible, however, to implement a different set of options for each application or individual function.

ℹ Different shortcut keys can be defined for each of these options by customising the [accelerators] section in the .INI file

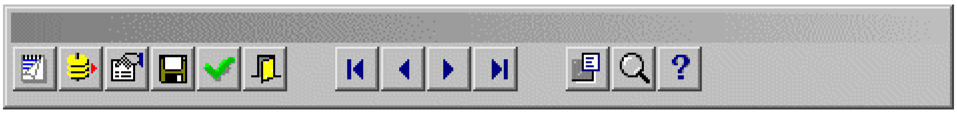
3.2.2 Panel Bars

These are optional areas that contain a series of panel buttons that provide an alternative method of selecting some of the UNIFACE triggers or structure-editor functions. To find out what a particular panel button means move the cursor until it is over the button and a tool tip will appear.

These areas are usually turned off as each transaction contains its own set of buttons in the action bar area (see below)

3.2.2.1 Session Panel

This is fixed for the application and is defined in the start-up shell. This is the panel provided as SESSION in the USYS library:



These buttons have the following meanings:

	clear	⏮	first occurrence		message frame
	retrieve	◀	previous occurrence	🔍	zoom
	detail	▶	next occurrence	?	help
💾	store	⏭	last occurrence		
✓	accept				
	quit				

3.2.2.2 Form Panel

This is defined within the properties of each form when it is compiled. It is possible for each form to use a different panel.

3.2.3 The Popup Menu

The Popup menu is activated by pressing the right-hand mouse button. The menu box will appear wherever the cursor is currently positioned on the screen. While keeping the right-hand button depressed, move the cursor up or down until the required command is highlighted, then release the right-hand button to activate that command.

To cancel the Popup menu move the cursor to a position outside the menu box, then release the right-hand button.

Different Popup Menus can be defined for each of the following objects:-

a) In the Application Start-up Shell - this is available globally.
b) For a Form Component - this is available only within this form.
c) For an entity within a component - this is available only within the area defined for this entity.
d) For a field within an entity - this is available only for this field.

3.2.4 The Message Line

The message line is a permanent area at the bottom of the screen where messages (such as validation errors) are shown.

The font used for messages can be changed by altering the **combo=** setting in the **.ini** file.

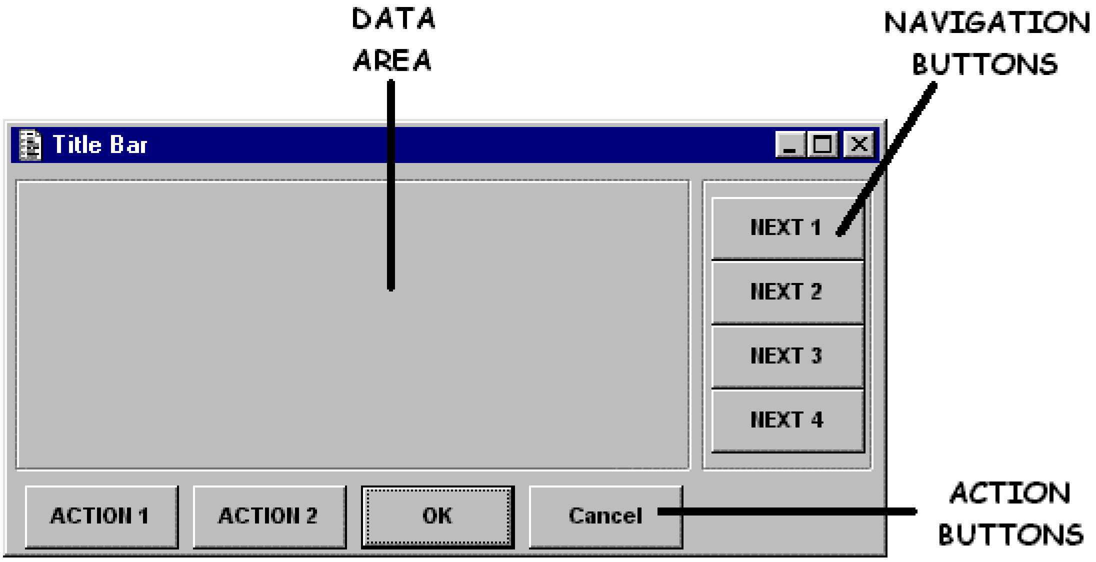
The message line shows the most recently issued message, but the ⬇ button on the end of the line can be used to view the history of previous messages.

3.2.5 The Message Frame

This area is not part of the standard screen, but is a secondary area that is maintained in the background. Its contents can be examined on request. It is normally used for special information such as errors detected by the database handler, or where a function generates multiple lines of information (eg: an audit trail) that would be lost in the single message line.

It is possible for the contents of the message frame to be automatically copied to a file on disk. This is especially useful as the volume of text that is available for online viewing may be exeeded, which causes the earliest text to be lost. It is also possible for components to empty the message frame programatically, but this does not affect the disk copy.

3.3 THE DIALOG BOX



Each form within the system should follow this standard layout, which contains the following areas:-

- A **Title Bar**, which contains a brief but meaningful description of the transaction. This description may be obtained from any one of the following sources:

  - may be hard-coded within the form
  - may be obtained from the message file at run time (the preferred method)
  - may be obtained from the menu system at run time

- A **Data Area**, which contains whatever data is manipulated by the transaction. The contents of this area will vary according to the needs of the individual transaction.

- A series of **Action Buttons**, which allow the user to perform actions on the current data.

- A series of **Navigation Buttons**, which allow the user to call other transactions.

Button bars for action and navigation buttons are preferred as they instantly show to the user what options are available within that form, and can be selected with a single click of the mouse. There are alternative methods, but they each have their own disadvantages:

- Dropdown menu items are not initially visible, and require more than one mouse click.
- Shortcut keys are difficult to remember, and usually require more than one key press.
- UNIFACE panel bars often show generic options which may not be relevant within the current form. Some options require the cursor to be positioned within an object before a button is pressed so that the function acts on that object instead of another. This again is more than one mouse click.

There may be additional borders within the data area, but this will depend on the requirements of each individual form.

Note that some forms may not contain all of these areas.

3.3.1 Prompt Sequence

The default prompt sequence in all form components is from left-to-right, top-to-bottom. The cursor should (generally) start at the top left-hand field on the screen, and end up on the bottom right-hand field. After leaving the last field in the data area the cursor should move along the buttons in the action bar from left-to-right.

If there are buttons on the navigation bar these should be traversed (from top-to-bottom) after the data area and before the action bar.

The cursor may also be made to travel backwards, in which case the prompt sequence should be the reverse of the forward movement.

The precise sequence can be modified within individual screens, but care should be taken to keep it logical otherwise this could cause confusion.

If the form contains multiple occurrences of an entity (as indicated by a scroll bar) the prompt sequence will only traverse the fields on a single (the current) occurrence. Movement from one occurrence to another is described in a later section.

If the form contains a Profile area and a set of CLEAR and RETRIEVE buttons, then the prompt sequence will vary depending on whether the Data area is empty or not. If it is empty (ie: no data has been retrieved yet), then the prompt sequence should include the Profile area and the Action Bar, but exclude the Data area and Navigation Bar. If data has been retrieved the prompt sequence should include the Data area, the Navigation Bar and the Action Bar, but should exclude the Profile area.

Where a form contains a mixture of amendable and display-only fields it is normal practice to exlude the display-only fields from the prompt sequence.

3.3.2 Cursor Movement

3.3.2.1 From Field to Field

The cursor can be moved from one field to another in a form in the following ways:-

⇒ With the TAB key - this will leave the current field and jump forwards to the next field in the prompt sequence.

⇒ With the BACKTAB key (SHIFT+TAB) - this will leave the current field and jump backwards to the previous field in the prompt sequence.

⇒ With the mouse - move the mouse arrow to a field, then click with the left-hand mouse button to select that field. This then becomes the “current” field, and its database occurrence becomes the “current” occurrence.

Note that the cursor control keys ← ↑ ↓ → will only allow the cursor to be moved within the current field. For radio groups and dropdown lists these keys will move the cursor between different options within the list.

3.3.2.2 From Occurrence to Occurrence

Where a form contains multiple occurrences (as signified by a scroll bar) the cursor can be moved from one occurrence to another in the following ways:-

⇒ By using the mouse to click on a field in another visible occurrence.

⇒ By using the mouse to click onto the scroll bar (described in more detail below).

⇒ By selecting the “Next Occurrence” or “Previous Occurrence” option from the pull-down menu, if these have been made available.

⇒ By using the occurrence navigation keys as defined in your current keyboard map (from your pull-down menu select HELP=>KEYBOARD HELP for more details on your current settings). The following table shows typical key strokes:-

Keystroke	**Action**
Alt+PgDn	Next Occurrence
Alt+PgUp	Previous Occurrence
Shift+Alt+PgDn	Next screenful of Occurrences
Shift+Alt+PgUp	Previous screenful of Occurrences
Ctrl+Alt+PgDn	Last Occurrence
Ctrl+Alt+PgUp	First Occurrence

3.3.3 Active Field / Occurrence

The location of the cursor within the current prompt sequence will normally be indicated to the user by highlighting the currently active field (the field which has focus) in a different colour. If the form contains multiple occurrences of an entity (as indicated by a scroll bar) then all the fields of the active occurrence can be highlighted.

As different colours can be defined for **field** highlighting and **occurrence** highlighting it will be possible to highlight both the active occurrence and the active field within that occurrence.

For radio groups, checkboxes and pushbuttons the colour should not be changed when the field becomes active. In these cases the default behaviour is to enclose the label or option in a box comprised of broken lines.

3.4 THE APPLICATION DATA AREA

The application data area will contain a mixture of the following objects, which are known as “widgets” in UNIFACE terminology:-

3.4.1 Fields or Items

Fields are areas on the screen that contain variable data. Each field holds a specific piece of data, which can either be filled in by the program at run time (by reading from a data file, or being calculated), or by entered/overwritten by the user. In some cases a field may be display-only, in which case its contents may not be changed by the user.

When using the TAB key to navigate around the screen the cursor will always move to the next (or previous) field, ignoring any intervening areas.

It is also possible for a field to contain more data than is indicated by its dimensions on a particular form. This is done to conserve space. The “missing” data can be viewd by using the ZOOM function, which will open up a separate window for the contents of that field.

3.4.2 Labels

Labels are used to identify fields, and can be defined in one of the following ways:-

3.4.2.1 Form Text

This text is hard-coded into the form. Form text cannot support proportional fonts.

3.4.2.2 Label Fields

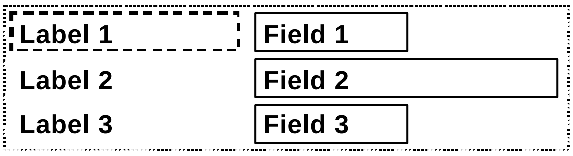
These are more versatile, and should therefore be used in preference to form text:-

⇒ These contain text that can be loaded at run time (eg: from the message file). This enables the contents of the label to be changed without modifying the form, which is extremely useful if it is ever required to supply foreign language versions of the forms.

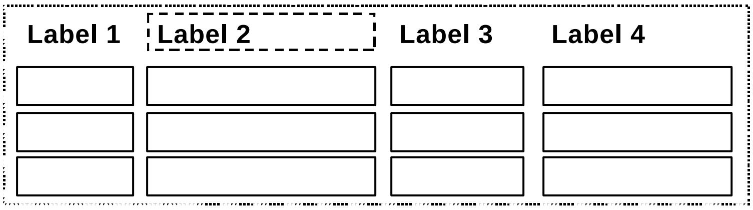
⇒ They support proportional fonts that can be changed by a setting in the .INI file.

⇒ They can be directly associated with a field so that clicking on the label is the same as giving focus to the associated field.

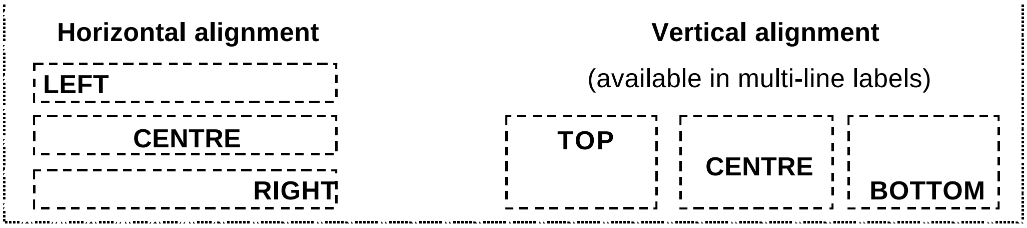
In the following example there are three fields with associated labels. The first label shows the dimensions of the field as dotted lines (not visible when the form is run) in order to show how much space is available if the contents of the field are changed.



Where there are multiple occurrences of an entity on the screen the labels should be defined only once, acting as a heading for an entire column of fields, not just a single field. Please also refer to section *3.9 Column Buttons*.



Label fields should always be defined with the maximum dimensions possible within the space available, thus allowing some leeway if the contents of the field ever need to be changed. It is also possible to change the alignment settings of each field so that the text can be positioned within the field at a point other than top left, as in the following examples:-

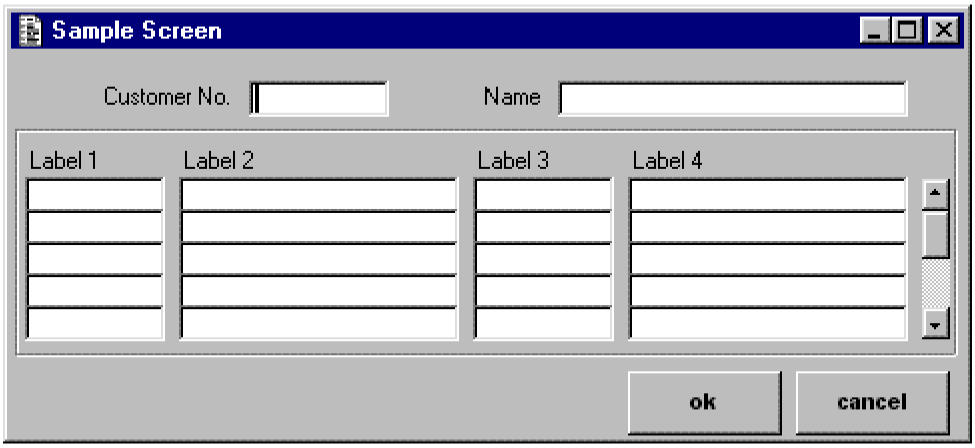


3.4.3 Borders

Borders are visible lines on the screen. They can be complete boxes, or single lines (either vertical or horizontal). Their function is purely cosmetic, to act as a separator between different areas on the screen, and as such their use is entirely optional. However, it is recommended that visible borders be used in the following circumstances:-

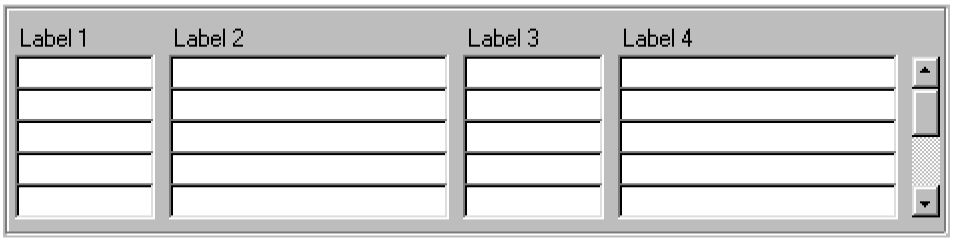
⇒ To enclose a group of radio buttons.

⇒ To enclose those fields affected by a scroll bar.



3.4.4 Scroll Bars

Where there are multiple occurrences of an entity there may not be room on the form to show all of them, so an object known as a scroll bar is utilised. This signifies to the user that there are more occurrences available, and allows him to move from one to another. A scroll bar is an oblong object with an arrow at each end (pointing in opposite directions) and a slider button between them. In the following example this is shown as a vertical bar to the right of the associated fields



If the user clicks on the ↓ button focus will move to the next occurrence, whereas the ↑ button will move focus to the previous occurrence The slider button will move up or down as an indication of the current position in the list. Alternatively the user may click onto the slider button and move it in one direction or the other. This will cause focus to jump a number of occurrences instead of just a single occurrence..

The size of the slider button can change dynamically to indicate the volume of occurrences that are available. If the slider is small the volume is large, and if the slider is large the volume is small.

If the list of available occurrences can fit into the whole of the current display then the scroll bar will become dimmed to show that clicking on it will have no effect.

Where a form contains fields from several entities, any which have scroll bars should be enclosed in a visible border (as in the above example) so that the user can easily identify which fields can have their contents changed when the scroll bar is used. The scroll bar should be either inside the border line, or on it, but not outside it.

The standard is to use a vertical scroll bar to the right of the data, but it is also possible for it to be on the left. Alternatively a horizontal scroll bar could be used, with its position being either at the top of the data or below it.

Where a form contains more than one entity with a scroll bar the form's designer should ensure that there is enough space between the entity borders so that each scroll bar is fully visible.

3.4.5 Check Boxes

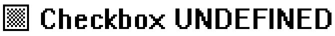
A check box is used to represent a field which has only two possible values - ON or OFF (also referred to as YES/NO or TRUE/FALSE, although any meaning can be attributed to these values). The example below shows how these values are represented when displayed on a form.

By clicking on the box (or its associated label) its contents are switched from one value to the other. The value of a checkbox can also be switched by pressing the keyboard space bar when it is the current field (signified when the label has a broken border).

☐ Checkbox OFF
☑ Checkbox ON

By default the label appears to the right of the check box, but this can be switched to the left.

UNIFACE also allows the option of check box which has a third state known as “undefined” (signified by shading, as shown below). This is known as a *tri-state checkbox*, but it is expected that the standard dual-state check box will be sufficient for most purposes.

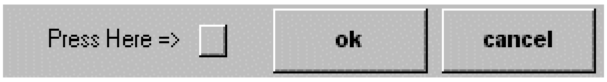


The tri-state variant can be used in cases where other data on the form causes this field to be unused or irrelevant.

3.4.6 Command Buttons

A command button (also known as a push button) is a control that is activated by clicking on it. When such a field is activated the commands that have been defined for that button are processed. The user cannot enter data into a command button.

A command button may contain text, or an image, both, or neither. The text may be taken from the message file so that it can be amended without having to change the form. The function performed by the button can either be displayed on the button face, or as a separate label, as shown in the following examples:-

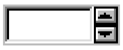


Internal text is preferred to external labels, so the button dimensions should be large enough to accommodate this text.

Command buttons will usually be contained within the Action Bar or the Navigation Bar, but there may be circumstances in which it is necessary to place a button inside the data area.

3.4.7 Spin Buttons

A spin button is an editbox with two arrow buttons next to it. It allows to user to change the value inside the editbox by pressing one of the buttons instead of overtyping the current value. The up arrow will increase the value by a predetermined amount each time it is clicked. The down arrow will decrease the value by the same amount each time it is clicked.

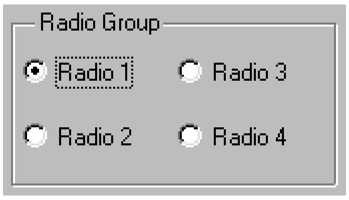


There is no upper or lower limit for the value. It will continue to be incremented or decremented as long as the user activates one of the up-down arrows.

3.4.8 Radio Buttons

Where a field may have a limited choice of values these may be represented by a group of radio buttons. The group signifies a single field, and the individual buttons signify each of the different mutually exclusive values that the field may contain. Only one of these values can be selected at any one time, and this is indicated by a black spot in the centre of the button. If the user selects a different button (by clicking on it or its associated label) the black spot is moved from the previous selection to the new selection.

The cursor control keys ← ↑ ↓ → can be used to navigate through the list of options.



Note that the radio group should be enclosed in a visible border so that the multiple options are clearly separated from other fields.

By default the label appears to the right of each button, but this can be switched to the left.

The list of options that are available within a radio group can be obtained from the same places as the contents of drop-down lists. The message file option is recommended as this will cater for the support of foreign languages.

3.4.9 Pick Lists

Where a field has a large choice of values, and this choice is too large to be represented by radio buttons, an object known as a pick-list can be used. When this field is activated the user is presented with a list of available options from which a single one must be picked. Several types of pick-list are available, as described below:-

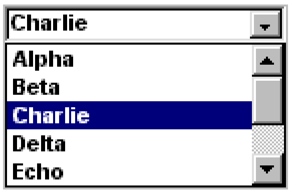
3.4.9.1 Drop-Down Lists

This type of pick-list is indicated by a button with a downward-pointing arrow on the right-hand edge of the field.

When the user activates this field, either by clicking on the field or its associated button, the list of available options will appear immediately below the target field. When the user selects an option it will be placed in the target field, and the list will disappear. If there are more options available than can be shown at any one time there will be a scroll bar attached to the list.

The cursor control keys ↑ and ↓ can be used to navigate up and down the list of entries.

If the user attempts to type into this field this starts a “speed search” for an entry which starts with the same letter as the user typed in; that is, it scrolls the list to the next entry that starts with that particular letter.



It is intended that this type of pick-list will be used under the following circumstances:-

⇒ Where the number of options is relatively small (ie: not more than 20).
⇒ Where the contents are static (ie: not liable to frequent change).
⇒ Where specific values can either be set or used by the software rather than by user action, and where any changes to the contents of the list should not be implemented without corresponding changes to the software.

The contents if this type of pick-list can be obtained in the following ways:-

1. From the Application Model. Values would be hard-coded in the UNIFACE Repository and cannot be modified by the user. Any changes to the contents of the list will require all affected forms to be re-compiled. Multi-language versions of the list would not be possible.

2. From the Message File. Values would be obtained at run time (once only, or for every reference). This allows multi-language versions. Modifications to the list cannot be made by the user, but will require the only the library file to be re-compiled, not any forms.

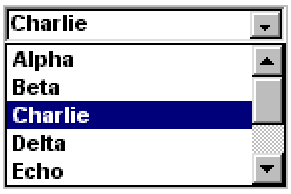
3. From the Application database. Values would be obtained at run time. This option would also requires transactions to maintain the list of options within the application datanas, as well as a function to load the database values into the list. This option should not be used where values are set by the software, or where values trigger software events.

3.4.9.2 Combo Boxes

This type of pick-list is indicated by a button with a downward-pointing arrow on the right-hand edge of the field. It is similar to a drop-down list, but allows the user to manually type in a value if the one required is not contained in the pre-defined list.

When the user activates this field, either by clicking on the field or its associated button, the list of available options will appear immediately below the target field. When the user selects an option it will be placed in the target field, and the list will disappear. If there are more options available than can be shown at any one time there will be a scroll bar attached to the list.

The cursor control keys ↑ and ↓ can be used to navigate up and down the list of entries.



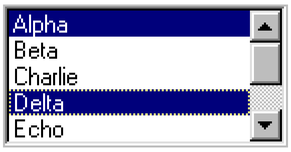
If the user types in a value instead of choosing an entry from the pre-defined list, the new value will not be added to the list unless the form contains code for that purpose.

3.4.9.3 List Boxes

A list box is a field that is displayed as a permanently open list. The user can select one or more of the items in the list. If there are more items in the list than can be visible at the same time, UNIFACE adds a scroll bar on the right-hand side of the list box.

The user cannot add new items to the list. Entering a character in the field starts a 'speed search' for an item that starts with the same letter as the user entered; that is, UNIFACE scrolls the list to the next item that starts with that letter.

The multiple-selection option may be turned off, in which case only a single value may be selected at any one time.



It is intended that this type of pick-list will be used under the following circumstances:-

⇒ Where the number of options is quite small (ie: not more than 20).

⇒ Where the contents are static (ie: not liable to frequent change).

⇒ Where multiple selections are allowed.

This type of pick-list has the following characteristics:-

⇒ Details must be hard-coded into the Application Model as the field's length must be large enough to contain all the possible values.

⇒ Changes to the contents of a list box will therefore require all effected forms to be recompiled.

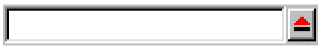
3.4.9.4 Pop-Up Lists

This type of pick-list is supplied by code specifically developed as an alternative to the drop-down list. It is indicated by a button containing an upward-pointing arrow on the field's right-hand edge.

When the user activates this field, either by double clicking on the field itself or its associated button, a separate form (which is described in more detail in a subsequent section) will be activated in order to display the list of available values.

If there are more values available than can be shown on the pop-up form at any one time there will be a scroll bar on the list. When the user selects one of the values the pop-up form will disappear, and the selected value will automatically appear in the target field.

If the user types anything into this field before clicking on the button this will be passed to the pop-up form as a retrieve profile. This will limit the display of options to only those that match the profile. For example, if the profile "AB<gold>*" (where <gold> is the character defined on your particular keyboard map) is used then only those values which begin with the letters "AB" will be included in the display. If the profile causes only one value to be retrieved then this will be selected automatically without any further user action.



It is intended that this type of pick-list will be used under the following circumstances:-

⇒ Where the number of options is quite large (ie: more than 20).

⇒ Where the contents are dynamic (ie: liable to frequent change).

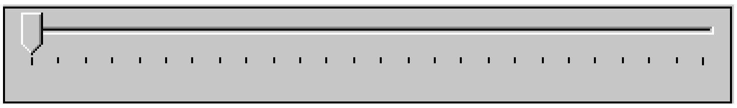
This type of pick-list has the following characteristics:-

⇒ Details will be stored within the database, therefore a maintenance function and a choose function will be required.

⇒ Changes to the contents of a popup list can be made simply by using the maintenance function, and will not therefore require any forms to be recompiled.

3.4.10 Sliders

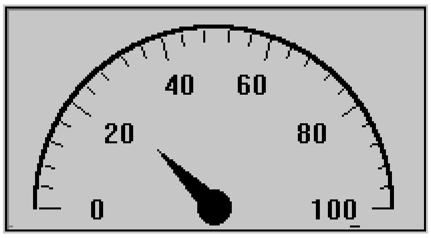
A slider provides a method of adjusting a value between pre-defined upper and lower limits. The user positions the cursor on the slider, and while holding down the mouse button moves the slider either to the left (to decrease the value) or to the right (to increase the value).



If required the slider may be presented vertically instead of horizontally.

3.4.11 Meters

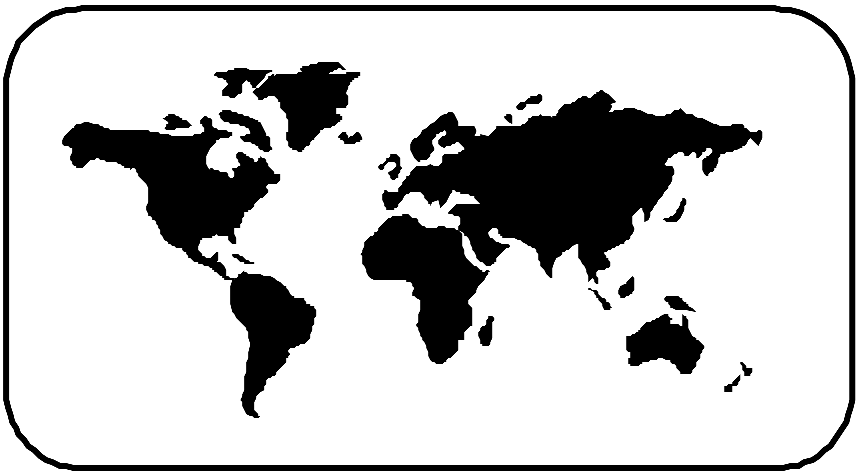
The meter provides a method of visually representing the current value of a field by setting the pointer at the appropriate position on the scale. The upper and lower limits of this scale are pre-defined. The value indicated by the pointer is display only, and cannot be changed.



By default the scale starts at 270 degrees and sweeps to 180 degrees, but both of these values can be changed. Different ranges of values on the scale can also be shown in different colours.

3.4.12 Maps

A map is an image containing any number of pre-defined arbitrary shapes, known as polygons. When the user selects a polygon the action defined for that polygon will be performed.



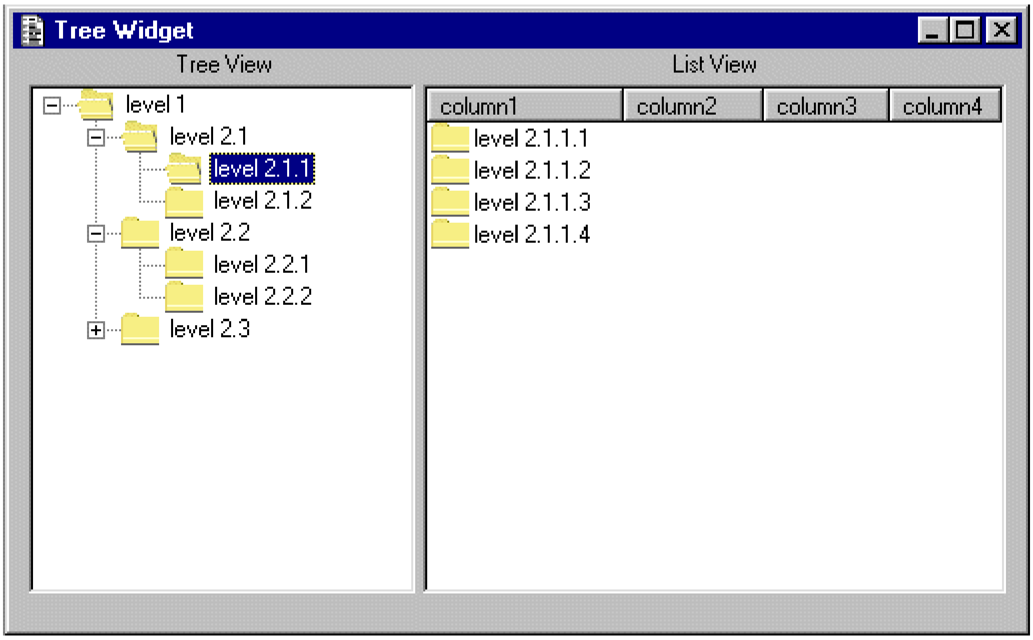
The choice of image and the size, content and number of polygons within the image is entirely at the discretion of the system designer.

The user can select a polygon by moving the cursor over it, then clicking with the left-hand mouse button. Alternatively the right-hand mouse button can be used to activate a popup menu containing a list of available polygons using their internal name tags.

Different visual feedback can be given when the cursor moves over an enabled polygon, and also when an enabled polygon is selected.

3.4.13 Trees

A tree displays data in a hierarchical structure similar to Windows Explorer. It can contain two views of the data, a Tree View and a List View. If both views are enabled the widget contains a split bar that splits the widget into these two views. Alternatively, only one of these views may be selected for display.



Each item in a tree can have the role of either a **NODE** or a **LEAF**. A node can contain child nodes and leaves. A leaf is at the end of a tree hierarchy and cannot contain either nodes or leaves. A node at the top of the hierarchy does not have a parent and is called a **ROOT** node. If the user clicks on an **Expand** (+) symbol this will cause the items one level below the current item to be included in the display. If the user clicks on the **Collapse** (-) symbol this will cause the items below the current item to be removed from the display.

The Tree View can display only Nodes. The List View can display Nodes and Leaves.

The icon against each node changes to indicate its current state:- normal, open, selected, or open & selected. Leaves can only have two states – normal and selected.

If the user single-clicks on an item in the Tree View its children will be displayed in the List View. Double clicking will cause any children to be displayed both in the List View and the Tree View.

If the user double-clicks on an item in the List View it will appear in the Tree View with the **open** icon, and will be replaced by its children in the List View.

In the List View the column labels are displayed in buttons that can be pushed to sort the data into a different sequence. This alternates between ascending and descending.

The contents of the list view may be set to any of the following:-

- **Detail** - with column headings and column data (pictured above).
- **List** - no column headings, data only for column 1.
- **Icon** – text appears below icon, items arranged horizontally as well as vertically.

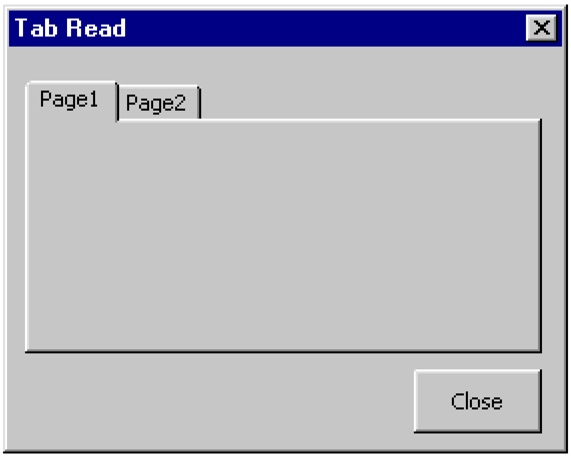
3.4.14 Tabs

A tab is a means of quickly changing the display from one tab page to another. The user clicks on a tab and the contents of the tab area changes to show the data associated with that tab. This is commonly used where the amount of data to be displayed is too large to fit onto a single page.

Within UNIFACE this is achieved by a group of forms – a parent form which contains the tab widget, and a series of child forms, one for each tab page. When the parent form is selected it immediately activates all the required child forms so that they are instantly available whenever their particular tab is selected.

It is usual for a single occurrence of an object to be selected in the parent form, and for each child form to display a different subset of data from that occurrence. Data is retrieved only in the parent form and passed to the child forms as parameters, with no duplicate database retrievals in any child form. This means that all data access is controlled by the parent form, and should not be altered in any child form.

Database updates using changes from all the tab pages are possible, but these changes must be passed back to the form that originally retrieved the data before they can be applied to the database.



The parent form may contain data fields in addition to the tab widget.

The tab labels may be presented in several rows if they cannot all fit into a single row.

1. A tab page can only be run from a form that contains a tab widget.
2. The dimensions of each tab page should not exceed those of the tab widget (minus the rows of labels) otherwise scroll bars will appear.

3.5 ACTION BUTTONS

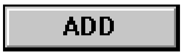
These will identify the particular actions that can be performed on the data within the current form. If an action is not identified within this area, it should follow that the action is not available within the current form.

The standard size of these buttons will be 11 characters wide by 2 characters deep. If there are more buttons than can fit onto a single line then an additional line should be created underneath.

The label text will be obtained from the message file at run-time. A more meaningful description of each button's purpose will be displayed as hint text in the message line whenever the cursor is positioned on that button, or by invoking the HELP function (see section 3.8.1).

The user will select a particular action by "clicking" on it with the mouse, or by moving the cursor to it with the TAB or BACKTAB key, then pressing the **<return>** key.

The following list is a sample only, and is not intended to be complete. The actual buttons required for a particular form will be documented within the specification for that form.

OK	This is used in a form that allows database updates to signify that all changes made within the current form are complete and should be committed to the database (provided that no validation errors are detected). The form will exit and return control to the previous form.
Cancel	In a form that allows database updates this will signify that all changes made on the current form are to be ignored. If any changes are detected the user will be given the chance to confirm this action before the changes are abandoned and control is returned to the previous form.
Close	In a form that does not allow database updates there is no need for separate OK and CANCEL buttons, therefore this button will cause the form to exit and return control to the previous form.
CLEAR	This will clear the contents of all fields on the current form, allowing fresh data to be input (or retrieved).
RETRIEVE	Will retrieve a new set of data based on the current selection criteria. This usually follows a CLEAR otherwise the current record will be retrieved again.
FIND	This is used in a SELECT form to signify that the selection criteria on the current form should be passed to the associated LIST form for processing.
Select	This is used In a POPUP form to signify the selection of an occurrence or value. The selection will then be passed back to the calling form.
 ADD	Will present the user with an empty occurrence of the relevant entity on the current form so that data for a new entry can be input. The database will not actually be updated until the OK or STORE button is pressed.
DELETE	Will delete the current occurrence of the relevant entity from the current form. The database will not actually be updated until the OK button is pressed.
STORE	This will update the database with any changes made in the current form (provided that no validation errors are detected), but will not return to the previous form. This is usually available in a form of type ADD so that the user can add another record without having to select the form again.

3.6 NAVIGATION BUTTONS

These will allow the user to activate another form within the system. If the new form is non-modal it will be possible to switch focus to an existing form in the application window, otherwise all existing forms will be suspended, and will not be re-activated until the new form is closed.

The standard size of these buttons will be 11 characters wide by 2 characters deep. If there are more buttons than can fit onto a single line then an additional line should be created alongside.

The label text will be obtained from the message file at run-time. A more meaningful description of each button's purpose will be displayed as hint text in the message line whenever the cursor is positioned on that button, or by invoking the HELP function (see section 3.8.1).

The user will select a particular action by "clicking" on it with the mouse, or by moving the cursor to it with the TAB or BACKTAB key, then pressing the **<return>** key.

It will be possible to "drill down" a number of levels, the only limiting factor being the size of each selected form and the amount of available memory within the PC. The particular navigation options that are available within each form should be determined within the design stage. It is usual to limit these options to forms dealing with data that is related to the current form - it is not usual to provide a complete list of all the other forms that are available within the system.

When a navigation button is selected for a particular form component a copy of that component is activated using an instance name, which can either be identical to the component name, or generated by incorporating the primary key of the current occurrence. The former method is normally used for forms with a dialog type of ADD or DELETE as it is not logical to have more than one copy of these forms active at any one time. The latter method is normally used for forms with a dialog type of UPDATE or ENQUIRE as the user may require to see different copies of the same form at the same time, although each would be for a different database occurrence.

Whichever method is employed, if an instance with the required name already exists in the component pool the existing instance will be re-activated. This will cause any data in the instance to be cleared before it is refreshed using the latest set of parameters. A new instance will not be created unless the name does not currently exist in the component pool.

If any data on the current screen has been modified it will be stored (subject to any validation errors) before the new child screen is activated. This will ensure that the same data as shown on the current screen will be available in the new screen, otherwise the user may be shown values that he thought he had changed. This has been known to cause a little confusion.

The following list is a sample only, and is not intended to be complete. The actual range of buttons available within a particular form will be documented within the specification for that form.

ADD	This will pass control to another form that will allow the user to create a new database occurrence.
MODIFY	This will pass control to another form that will allow the user to modify the current occurrence.
DELETE	This will pass control to another form that will allow the user to delete the current occurrence
DETAILS	This will pass control to another form that will allow the user to see more details on the current occurrence

3.7 SHORTCUT KEYS

As well as using the buttons provided within each form to perform specific actions, it is also possible to initiate the same actions by using a short-cut key, as shown in the following table:-

UNIFACE Trigger	Button Name	shortcut key
<accept>	OK	Ctrl-A
<clear>	Clear	Ctrl-G
---	Close	Ctrl-A, Ctrl-Q, or the Close button
<detail>	⏏	Ctrl-D
<help>	---	Ctrl-H
message frame	---	Ctrl-M
<quit>	Cancel	Ctrl-Q, or the Close button
<retrieve>	Retrieve	Ctrl-R
<store>	Store	Ctrl-S
<zoom>	---	Ctrl-Z

These short-cut keys may vary according to the specifications within your current keyboard map and/or definitions within the .INI file

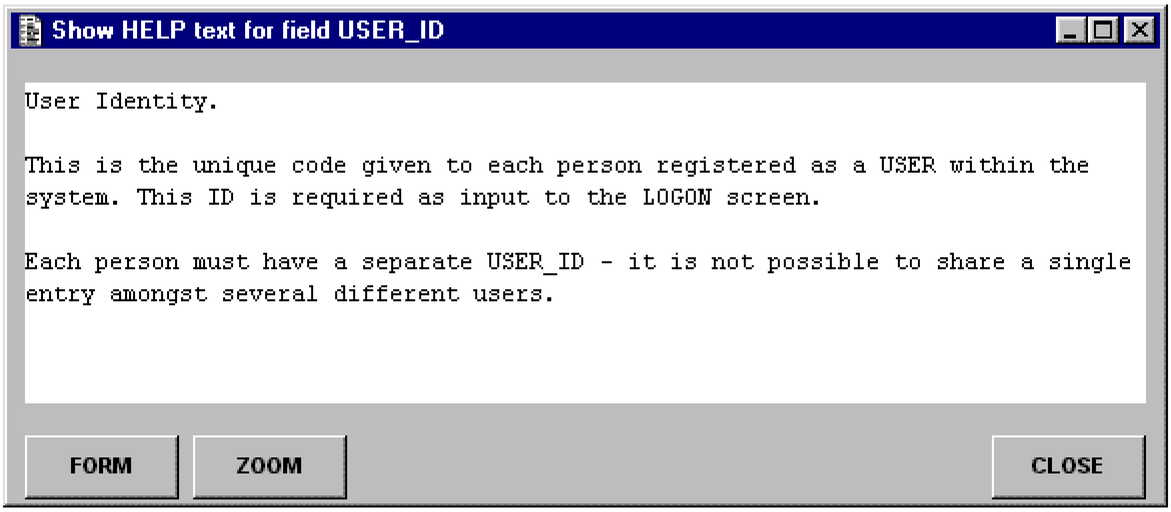
These alternative options are provided as we cannot predict how individuals will wish to interface with the system - some users prefer the mouse, while others are more at home with the keyboard.

The UNIFACE triggers are only identified here for those who are familiar with this terminology.

3.8 HELP OPTIONS

3.8.1 Show Help

This screen is invoked from the pulldown menu (see section 3.2.1), or by using the relevant shortcut key to activate the <help> trigger (see section 3.7). When selected a screen similar to the following will be displayed:-



User Identity.

This is the unique code given to each person registered as a USER within the system. This ID is required as input to the LOGON screen.

Each person must have a separate USER_ID - it is not possible to share a single entry amongst several different users.

This function will extract text from the application database and display it in the area provided. This text will either be for the current FIELD (on the calling form) or for the calling FORM itself. When first invoked the screen will show the text for the current field.

The form title will identify the object (field or form) for which help text is currently being displayed. The left-hand pushbutton can be used to switch to the other object, and its label will change accordingly.

There may be more lines of text available than can be viewed within the initial area provided, in which case ZOOM can be invoked to enlarge this area.

It should be noted that we can only provide the mechanism for displaying help text - the creation of this text is the responsibility of the client as the general user will want explanations expressed in business terms that are familiar to him. Such detailed knowledge is usually in the domain of the business analyst, not the software provider.

3.8.1.1 HTML Help

It is now possible to provide help text from HTML files rather than the application database. This feature can be turned on for individual applications running under the Menu system rather than globally for all applications.

To turn this feature ON for an application you must add the following entry into the [logicals] section of the assignment file:

```
<variation>_HTMLHELP=<directory>
```

where <variation> is the application (library) name
and <directory> is the location of the directory containing the HTML files.

For example, if the HTML files for the MENU system are located in directory Htmlhelp\Menu (relative to the current working directory) then the following entry should be inserted into the assignment file:

```
menu_htmlhelp=htmlhelp\menu\
```

When the SHOW HELP operation is invoked it will first look for the existence of **<variation>_htmlhelp** in the assignment file by substituting the current value of $variation for <variation>. If this entry does not exist then help text will continue to be obtained from the database and displayed using the form in section 3.8.1.

If this entry does exist then it will activate the Windows API “ShellExecute” on a file name which is constructed as follows:

<directory> + $componentname + “.htm”

If this fails it will try again with the “**.html**” extension.

If this fails again it will display an error message.

The Windows API “ShellExecute” performs the same operation as double-clicking on the file name within Windows. It runs the application which has been assigned to that file’s extension as recorded within the Windows registry. In this case it should be a web browser.

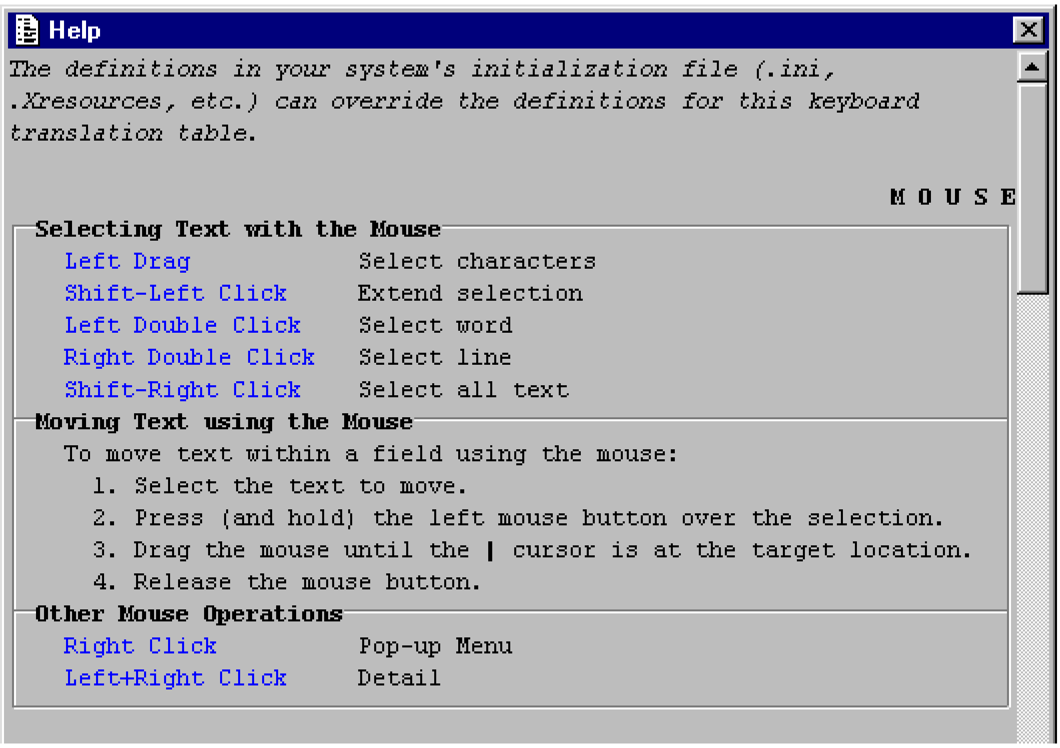
In order for the various Windows APIs to be executed the following line must be inserted into the [userdlls] section of the .INI file:

```
demandload=c:\windows\system\advapi32.dll,c:\windows\system\shell32.dll,c:\windows\system\kernel32.dll
```

where **C:\windows\system** identifies the directory where these system files are located.

3.8.2 Keyboard Help

This screen is invoked from the pulldown menu (see section 3.2.1). When selected a screen similar to the following will be displayed:-



The definitions in your system's initialization file (.ini, .Xresources, etc.) can override the definitions for this keyboard translation table.

M O U S E

Selecting Text with the Mouse

Left Drag	Select characters
Shift-Left Click	Extend selection
Left Double Click	Select word
Right Double Click	Select line
Shift-Right Click	Select all text

Moving Text using the Mouse

To move text within a field using the mouse:

1. Select the text to move.
2. Press (and hold) the left mouse button over the selection.
3. Drag the mouse until the | cursor is at the target location.
4. Release the mouse button.

Other Mouse Operations

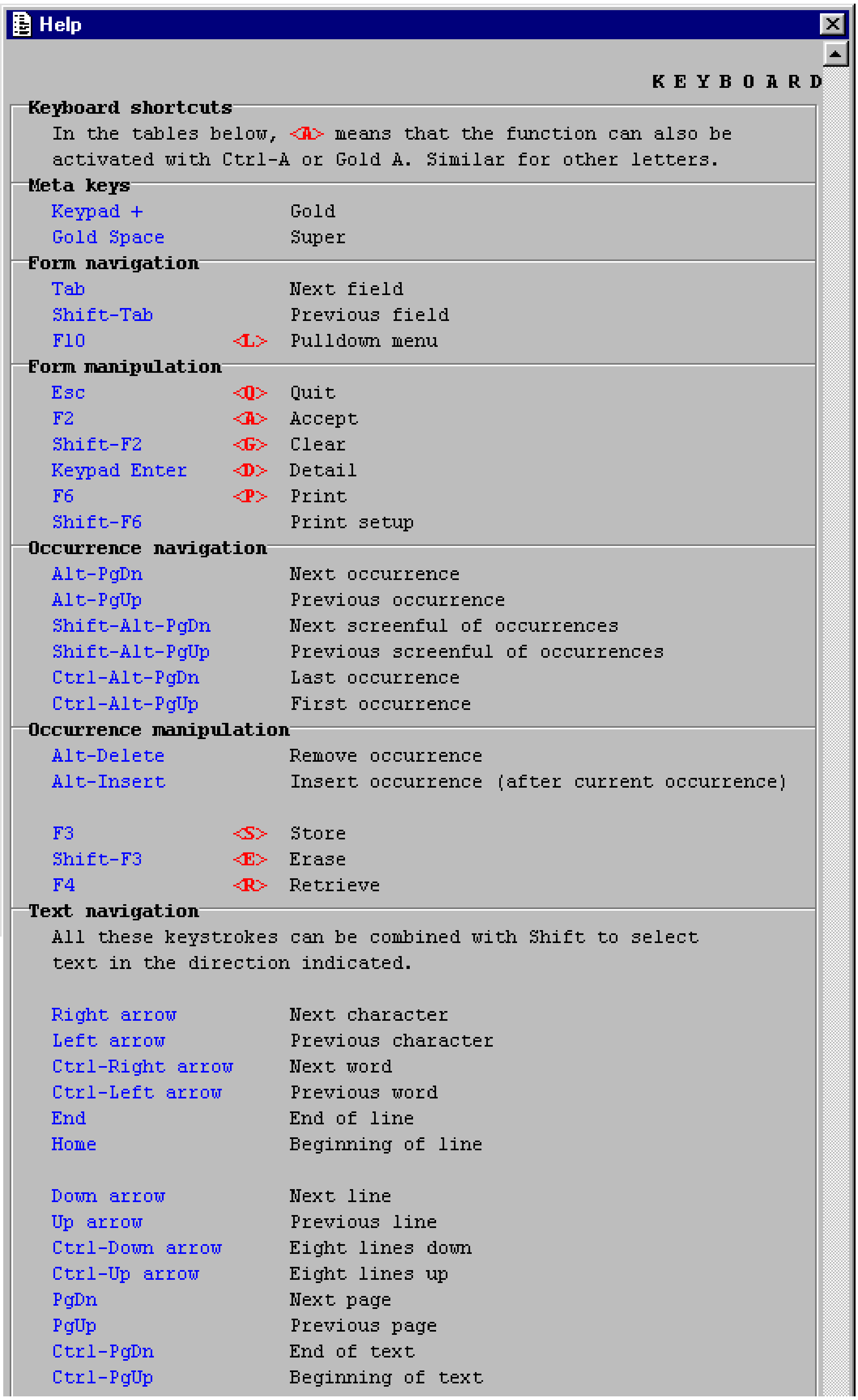
Right Click	Pop-up Menu
Left+Right Click	Detail

This screen contains more lines of text than can be displayed at a time, therefore a scroll bar is available on the right-hand side to move the viewing window either up or down.

The first section of this screen shows the functions that can be performed with the mouse, while the second section (contained on the following page) shows the functions that can be performed with the keyboard.

It should be noted that this keyboard map could change depending on the physical device being used at the time. It is also possible for the keyboard map to be customised further to suit individual tastes or requirements, therefore the information given here should be treated as an example only.

Here is part 1 of the keyboard map:-



KEYBOARD

Keyboard shortcuts

In the tables below, <A> means that the function can also be activated with Ctrl-A or Gold A. Similar for other letters.

Meta keys

Keypad +		Gold
Gold Space		Super

Form navigation

Tab		Next field
Shift-Tab		Previous field
F10	<L>	Pulldown menu

Form manipulation

Esc	<Q>	Quit
F2	<A>	Accept
Shift-F2	<G>	Clear
Keypad Enter	<D>	Detail
F6	<P>	Print
Shift-F6		Print setup

Occurrence navigation

Alt-PgDn		Next occurrence
Alt-PgUp		Previous occurrence
Shift-Alt-PgDn		Next screenful of occurrences
Shift-Alt-PgUp		Previous screenful of occurrences
Ctrl-Alt-PgDn		Last occurrence
Ctrl-Alt-PgUp		First occurrence

Occurrence manipulation

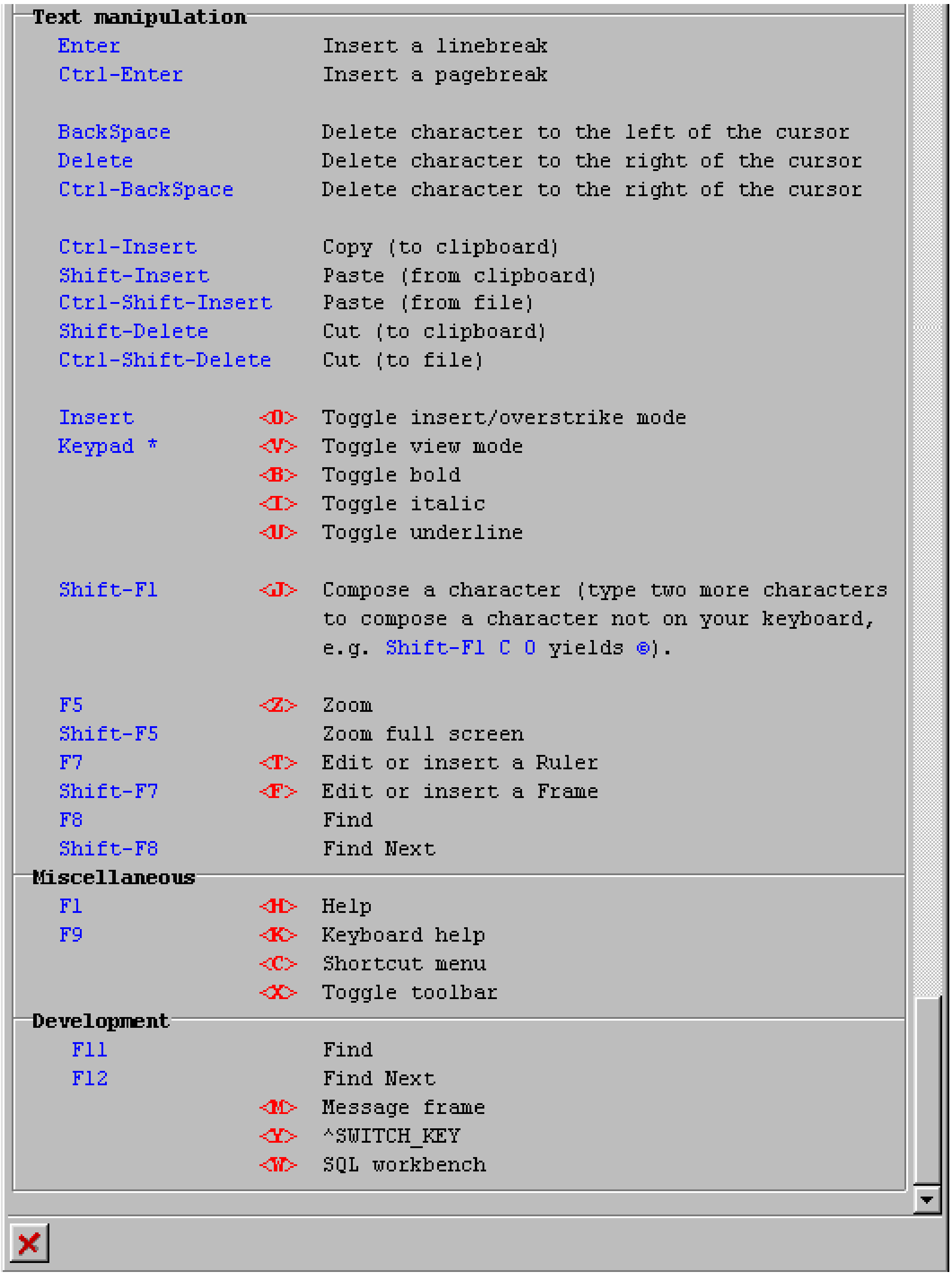
Alt-Delete		Remove occurrence
Alt-Insert		Insert occurrence (after current occurrence)
F3	<S>	Store
Shift-F3	<E>	Erase
F4	<R>	Retrieve

Text navigation

All these keystrokes can be combined with Shift to select text in the direction indicated.

Right arrow	Next character
Left arrow	Previous character
Ctrl-Right arrow	Next word
Ctrl-Left arrow	Previous word
End	End of line
Home	Beginning of line
Down arrow	Next line
Up arrow	Previous line
Ctrl-Down arrow	Eight lines down
Ctrl-Up arrow	Eight lines up
PgDn	Next page
PgUp	Previous page
Ctrl-PgDn	End of text
Ctrl-PgUp	Beginning of text

Here is part 2 of the keyboard map:-



Text manipulation

Enter		Insert a linebreak
Ctrl-Enter		Insert a pagebreak
BackSpace		Delete character to the left of the cursor
Delete		Delete character to the right of the cursor
Ctrl-BackSpace		Delete character to the right of the cursor
Ctrl-Insert		Copy (to clipboard)
Shift-Insert		Paste (from clipboard)
Ctrl-Shift-Insert		Paste (from file)
Shift-Delete		Cut (to clipboard)
Ctrl-Shift-Delete		Cut (to file)
Insert	<O>	Toggle insert/overstrike mode
Keypad *	<V>	Toggle view mode
	<B>	Toggle bold
	<I>	Toggle italic
	<U>	Toggle underline
Shift-F1	<J>	Compose a character (type two more characters to compose a character not on your keyboard, e.g. Shift-F1 C O yields ©).
F5	<Z>	Zoom
Shift-F5		Zoom full screen
F7	<T>	Edit or insert a Ruler
Shift-F7	<F>	Edit or insert a Frame
F8		Find
Shift-F8		Find Next

Miscellaneous

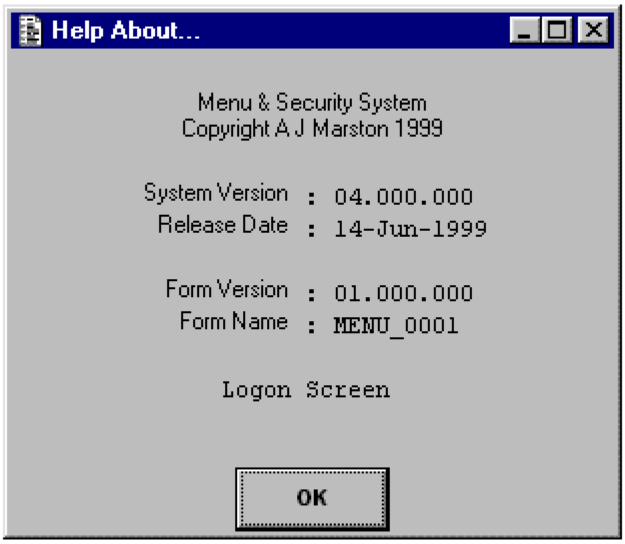
F1	<H>	Help
F9	<K>	Keyboard help
	<C>	Shortcut menu
	<X>	Toggle toolbar

Development

F11		Find
F12		Find Next
	<M>	Message frame
	<Y>	^SWITCH_KEY
	<W>	SQL workbench

3.8.3 Help About ...

This screen is invoked from the pulldown menu (see section 3.2.1). When selected a screen similar to the following will be displayed:-



Menu & Security System
Copyright A J Marston 1999

System Version : 04.000.000
Release Date : 14-Jun-1999

Form Version : 01.000.000
Form Name : MENU_0001

Logon Screen

Each application that runs under the menu system should have its own version of this form. The top line shows the application name (Menu & Security System in this example) followed by the system version number and release date. The last three lines show the version number, name and title of the current form.

A system (or application) is comprised of many modules (or forms, in UNIFACE terminology). These are usually made available (or released) to the client in batches, either when a group of new modules have been developed and tested, or when existing modules have been updated.

The format of the version number is usually **MM.ccc.bbb** where

MM	is the major number
ccc	is the count of user-initiated (minor) changes or enhancements
bbb	is the count of bug fixes

When a module is initially developed its version number is set to 01.000.000.

If a bug is detected, either during link/system testing or user-acceptance testing, the **bbb** portion of the version number should be incremented by 1 when the bug is fixed. No other portion of the version number should be changed.

If the user requests a minor change then the **ccc** portion of the version number should be incremented by 1 when the change is implemented. No other portion of the version number should be changed.

If the user requests a major change (usually accompanied by a completely new version of the module specification) then the **MM** portion of the version number should be incremented by 1, and the other portions should be reset to zero.

When a collection of modules is ready to be released to the user, then the system version number and release date should be changed accordingly.

3.9 COLUMN BUTTONS

It is a convention in some programs that where a screen contains rows of occurrences the user can click on any column label to sort the display using the values contained in that column. By clicking repeatedly on the button the sort will automatically switch from ascending sequence to descending sequence.

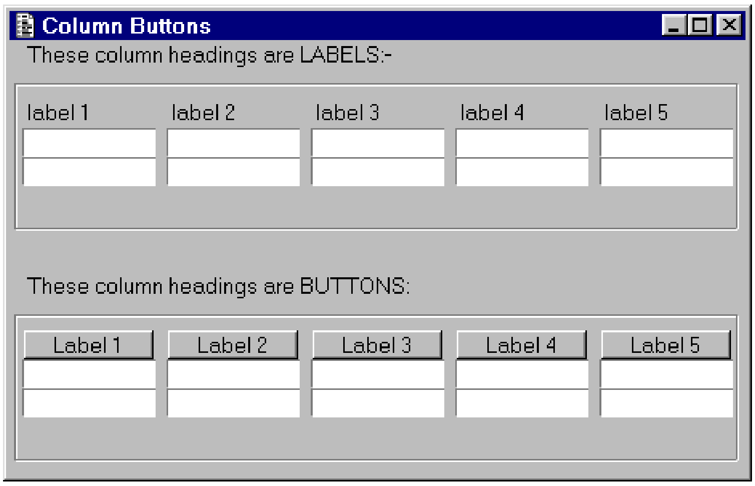
This feature cannot be provided with UNIFACE labels as it is not possible to define any processing that should take place if the user clicks on the label.

One method in which this functionality can be simulated is by using the *Sort* option on the standard pulldown menu, which presents the user with an *ascending* and *descending* option. This requires that the user place the cursor in the relevant field before selecting the pulldown option. This has the advantage that it provides the ability to sort by columns without having to change the labels in any form.

The only way to provide the ability to sort a column by clicking on its label is to change the label into a button. This then provides a <detail> trigger into which the relevant code can be placed. This will have to detect the current sort sequence for that column, then issue a *sort* command to reverse the sequence.

The hitlist must be complete before it can be sorted. If a stepped hitlist contains any unfetched steps these must be retrieved from the database beforehand. On large hitlists this may cause a delay before the *sort* operation can be completed.

The visual difference between labels and buttons is shown in the following screen shot:



These column headings are LABELS:-

These column headings are BUTTONS:

4. STANDARD DIALOG TYPES

This section identifies the standard dialog types which should be used as the basis for constructing each component within the system. The standard behaviour of each dialog type is described in detail, so when the component specifications are being prepared for the development team it should be sufficient to simply refer to the dialog type by name rather than duplicate its details. This also means that once the users have become familiar with a particular dialog type they should have no difficulty in using any function which is based on the same dialog type.

Each of these dialog types has been established as a Component Template in the development environment. Component templates contain objects known as generic entities and generic fields which correspond to the major objects of the form. The templates also contain generic code which will provide the compiled form with the basic behaviour of that dialog type.

Where a dialog type is designed for a single entity the entity name is hown as “MAIN”. Where it deals with a ONE to MANY relationship then the names “ONE” and “MANY” will be used. Field names will be shown either as “FIELD” where a single field is expected, or “FIRST” and “LAST” to indicate the first/last fields in the prompting sequence. Any number of other fields can be inserted between FIRST and LAST, including those on “up” entities if required.

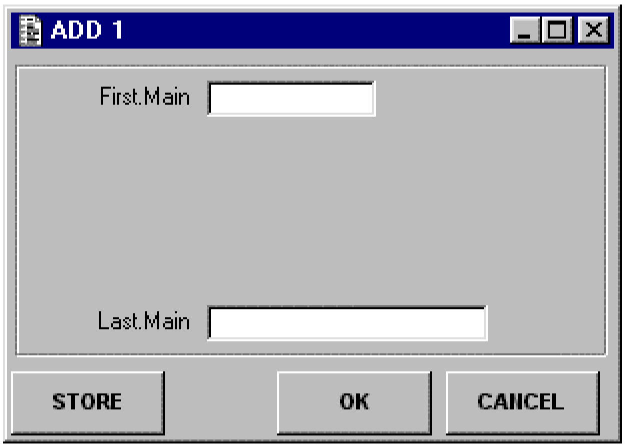
When a component is created from a template the generic objects in the template must be bound (mapped) to the actual objects required in that component. These objects should already be defined within the application model. Any default trigger code defined in the template will override any default code in the application model. Once all the relevant objects are mapped the component structure is created, and the developer can then insert fields and/or proc code as required.

Apart from the ability to quickly create a component with the necessary default behaviour the use of component templates within UNIFACE has an additional advantage - any default proc code defined in any component-level trigger will automatically be inherited by the component whenever it is compiled. It is therefore possible to modify trigger code within a single component template, then recompile all components associated with that template to have those changes automatically inherited by those components. This inheritance is not carried forward into any entity triggers or field triggers. Any changes to these must be made manualy within each affected component.

Note that this list of dialog types may not be definitive. Circumstances may require new or variant dialog types, but these should only be implemented after careful thought, and where an existing dialog type cannot be used.

4.1 ADD / CREATE

4.1.1 Add 1



This type of form is used to create a new occurrence of an entity. Only one occurrence can be stored with each operation of this form.

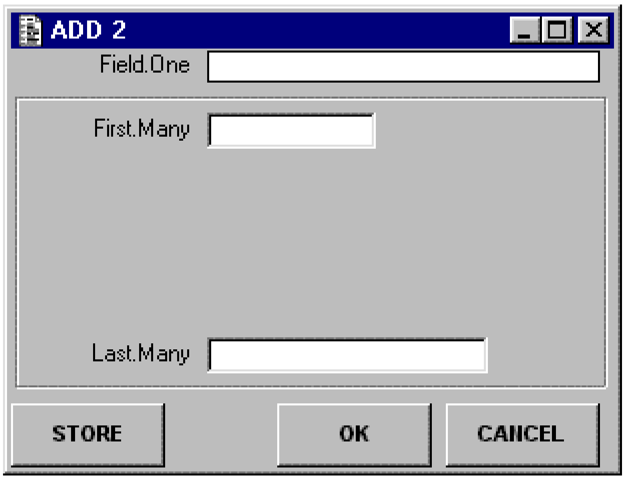
Upon initial entry the user is presented with a blank screen, although initial values may have been defined for some fields. The user then enters the required data, then presses either the OK button or the STORE button to update the database.

The parent form will be informed of the addition, and may update its display accordingly.

The active field will be highlighted in a different colour.

STORE	Will add the current entry to the database and clear the screen, allowing details for another occurrence to be entered.
OK	Will add the entry to the database and return to the parent form.
Cancel	Will abandon the current entry and return to the parent form.

4.1.2 Add 2



This type of function is used in a ONE-to-MANY relationship in order to create a new occurrence of the MANY entity. Only one occurrence can be stored with each operation of this form.

Upon initial entry the function will retrieve an occurrence of the ONE entity using the primary key passed down by the parent form. The remainder of the screen will be blank, although initial values may have been defined for some fields The user then enters the required data, then presses either the OK button or the STORE button to update the database.

The parent form will be informed of the addition, and may update its display accordingly.

The active field will be highlighted in a different colour.

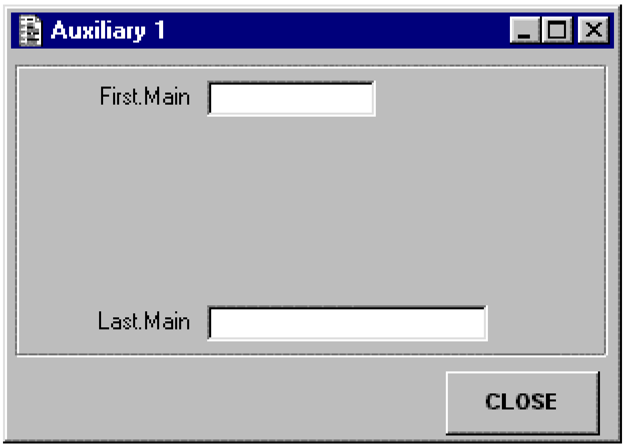
STORE	Will add the current entry to the database and clear the screen, allowing details for another occurrence to be entered.
OK	Will add the entry to the database and return to the parent form.
Cancel	Will abandon the current entry and return to the parent form.

4.2 AUXILIARY

This type of form is used where data has been read into, or must be written from a single form, but where there is so much data it cannot be handled in a single screen. In this case an overflow or auxiliary screen is required. All database access is performed within the original form (the parent), with the necessary data being passed to an auxiliary (the child) as arguments on the **activate** statement. The auxiliary form may just display the data, or allow it to be amended before passing it back to the parent form.

This type of form cannot be called directly from a menu - it can only be called from a designated parent form which has the correct parameter specifications in its **activate** statement.

4.2.1 Auxiliary 1



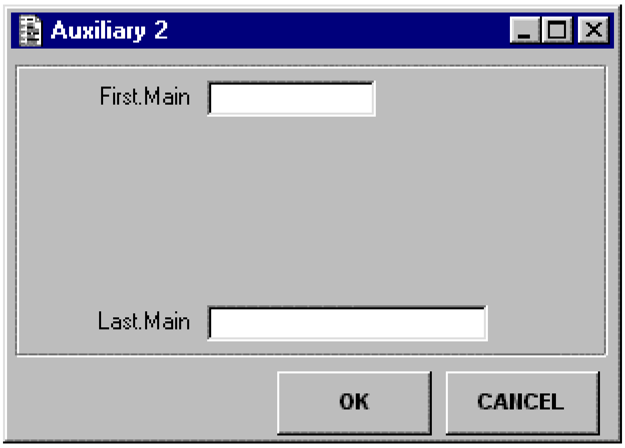
This type of form will display the current values for the occurrence(s) passed to it by the parent form, and allow the user to view them before returning to the parent form.

The values displayed will be those passed down from the parent form, and not retrieved from the database (except for the retrieval of foreign entities).

The active field will be highlighted in a different colour.

Close Will return to the parent form.

4.2.2 Auxiliary 2



A typical use of this type of form is in a ONE to MANY relationship where the ONE and the first occurrence of the MANY have to be created at the same time, but require separate screens. All the database updates must be controlled in the parent form - it would be a violation of database integrity of an occurrence of the MANY entity were to be applied to the database before the occurrence of the ONE.

This type of form will display the current values for the occurrence(s) passed to it by the parent form, and allow the user to modify them before returning to the parent form.

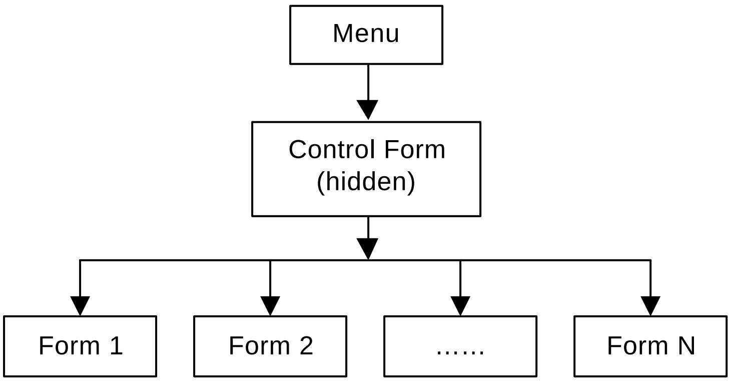
The values displayed will be those passed down from the parent form, and not retrieved from the database (except for the retrieval of foreign entities). Any changed values will be passed back to the parent form, and not written to the database within this form.

The active field will be highlighted in a different colour.

OK	Will return to the parent form with any values that have changed. These can only be committed to the database within the parent form.
Cancel	Will return to the parent form without passing any changed values.

4.3 CONTROL

There may be occasions when the data for a logical transaction needs to be assembled from several screens before being stored in the database. This situation calls for a Control form, or Controller. All the data is held within this control form, which is hidden, but subsets are made available to the user via a series of visible forms, typically of the Auxiliary dialog type. The structure can be represented in the following diagram:



The identities of the child forms to be processed are defined in a list within the Controller. This list is processed in sequence. As each form terminates the Controller activates the next form in the list. After the last form has been processed the Controller will perform the action specified by the **$auto_store$** option (see below).

Within the Controller is the option to specify additional processing both before and after the activation of each child form. This will allow the data to be modified, or even to alter the contents of the list of child forms.

Each child form may have the following action buttons:

OK	Will pass the data back to the Controller, which will then activate the next form in the sequence.
Cancel	Will cancel the entire sequence and terminate the transaction without updating the database.
PREVIOUS	Will go back to the previous form in the sequence.

The following options can be specified in *Extra Parameters* in the transaction definition for the Controller on the Menu database:-

$auto_store$	The possible values are “Y” or “N” (the default is “N”). If set to “N” the user will be asked to REVIEW or STORE the data. ▪ If STORE is chosen the current data will be stored and the Controller will terminate. ▪ If REVIEW is chosen then the form sequence will be reactivated from the beginning, allowing the data to be reviewed and modified as necessary. If set to “Y” the data will be stored automatically without the option to review, and the Controller will terminate.

4.3.1 Control 1

This creates an occurrence of the MAIN entity without requiring the identities of any foreign entities to be pre-selected. Upon initial entry a blank occurrence of MAIN is created, then loaded with any initial values defined for the controller and the child forms. The first child form in the sequence is then activated.

4.3.2 Control 2

This creates an occurrence of MANY in a ONE-to-MANY relationship where the identity of ONE is passed in as a parameter. Upon initial entry the specified occurrence of the ONE entity is retrieved, then a blank occurrence of the MANY entity is created. This is then loaded with any initial values defined for the controller and the child forms before the first child form in the sequence is activated.

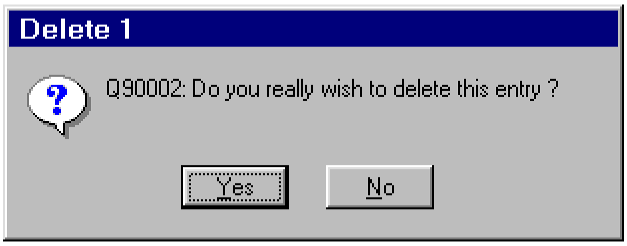
4.3.3 Control 3

This modifies the details for a selected occurrence where the identity of the selected occurrence is passed in as a parameter. The details are retrieved from the database before the first child form in the sequence is activated.

4.4 DELETE

This type of function is used to delete a single occurrence of an entity. Only one occurrence can be deleted with each operation of this function, although there may be numerous occurrences from subordinate entities.

4.4.1 Delete 1



Q90002: Do you really wish to delete this entry ?

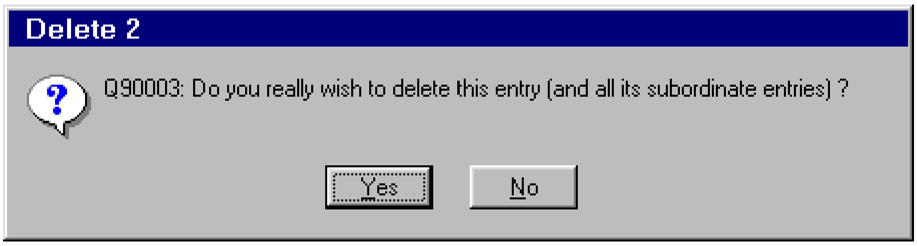
Upon initial entry this form will perform an automatic retrieve of the specified occurrence using the primary key value passed down from the parent form.

There will be no screen to display - instead the user will be presented with a simple dialog box which asks that the deletion be confirmed. The user is given the choice of two buttons.

If the specified occurrence has subordinate entries on other entities within the database the deletion will be rejected, with a suitable error message. All subordinate entries will have to be deleted using a separate form before the specified occurrence can be deleted.

Yes	Will delete the specified entry from the database, then return to the parent form indicating that the deletion has been completed.
No	Will return to the parent form without deleting the entry from the database.

4.4.2 Delete 2



Q90003: Do you really wish to delete this entry (and all its subordinate entries) ?

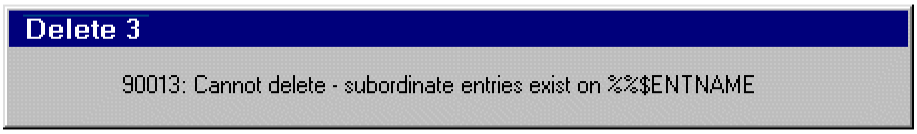
Upon initial entry this form will perform an automatic retrieve of the specified occurrence using the primary key value passed down from the parent form.

There will be no screen to display - instead the user will be presented with a simple dialog box which asks that the deletion be confirmed. The user is given the choice of two buttons.

If the specified occurrence has subordinate entries on other parts of the database then these entries will also be deleted.

Yes	Will delete the specified entry (and all of its subordinate entries) from the database, then return to the parent form indicating that the deletion has been completed.
No	Will return to the parent form without deleting the entry from the database.

4.4.3 Delete 3



90013: Cannot delete - subordinate entries exist on %%$ENTNAME

This type of form is used in conjunction with the DELETE button on the Action Bar of a Multi-Purpose transaction. It does not actually delete anything from the database, it just verifies that the specified occurrence can be deleted.

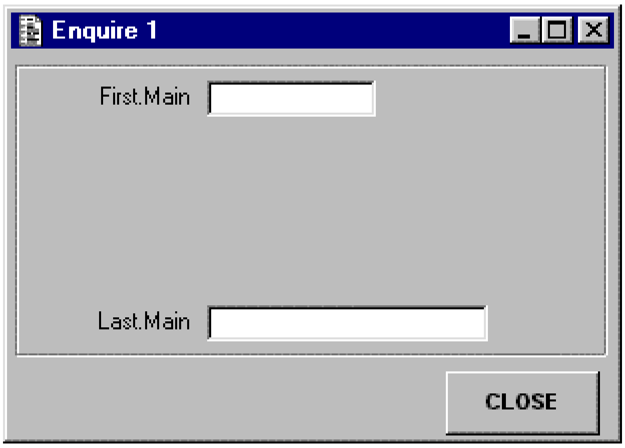
Upon entry this function will perform an automatic retrieve of the specified occurrence using the primary key value passed down from the parent form. It will then look for subordinate entries that may exist on the database.

If the specified occurrence has any subordinate entries then an error message will be displayed in the message line, and the parent form will not delete the occurrence.

If the occurrence has no subordinate entries then its deletion will be permitted.

4.5 ENQUIRE / READ / DISPLAY

4.5.1 Enquire 1

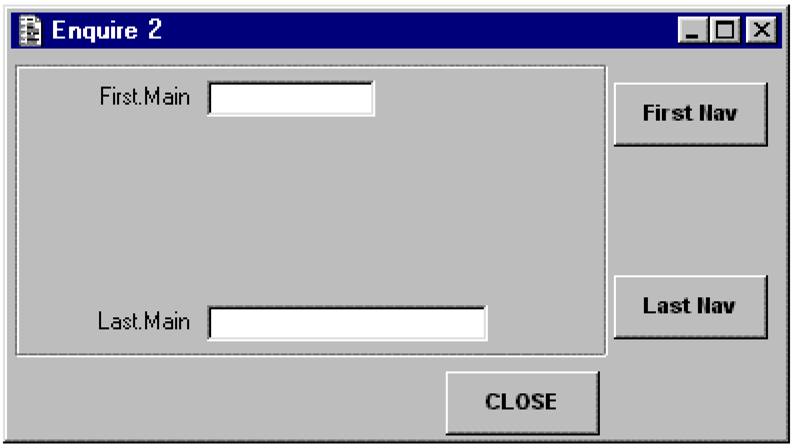


This type of form is used to display the contents of a specified database occurrence. Only one occurrence can be displayed with each operation of this form.

Upon initial entry there will be an automatic retrieve of the MAIN entity using the primary key value passed down from the parent form.

Will return to the parent form.

4.5.2 Enquire 2



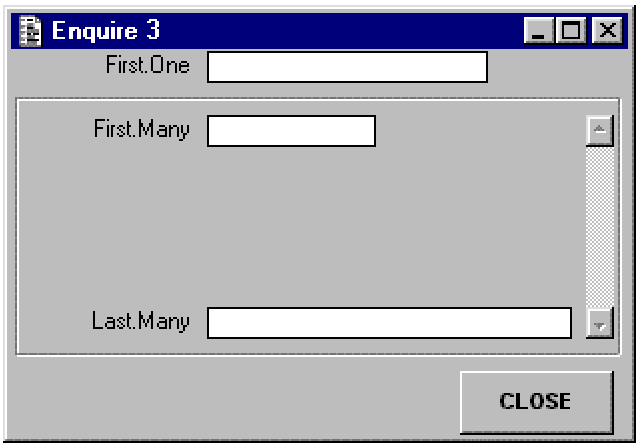
This type of form is used to display the contents of a specified occurrence. Only one occurrence can be displayed with each operation of this form.

Upon initial entry there will be an automatic retrieve of the MAIN entity using the primary key value passed down from the parent form.

Close	Will return to the parent form.
First Nav Last Nav	Each of these navigation buttons will pass control to another form in the system, passing as a parameter the primary key of the current occurrence.

If this form is called from a navigation bar, then it may be a better idea if the navigation options specified here were to be included in the parent's navigation bar. This will avoid the problem of having too many processes stacked on top of each other.

4.5.3 Enquire 3

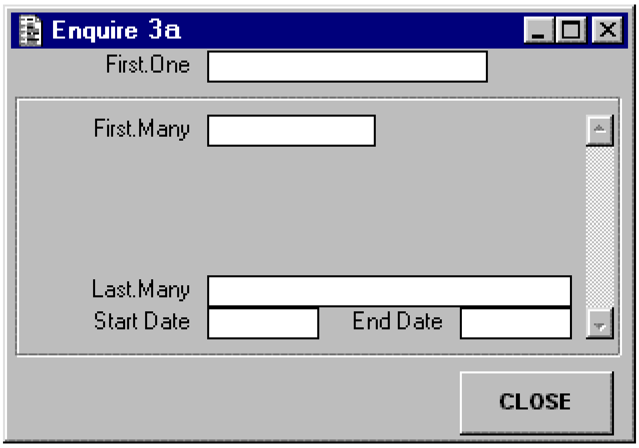


This type of form is used to display a series of occurrences in a ONE-to-MANY relationship where the the identity of the ONE entity is passed down from the parent form. Only one occurrence of the ONE entity can be displayed with each operation of this form.

Upon initial entry there will be an automatic retrieve of the ONE entity using the primary key value passed down from the parent form. This will be followed by the retrieval of the first available occurrence of the MANY entity. The scroll bar can be used to examine other occurrences of the MANY entity.

Close Will return to the parent form.

4.5.4 Enquire 3a

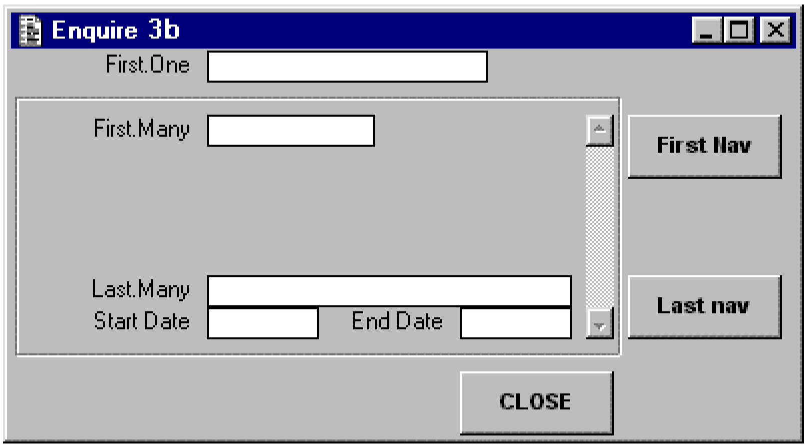


This type of form is used to display a series of occurrences in a ONE-to-MANY relationship where the the identity of the ONE entity is passed down from the parent form. Only one occurrence of the ONE entity can be displayed with each operation of this form. The MANY entity contains a START and END date that allows only one occurrence to be valid for any particular date.

Upon initial entry there will be an automatic retrieve of the ONE entity using the primary key value passed down from the parent form. All occurrences of the MANY entity will be retrieved, but the occurrence whose START and END dates cover a period which contains the current system date will be displayed first. The scroll bar can be used to examine other occurrences with earlier or later date ranges.

Close	Will return to the parent form.

4.5.5 Enquire 3b



This type of form is used to display a series of occurrences in a ONE-to-MANY relationship where the the identity of the ONE entity is passed down from the parent form. Only one occurrence of the ONE entity can be displayed with each operation of this form. The MANY entity contains a START and END date that allows only one occurrence to be valid for any particular date.

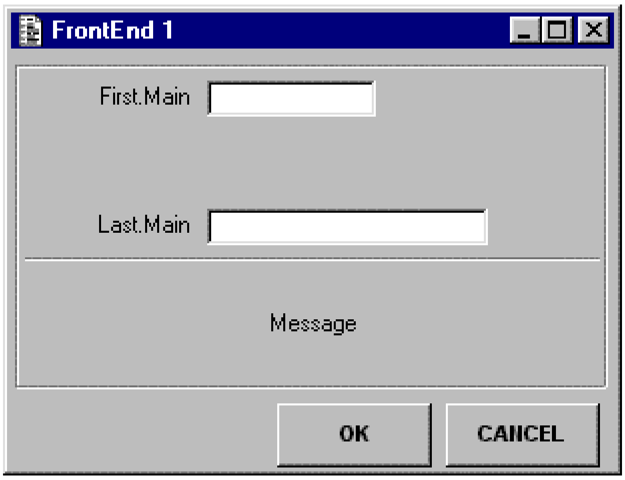
Upon initial entry there will be an automatic retrieve of the ONE entity using the primary key value passed down from the parent form. All occurrences of the MANY entity will be retrieved, but the occurrence whose START and END dates cover a period which contains the current system date will be displayed first. The scroll bar can be used to examine other occurrences with earlier or later date ranges.

Close	Will return to the parent form.
First Nav Last Nav	Each of these navigation buttons will pass control to another form in the system, passing as a parameter the primary key of the current occurrence.

4.6 FRONTEND

There may be transactions within the system that perform bulk processing, either for reporting purposes or to update large numbers of records on the database. These functions may perform their actions on a remote device so as to avoid passing great volumes of data over the network, thus avoiding a potential performance bottleneck. These functions do not have any dialog with the user during their processing, but they do require a front-end screen in order to be activated. This screen may also be used for the input of any parameters that the function may require.

4.6.1 FrontEnd 1



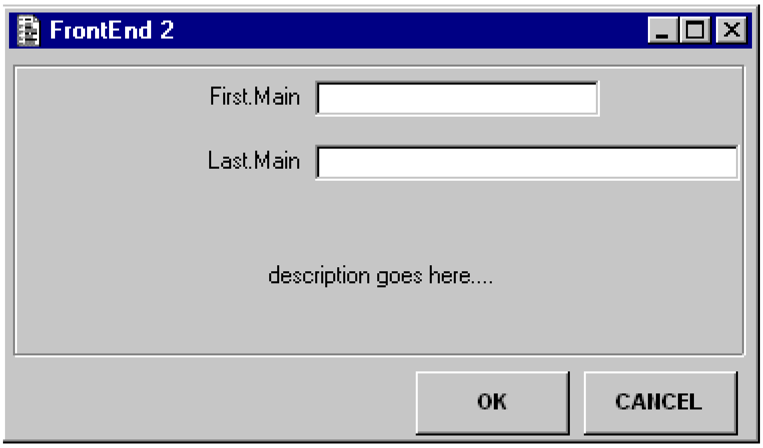
This type of form is used to initiate a function which requires to process a high volume of data on the database. The message area will indicate the type of processing that will take place.

This version does not expect any parameters to be passed down from the parent form, therefore will not attempt to perform an automatic retrieve.

The user will be able to add in any selection parameters that may be required, then press the OK button to perform the designated processing via a child form. After the child form has completed its processing it will return control to this form, which will immediately pass control back to its parent form.

OK	Will activate a child form to carry out the designated processing, after which it will return to the parent form.
Cancel	Will return to the parent form without performing any processing.

4.6.2 FrontEnd 2



description goes here....

This type of form is used to initiate a function which requires to process a high volume of data on the database. The message area will indicate the type of processing that will take place.

This version expects the primary key of a database occurrence to be passed down from the parent form so that it can perform an automatic retrieve..

Upon initial entry any parameters passed down from the parent form will be processed. If the user presses the OK button then the designated processing will be performed by activating a child form. After the child form has completed its processing it will return control to this form, which will immediately pass control back to its parent form.

OK	Will activate a child form to carry out the designated processing, after which it will return to the parent form.
Cancel	Will return to the parent form without performing any processing.

4.7 HIDDEN / SERVICES

These are called “hidden” forms because they have no dialog with the user, and are called *service* components rather than *form* components. They are called from other functions in order to perform a service. They will usually require parameters to be passed from the parent to identify the object on which they should perform their activity, and will usually have a result to pass back.

This type of form can be constructed as a self-contained service so that it may be run on a remote device, thus reducing the need to transmit data over the network unnecessarily.

4.7.1 Hidden 1

This type of function does not update the database, it is read-only. It will usually have only one operation (the <exec> trigger).

4.7.2 Hidden 2

This type of function updates the database with a COMMIT. It will usually have only one operation (the <exec> trigger).

4.7.3 Hidden 3

This type of function updates the database, but without a COMMIT as this must be left to the parent form. Please refer to *UNIFACE Development Guidelines Part 2 (Internals)* on the section titled *Logical Updates Across Multiple Forms* for more details.

4.7.4 Hidden 4

It is common for a hidden form to be invoked for a single iteration of its designated function, after which it terminates and ceases to exist in the component pool.

With UNIFACE version 7 it is now possible to activate a component and begin processing from any operation defined within that form. Control can be returned to the parent component after the completion of each operation without the component being terminated and without the loss of any data. This allows the component to be activated again to perform additional processing. Each activation can be accompanied by its own set of input and output parameters.

A component may contain any number of operations, and each operation may be activated any number of times. Provided that the operation ends with a **return** statement and not an **exit** it will continue to exist.

The following operation names are reserved words:-

INIT	Invoked automatically when the component is initiated.
EXEC	Activates the <exec> trigger. Note that completion of this operation will cause the component to terminate (**return** is treated the same as **exit**).
ACCEPT	Activates the <accept> trigger.
QUIT	Activates the <quit> trigger.
CLEANUP	Invoked automatically when the component is terminated.

4.8 LIST / BROWSE

This type of function is used to display multiple occurrences of an entity. It shows summary details only, one entry per line. It does not allow any updates, but a series of navigation buttons provide access to other functions that should be able to perform whatever processing is necessary.

As the number of database occurrences that may be retrieved within this type of function is potentially very large the RDBMS may employ a stepped hitlist. If a child form that is launched attempts to construct another hitlist on the same database table there will be a conflict. To deal with this situation the following options are available:

- **a) DEFAULT -** If the hitlist in the parent process is incomplete (ie: there are entries that match the current retrieve profile that have yet to be retrieved from the database) then all unfetched occurrences will be retrieved. NOTE THAT FOR LARGE DATABASE TABLES THIS COULD RESULT IN A SIGNIFICANT DELAY.
- **b) DROP HITLIST -** Drop the hitlist at the current point for the named entity in the parent process. If the current hitlist is incomplete then all unfetched occurrences will be dropped and will no longer be available in the current operation - the user will have to perform a <clear> followed by a <retrieve> in order to build a new hitlist.
- **c) NEW DATABASE PATH -** Open up another database path for the child process, thus avoiding any conflicts. NOTE THAT THIS WILL RESULT IN MULTIPLE DATABASE OPENS FOR A SINGLE USER.

The required option can be defined on a form-by-form basis – please refer to the section titled *Child Properties* in the *Menu & Security System User Guide* for more details.

Specifying a read limit for large hitlists

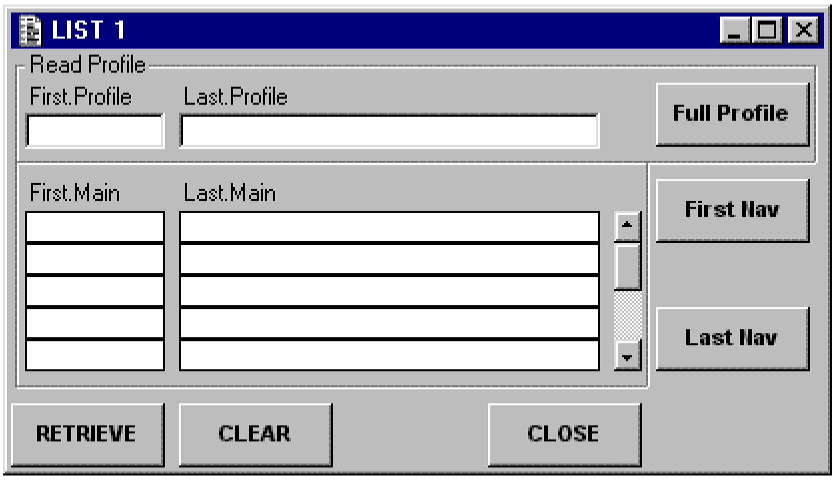
To avoid the possibility of a session being suspended for a significant period of time due to the retrieval of a large number of occurrences, these functions include code to interrupt the retrieve after a certain number of entries have been read into the form's structure. The actual number can be set using the **$$read_limit=n** parameter in the assignment file. Each time the limit is reached the user will be presented with a dialog box requiring a YES or NO response. If NO is selected the retrieve will be terminated at this point, and all remaining entries in the hitlist will be dropped. If YES is selected the retrieve will continue until the limit is reached again.

Additional functionality available with non-modal forms

A LIST form is usually the parent in a group of forms known as a forms family. Please refer to section *2.3.3.5 Modal and Non-Modal Forms* for more details.

When selecting occurrences in the parent form it is possible to (a) create a new instance of a child form to process that selection, or (b) refresh an existing instance with the new selection. Please refer to section *2.3.3.6 Interaction between Non-Modal Forms* for more details.

4.8.1 List 1



Upon initial entry this form has the following options (determined by a value in *Extra Parameters* in the transaction definition on the Menu database):-
⇒ Perform an automatic retrieve using a null profile, thus including all available database entries.
⇒ Do not perform an automatic retrieve, but present the user with a blank screen.

This function may be run as a child from another function, in which case parameters for the profile area may be passed down from the parent function.

The Profile is limited to fields that are contained on the outer (main) entity.

Prompt sequence: If no data has been retrieved the prompt sequence will be Profile area then Action bar, excluding the Data area and Navigation bar. If data has been retrieved then the prompt sequence will be Data area, Navigation bar, then Action bar, excluding the Profile area.

The current occurrence will be highlighted in a different colour.

The following options can be specified in *Extra Parameters* in the transaction definition on the Menu database:-

$auto_select$=Y/N	Will determine if an automatic retrieve is performed on initial entry.
$select_form$=*name*	The form that will be activated when the *Full Profile* button is pressed. NULL will signify that no form is available and the button will be hidden.

If more selection criteria are required than can fit into the profile area, this button will pass control to a separate form (see dialog type SELECT 1) which will allow a far wider range of selection criteria to be entered. Upon returning to this form an automatic retrieve will be performed using the entered selection criteria.

Will retrieve entries from the database using whatever retrieve profile has been defined. If the profile is blank then all available entries will be retrieved. If no qualifying entries can be found then a warning message will be displayed.

If an additional retrieve is performed without using the CLEAR button the results of this retrieve will be appended to the current screen contents. The existing hitlist will be completed so that any duplicate entries can be identified and removed.

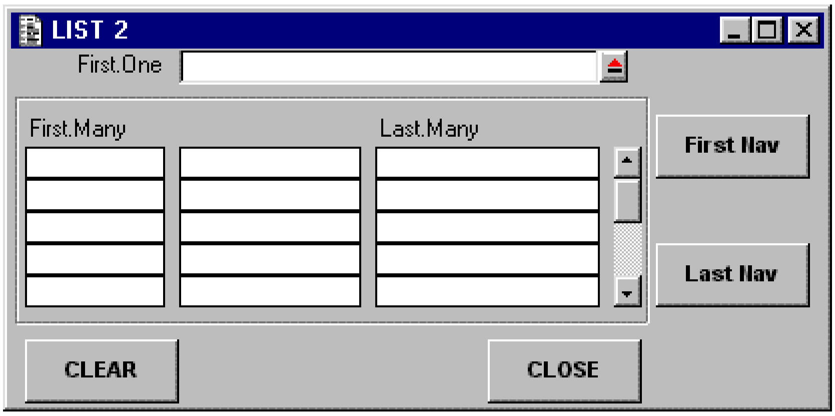
This will clear the contents of all fields on the current screen and drop the current hit list, allowing a different profile to be defined in the area provided at the top of the screen before attempting another retrieve.

Will return to the parent form.

Each of these navigation buttons will pass control to another form in the system, passing as a parameter the primary key of the current (highlighted) occurrence.

- If an ADD function is called this will cause the new database occurrence to be inserted into the screen display after the current occurrence.
- If a MODIFY function is called this will cause the screen display to be updated to reflect any changed values.
- If a DELETE function is called this will cause the relevant occurrence to be removed from the current screen display.

4.8.2 List 2



This type of function is used in a ONE-to-MANY relationship where only those occurrences of MANY that are related to occurrence ONE can be displayed.

The action performed upon initial entry depends on the existence of a parameter being passed from the parent form:-

- If a parameter is passed an occurrence of the ONE entity will be retrieved, followed by any associated occurrences of the MANY entity. The cursor will be positioned on the first occurrence of the MANY entity.
- If no parameter is passed the function will not be able to perform an automatic retrieve. The cursor will be positioned on the first field of the ONE entity, and the user must select an occurrence (using a popup) before any occurrences of the MANY entity can be retrieved.

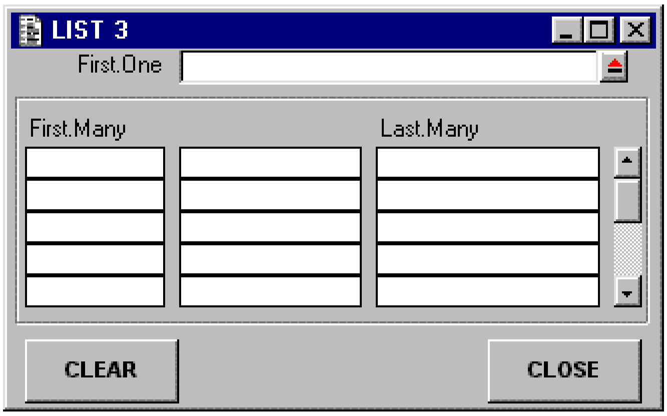
The active field/occurrence will be highlighted in a different colour.

The following options can be specified in *Extra Parameters* in the transaction definition on the Menu database:-

$change_allowed$	The possible values are “Y” or “N” (the default is “N”). If set to “N” the popup button will not appear and the field on the ONE entity will be display only. This will prevent the user from changing the value passed down by the parent form. If a value is not passed down by the parent form the function will immediately exit with a suitable error message.

CLEAR	This will clear the contents of all fields on the current screen, allowing a different occurrence of the ONE entity to be selected.
Close	Will return to the parent form.
First Nav Last Nav	Each of these navigation buttons will pass control to another form in the system, passing as a parameter the primary key of the current (highlighted) occurrence. • If an ADD function is called this will cause the new database occurrence to be inserted into the screen display after the current occurrence. • If a MODIFY function is called this will cause the screen display to be updated to reflect any changed values. • If a DELETE function is called this will cause the relevant occurrence to be removed from the current screen display.

4.8.3 List 3



This type of function is used in a ONE-to-MANY relationship where only those occurrences of MANY that are related to occurrence ONE can be displayed.

The action performed upon initial entry depends on the existence of a parameter being passed from the parent form:-

- If a parameter is passed an occurrence of the ONE entity will be retrieved, followed by any associated occurrences of the MANY entity. The cursor will be positioned on the first occurrence of the MANY entity.
- If no parameter is passed the function will not be able to perform an automatic retrieve. The cursor will be positioned on the first field of the ONE entity, and the user must select an occurrence (using a popup) before any occurrences of the MANY entity can be retrieved.

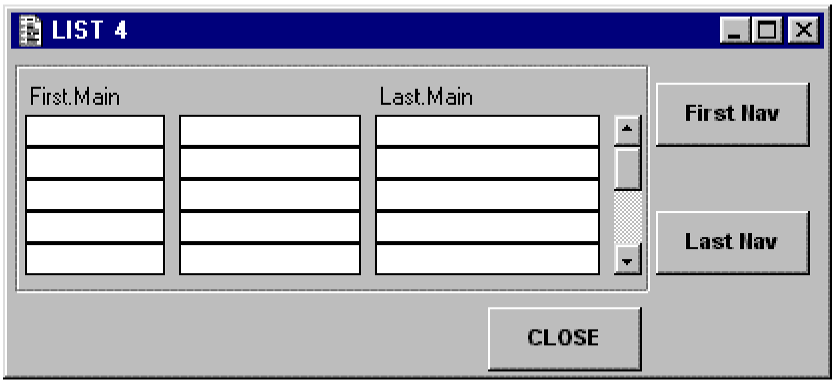
The active field/occurrence will be highlighted in a different colour.

The following options can be specified in *Extra Parameters* in the transaction definition on the Menu database:-

$change_allowed$	The possible values are “Y” or “N” (the default is “N”). If set to “N” the popup button will not appear and the field on the ONE entity will be display only. This will prevent the user from changing the value passed down by the parent form. If a value is not passed down by the parent form the function will immediately exit with a suitable error message.

CLEAR	This will clear the contents of all fields on the current screen, allowing a different occurrence of the ONE entity to be selected.
Close	Will return to the parent form.

4.8.4 List 4



This function is similar to LIST 1, but does not contain any selection criteria as this is handled by the parent form (see SELECT 1). It is also capable of receiving a hitlist of occurrences which have been pre-retrieved in the parent form (see SELECT 2).

If just the profile (selection criteria) is passed down from the parent form, then upon initial entry this function will perform an automatic retrieve using that profile. If this is a null string (ie: the previous form was a menu), then all available entries will be selected. If no entries are found a warning message will be issued.

If a hitlist of entries is passed down from the parent form then these will be displayed after performing any sort as specified in **$sort_spec$** (see below).

The active field/occurrence will be highlighted in a different colour.

Close Will return to the parent form.

First Nav
Last Nav

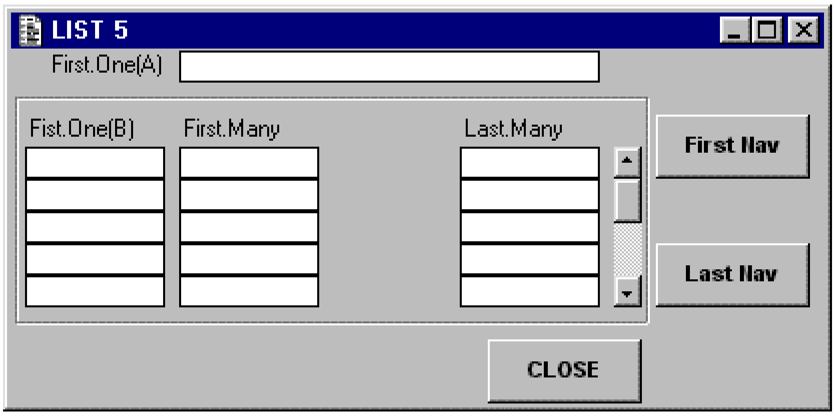
Each of these navigation buttons will pass control to another form in the system, passing as a parameter the primary key of the current (highlighted) occurrence.

- If an ADD function is called this will cause the new database occurrence to be inserted into the screen display after the current occurrence.
- If a MODIFY function is called this will cause the screen display to be updated to reflect any changed values.
- If a DELETE function is called this will cause the relevant occurrence to be removed from the current screen display.

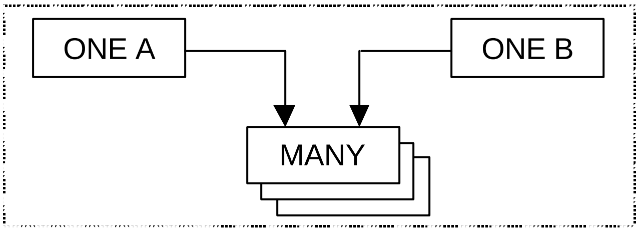
The following options can be specified in *Extra Parameters* in the transaction definition on the Menu database:-

$sort_spec$	A list of field names separated by commas. By default the sort will be performed in ascending sequence, but this can be altered by appending the keyword ':desc' to any fieldname, as in the following example: field1:desc,field2:asc,field3,field4

4.8.5 List 5



This type of function is used in a ONE(a)-to-MANY-to-ONE(b) relationship where it is possible to have multiple occurrences of MANY for each combination of ONE(a) and ONE(b), as shown in the following diagram:



There may be a different occurrence of MANY covering a different date range, for example. In this form it is required that only one occurrence of MANY be shown (either the first or the last). An option on the navigation bar may provide the ability to view the remaining occurrences of MANY. It is possible for an occurrence of MANY not to exist yet, in which case the values for ONE(a) and ONE(b) will still be shown, with the values for MANY being blank. An option on the navigation bar may provide the ability to create occurrences of MANY.

Upon initial entry the function will retrieve an occurrence of the ONE(a) entity using the primary key passed down by the parent form, then retrieve all associated occurrences of the ONE(b) entity, followed by the MANY entity.

The active field/occurrence will be highlighted in a different colour.

Close Will return to the parent form.

First Nav
Last Nav

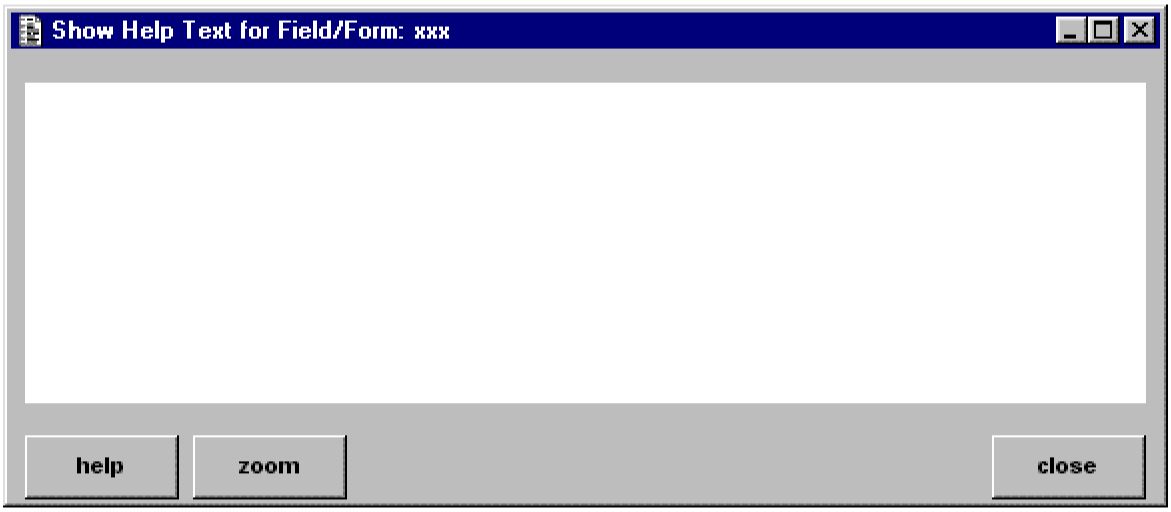
Each of these navigation buttons will pass control to another form in the system, passing as a parameter the primary key of the current (highlighted) occurrence.

- If an ADD function is called this will cause the new database occurrence to be inserted into the screen display after the current occurrence.
- If a MODIFY function is called this will cause the screen display to be updated to reflect any changed values.
- If a DELETE function is called this will cause the relevant occurrence to be removed from the current screen display.

4.9 MISCELLANEOUS

It is envisaged that each application will need one copy of each of the following forms. They are included here so that a common pattern can be adopted across different applications.

4.9.1 Help - Show Help



This form is activated from the pulldown menu (see section 3.2.1), or by using the relevant shortcut key to activate the <help> trigger (see section 3.7).

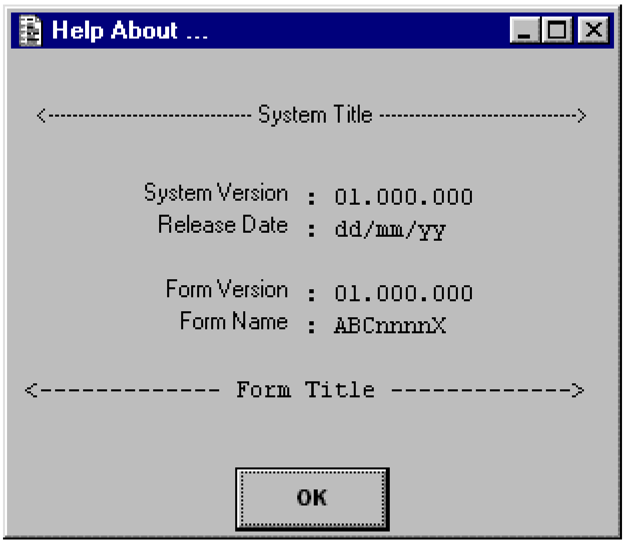
This form will extract help text from the application database and display it in the area provided. This text will either be for the current FIELD (on the calling form) or for the calling FORM itself. The screen title will identify the object (field or form) for which help text is currently being displayed.

When first activated the screen will show the text for the current field.

This form is read only. The text is maintained using function HELPM (see below).

Field/Form	The button label will be different from the object (FIELD or FORM) for which text is currently being displayed. When selected both the text and label will change to the other object.
Zoom	This button will enlarge the text area for those instances where there may be more lines of text available than can be viewed within the initial area provided.
Close	Will exit and return control to the parent function.

4.9.2 Helpa - Help About



<------------------------------- System Title ------------------------------->

System Version : 01.000.000
Release Date : dd/mm/yy

Form Version : 01.000.000
Form Name : ABCnnnnX

<------------- Form Title ------------->

This form is invoked from the pulldown menu (see section 3.2.1). It operates as described in section 3.8.3. It is used to identify which version of the software is being run.

SYSTEM TITLE, SYSTEM VERSION and RELEASE DATE refer to the application as a whole.

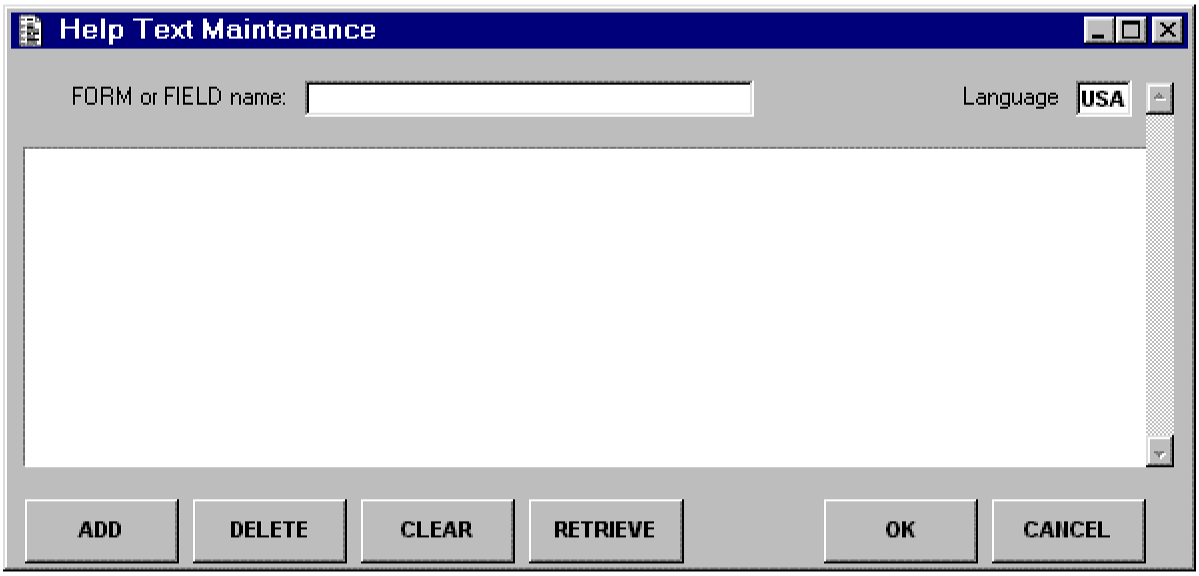
SYSTEM VERSION and RELEASE DATE should be updated each time a new version of the software is released.

FORM VERSION, FORM NAME and FORM TITLE refer to the individual component that was active at the time this option was invoked.

FORM VERSION should be updated each time a change is made to the form component.

OK	Will exit and return control to the parent form.

4.9.3 Helpm - Help Text Maintenance



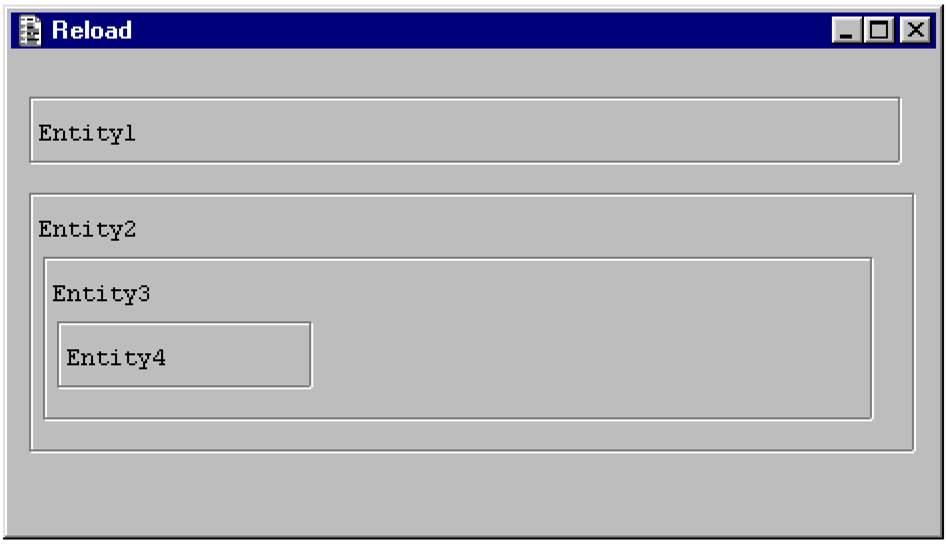
Language USA

This form is used to maintain the help text that will be displayed when the HELP form is activated.

Each application should have its own copy of the HELP table in its database so that it can be physically separated from the help text for other applications. This should avoid those situations where a field (column) with the same name has different meanings in different applications.

ADD	Will present the user with an empty occurrence so that a new entry may be input. The database will not be updated until the OK button is pressed.
DELETE	Will delete the current occurrence from the screen. The database will not be updated until the OK button is pressed.
RETRIEVE	Will retrieve a new set of entries from the database using the current retrieve profile. If the profile is blank then all available entries will be retrieved. If no qualifying entries can be found then a warning message will be displayed.
CLEAR	This will clear the contents of all fields on the current screen, allowing a different profile to be defined in the area provided before attempting another retrieve.
OK	Will commit all changes to the database (provided that all validation is successful), then exit and return control to the parent form.
Cancel	Will abandon all changes and return control immediately to the parent form.

4.9.4 Reload - Reload Data from TXT files



This form, and its companion UNLOAD form, will probably not be used in the finished application, but they may prove useful during the development phase for the unloading and reloading of data following a modification to the database. These forms can also be used to re-instate the database to a known condition so that tests can be repeated on the same set of data. An additional use would be the loading of initial data during system installation.

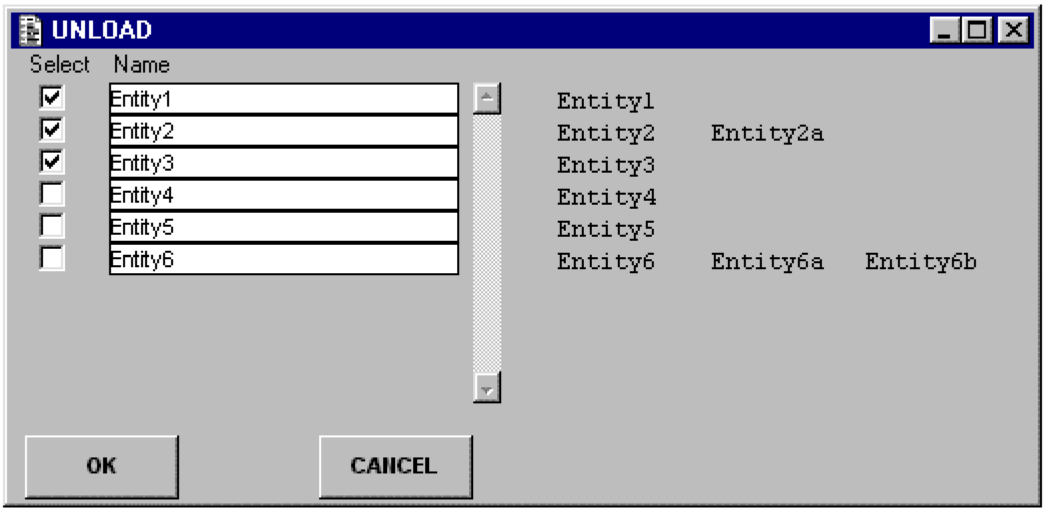
This form can only be used to reload data from TXT files that were created by the UNLOAD function described below. The hierarchy of entities should be the same as those defined in the UNLOAD form so that data can be reloaded in the same sequence as it was unloaded.

The identity of the file to be loaded is obtained from a standard dialog box which shows the user a selection of file names from which a selection can be made. This is repeated until the CLOSE button is selected.

For each input file the sequence of events is as follows:-
- Extract the data for a single record from the input file.
- Extract the entity (table) name, and ensure it exists within the external structure. This means that data extracted from one application cannot be accidentally loaded into another application (unless they share both entity names and field names).
- Test if a record already exists with the primary key supplied - if it does then delete it and all its subordinate entries. Create a new record using the data from the input file.
- Repeat until all data from the input file has been processed.

Note that where an entity is defined as an outer entity in the structure (eg: Entity1 and Entity2 in the above example) it is not affected unless the input file contains a replacement. Any inner entities (eg: Entity3 and Entity4 in the above example) which have their parent (outer) entity replaced will be deleted when the parent is deleted, and may or may not be replaced, depending on the contents of the input file.

4.9.5 Unload - Unload Data to TXT Files



☑ Entity1
☑ Entity2
☑ Entity3
☐ Entity4
☐ Entity5
☐ Entity6

```
Entity1
Entity2    Entity2a
Entity3
Entity4
Entity5
Entity6    Entity6a    Entity6b
```

This form is designed to be used only with its companion RELOAD form, as described in the previous section. It creates files in TXT format (which can be viewed and modified using any text editor) rather than in UNIFACE TRX format.

The contents of each database occurrence will be written as a string terminated by a carriage return. Each string will be in the following format:-

ENTITY=<entityname>;field1=value1;field2=value2;.....;fieldn=valuen

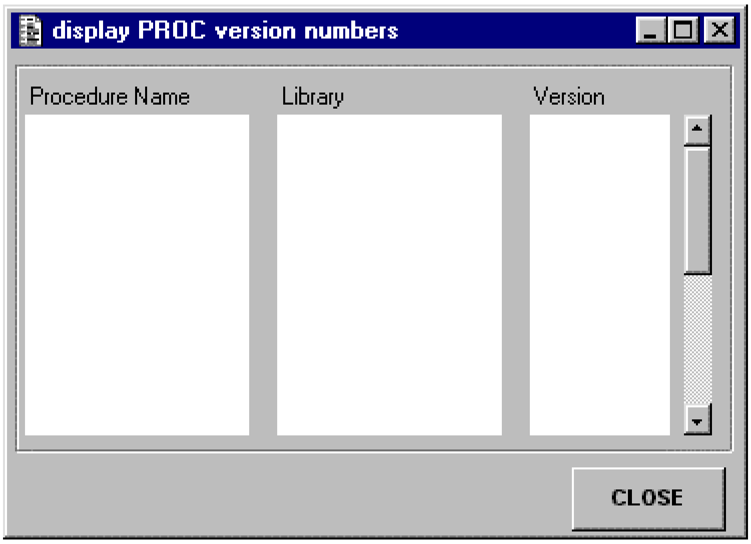
where

<entityname>	is the name of the actual entity (outer or inner)
field1,2,n	is the name of a field (column) belonging to that entity
value1,2,n	is the value relating to that field
;	is a separator (semi-colon prefixed with the UNIFACE <GOLD> character)

The SELECT area contains names of the outer entities only. Inner entities are automatically unloaded immediately after their parent.

OK	Will extract all selected entities (tables) before returning to the parent form. The contents of each outer entity (and all related inner entities) will be written to a separate file, with the same name as the outer entity.
Cancel	Will exit back to the parent form.

4.9.6 Version - Display Proc Version Numbers



This form is used to display the contents of the procedure library being accessed (usually from a DOL file) to verify that the correct version is being referenced.

PROC NAME identifies the procedure name.

PROC LIBRARY identifies the name of the library containing that procedure, which will either be the library created for that particular application, or the default library (SYSTEM_LIBRARY).

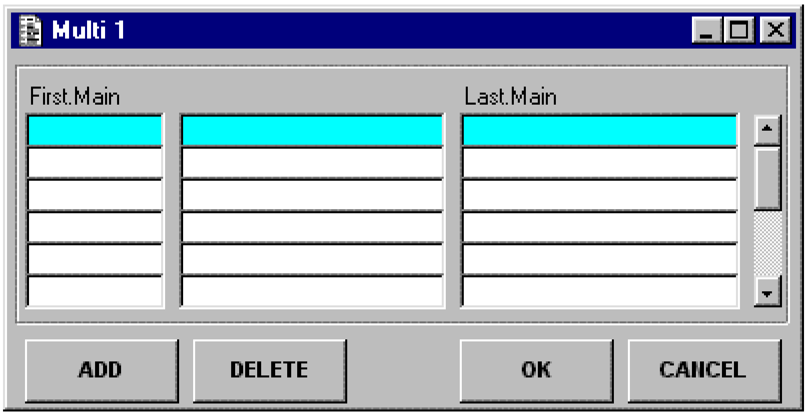
PROC VERSION is the version number for that procedure (starting at 01.000.000). Version numbers are as described in section 3.8.3.

Close	Will exit back to the parent form.

4.10 MULTI-PURPOSE

This type of form is for those occasions where the database tables are fairly static and can only be changed by someone with high security clearance (eg: the System Supervisor). In this case having separate forms to add, delete and modify would seem to be an overkill. The general user will only be able to view the contents of this table through a separate read-only function (eg: a popup).

4.10.1 Multi 1



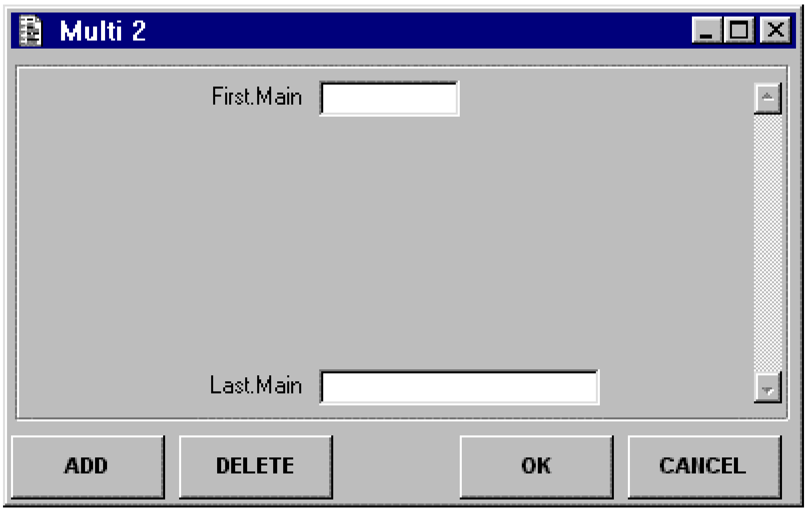
This type of form is for those entities where the number of fields is so small that each occurrence can be displayed on a single line. Two fields are shown in this example, but there may be more.

Upon initial entry the form will perform an automatic retrieve of all available entries.

The active occurrence/field will be highlighted in a different colour.

ADD	Will present the user with an empty occurrence so that a new entry may be input. The database will not be updated until the OK button is pressed.
DELETE	Will delete the current occurrence from the screen, provided that no subordinate entries exist. The database will not be updated until the OK button is pressed.
OK	Will commit all changes to the database (provided that all validation is successful), then exit and return control to the parent form.
Cancel	Will abandon all changes and return control immediately to the parent form.

4.10.2 Multi 2



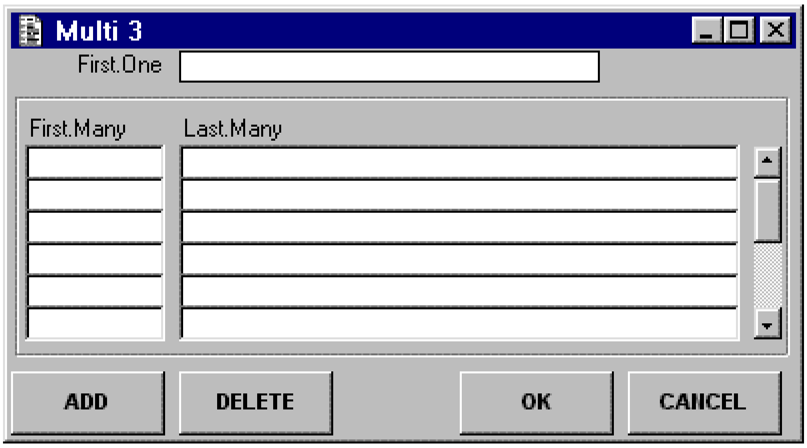
This type of form is for those occasions where the contents of each occurrence is too large to fit into a single line, therefore the screen can only show one occurrence at a time.

Upon initial entry the function will perform an automatic retrieve of all available entries.

The active field will be highlighted in a different colour.

ADD	Will present the user with an empty occurrence so that a new entry may be input. The database will not be updated until the OK button is pressed.
DELETE	Will delete the current occurrence from the screen, provided that no subordinate entries exist. The database will not be updated until the OK button is pressed.
OK	Will commit all changes to the database (provided that all validation is successful), then exit and return control to the parent form.
Cancel	Will abandon all changes and return control immediately to the parent form.

4.10.3 Multi 3



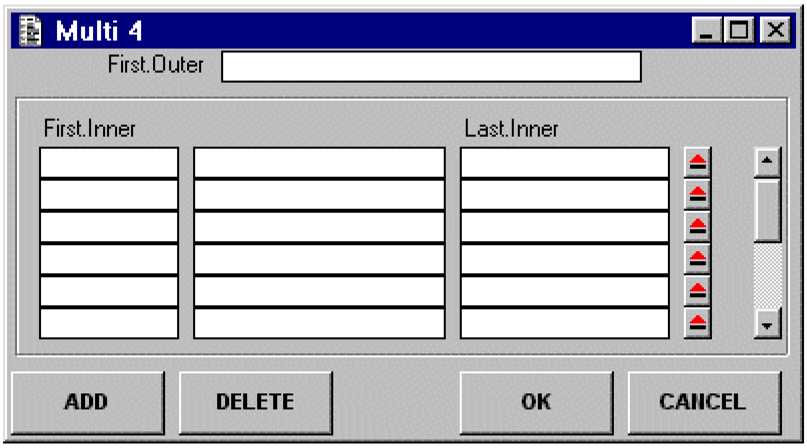
This type of form is used in a ONE-to-MANY relationship in order to maintain the occurrences of the MANY entity for a selected occurrence of the ONE entity.

Upon initial entry the form will retrieve an occurrence of the ONE entity using the primary key passed down by the parent form, then retrieve all associated occurrences of the MANY entity.

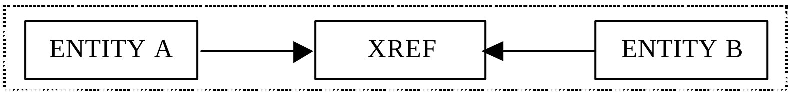
The active occurrence/field will be highlighted in a different colour.

ADD	Will present the user with an empty occurrence so that a new entry may be input. The database will not be updated until the OK button is pressed.
DELETE	Will delete the current occurrence from the screen, provided that no subordinate entries exist. The database will not be updated until the OK button is pressed.
OK	Will commit all changes to the database (provided that all validation is successful), then exit and return control to the parent form.
Cancel	Will abandon all changes and return control immediately to the parent form.

4.10.4 Multi 4

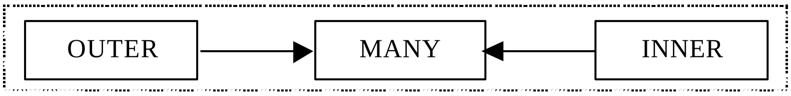


This type of form is used to maintain the details of what is sometimes referred to as a MANY-to-MANY relationship. This can only be supported with the introduction of linking (cross-reference) entity which then produces two ONE-to-MANY relationships, as in the following diagram:-



The primary key of the MANY entity is actually a combination of the primary keys from ENTITY A and ENTITY B. This means that only one occurrence of XREF can exist for a combination of primary key A and primary key B.

In this type of form the structure of the entities is as follows:-



Only occurrences of MANY can be modified - both OUTER and INNER are read-only.

Note that no fields from entity MANY are displayed as it contains nothing but foreign key fields whose values are obtained from the two related entities.

This function will probably need two versions, one where ENTITY A is used as OUTER and ENTITY B is used as INNER, and another where the positions are reversed.

Upon initial entry to this function there will be an automatic retrieve of entity OUTER using the value passed down from the parent form. The main body of the screen will show all occurrences of entity INNER as identified on entity MANY.

The active occurrence/field will be highlighted in a different colour.

ADD Will create a new occurrence of the MANY entity, with an empty INNER entity, and do one of the following, depending on the setting in AUTO_POPUP (which is a local constant):-

Y = automatically call the relevant popup form (without allowing a profile to be entered) so that the user may choose an occurrence of the INNER entity. If no entry is selected the empty occurrence will be deleted.

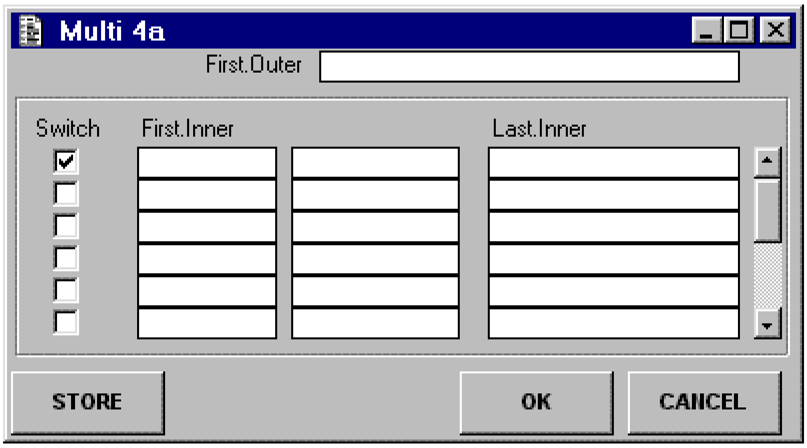
N = wait for the user to press the ⏏ button (or activate the <detail> trigger), which allows a profile to be entered beforehand. If no entry is selected the occurrence will remain empty.

If an occurrence of INNER is selected which already has a link on MANY with the current OUTER it will be discarded with a DUPLICATE PRIMARY KEY message.

The database will not be updated until the OK button is pressed.

DELETE	Will delete the current (highlighted) occurrence of the MANY entity from the screen. The database will not be updated until the OK button is pressed.
OK	Will commit all updates to the database and return to the parent form.
Cancel	Will abandon all updates and return to the parent form.

4.10.5 Multi 4a



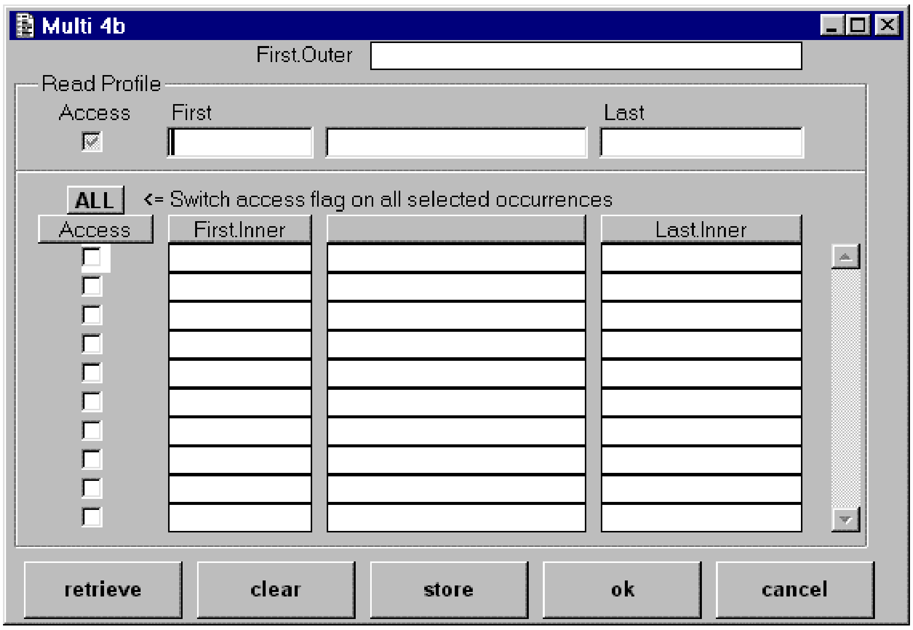
This type of form is similar to Multi-Purpose 4, but displays all occurrences of entity INNER. If a link exists on entity MANY with entity OUTER, then field SWITCH will be ON (non-blank). If no link exists then field SWITCH will be OFF (blank).

Entries on entity MANY can be deleted by turning SWITCH from ON to OFF, or created by turning SWITCH from OFF to ON. This can be done by double clicking on any field within the line. This provides a very fast method of creating or deleting the link between ENTITY A and ENTITY B.

The active occurrence/field will be highlighted in a different colour.

STORE	Will commit all pending updates to the database, but will remain within this function.
OK	Will commit all pending updates to the database and return to the parent form.
Cancel	Will abandon all updates and return to the parent form.

4.10.6 Multi 4b



This type of function is similar to Multi-Purpose 4a, but in cases where there can potentially be a large number of occurrences of entity INNER it may be useful to offer the facility of processing the data in subsets by using the profile area.

Upon initial entry this form will perform an automatic retrieve using a profile with ACCESS SWITCH set to ON so that only those occurrences of entity INNER where there is a linking entry on entity MANY will be displayed. The ACCESS SWITCH in the profile area is actually a tri-state checkbox. The three possible values are:-

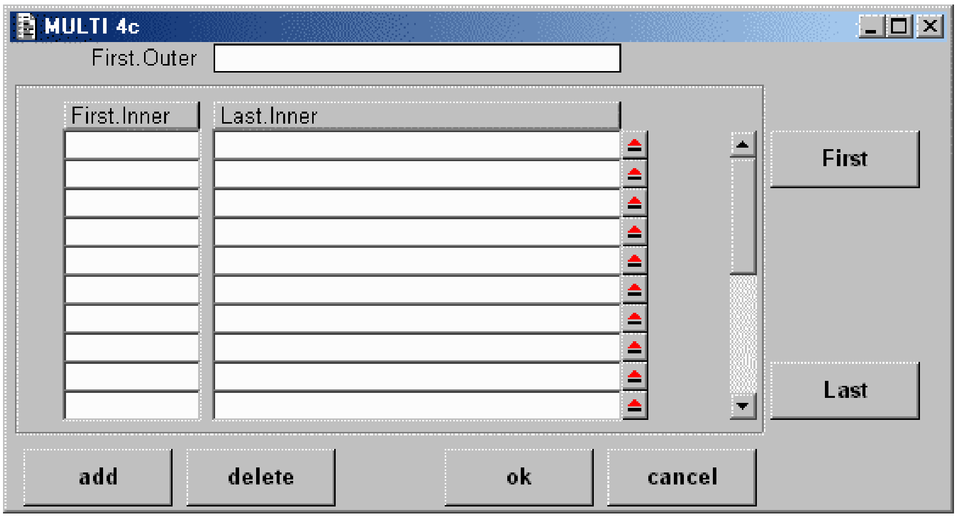
- Blank - select entries which have this switch OFF.
- Checked - select entries which have this switch ON.
- Greyed - select all entries, either ON or OFF.

Entries on entity MANY can be deleted by turning ACCESS SWITCH from ON to OFF, or created by turning ACCESS SWITCH from OFF to ON. This can be done by double clicking on any field within the line.

The active occurrence/field will be highlighted in a different colour.

RETRIEVE	Will retrieve a new set of entries from the database using the current retrieve profile. If the profile is blank then all available entries will be retrieved. If no qualifying entries can be found then a warning message will be displayed.
CLEAR	This will clear the contents of all fields on the current screen, allowing a different profile to be defined in the area provided before attempting another retrieve.
STORE	Will commit all pending updates to the database, but will remain within this form.
OK	Will commit all pending updates to the database and return to the parent form.
Cancel	Will abandon all updates and return to the parent form.

4.10.7 Multi 4c



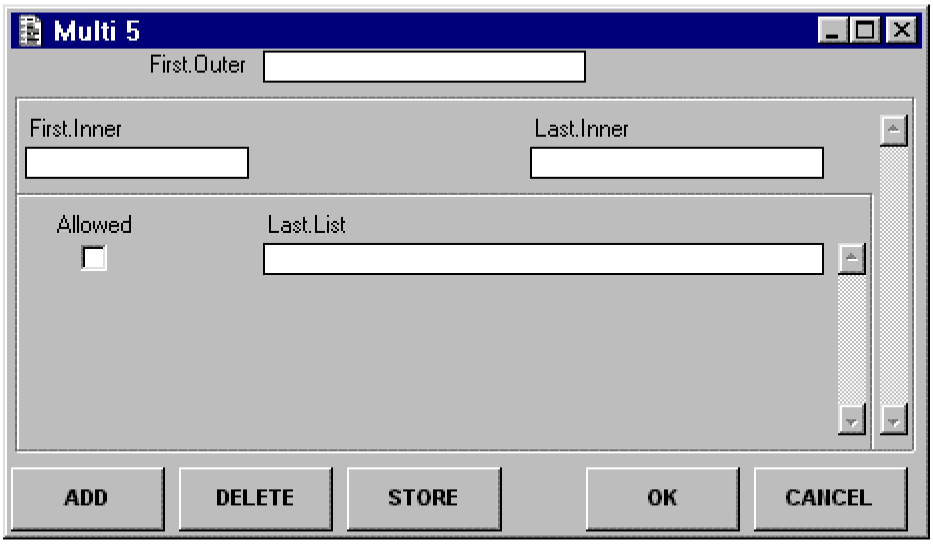
This is the same as Multi 4, but with the addition of a navigation bar.

Upon initial entry to this function there will be an automatic retrieve of entity OUTER using the value passed down from the parent form. The main body of the screen will show all occurrences of entity INNER as identified on entity MANY.

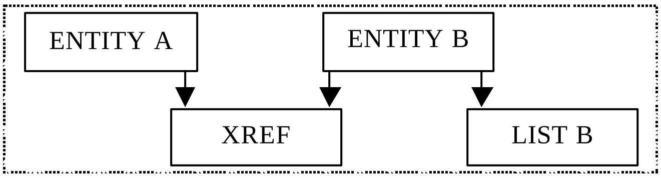
The active occurrence/field will be highlighted in a different colour.

ADD	Will create a new occurrence of the MANY entity, with an empty INNER entity, and do one of the following, depending on the setting in AUTO_POPUP (which is a local constant):- Y = automatically call the relevant popup form (without allowing a profile to be entered) so that the user may choose an occurrence of the INNER entity. If no entry is selected the empty occurrence will be deleted. N = wait for the user to press the ⏏ button (or activate the <detail> trigger), which allows a profile to be entered beforehand. If no entry is selected the occurrence will remain empty. If an occurrence of INNER is selected which already has a link on MANY with the current OUTER it will be discarded with a DUPLICATE PRIMARY KEY message. The database will not be updated until the OK button is pressed.
DELETE	Will delete the current (highlighted) occurrence of the MANY entity from the screen. The database will not be updated until the OK button is pressed.
OK	Will commit all updates to the database and return to the parent form.
Cancel	Will abandon all updates and return to the parent form.
First Nav **Last Nav**	Each of these navigation buttons will pass control to another form in the system, passing as a parameter the primary key of the current (highlighted) occurrence.

4.10.8 Multi 5



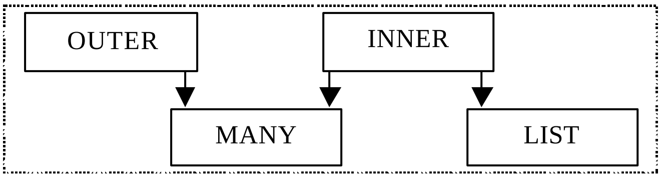
This type of function is similar to Multi-Purpose 4, but caters for a more complex form of linking (cross-reference) entity, as in the following diagram:-



LIST B is a series of occurrences that identify the total number of options that are available for an instance of ENTITY B.

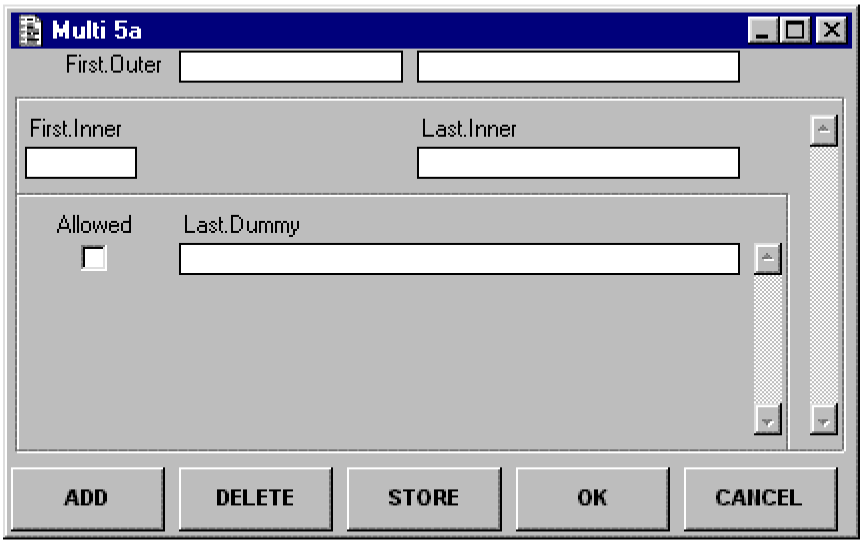
XREF is a link between ENTITY A and ENTITY B containing a single string field (an indexed list) which identifies which options from LIST B have been selected for each instance of ENTITY A.

The structure of the form is as follows:-



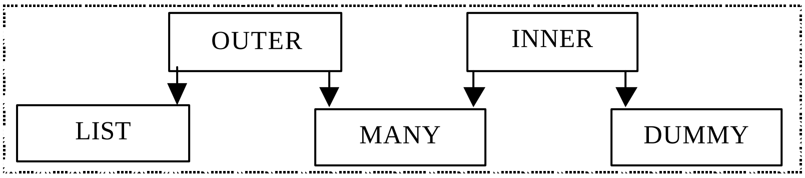
ADD	Will present the user with an empty occurrence so that a new entry may be input. The database will not be updated until the OK button is pressed.
DELETE	Will delete the current occurrence from the screen, provided that no subordinate entries exist. The database will not be updated until the OK button is pressed.
STORE	Will commit all pending updates to the database, but will remain within this form.
OK	Will commit all pending updates to the database and return to the parent form.
Cancel	Will abandon all updates and return to the parent form.

4.10.9 Multi 5a



This type of form is similar to Multi-Purpose 5, but caters for the situation where ENTITY B has to be the outer entity instead of ENTITY A.

The structure of the form is as follows:-



ADD	Will present the user with an empty occurrence so that a new entry may be input. The database will not be updated until the OK button is pressed.
DELETE	Will delete the current occurrence from the screen, provided that no subordinate entries exist. The database will not be updated until the OK button is pressed.
STORE	Will commit all pending updates to the database, but will remain within this form.
OK	Will commit all pending updates to the database and return to the parent form.
Cancel	Will abandon all updates and return to the parent form.

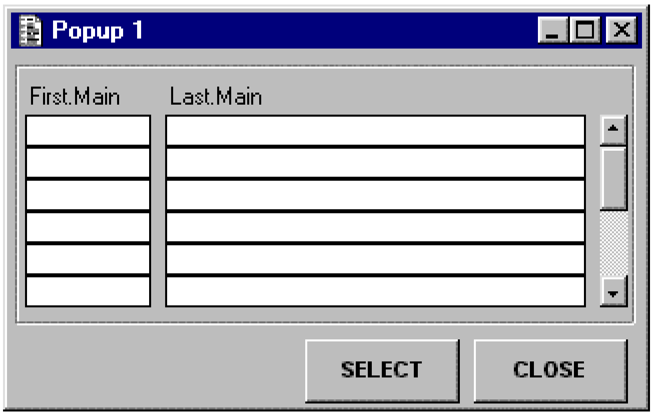
4.11 POPUP / PICKLIST

This type of form is normally invoked by pressing the popup button ⏏ next to the relevant field (known as the popup field), or by double clicking on that field. It is used to display a pick list of allowable values when the user is attempting to enter or change the contents of that field.

This type of form is read-only, and cannot be used to modify the contents of the database.

It is not logical for this type of function to be non-modal therefore, once initiated, it will not be possible to switch focus to another form.

4.11.1 Popup 1



Upon initial entry this form will perform an automatic retrieve using the contents of the popup field on the parent form as a retrieve profile. If the profile is blank (null) then all available entries from the database will be retrieved. If the profile is not null then only those entries that match the profile will be retrieved and made available for display.

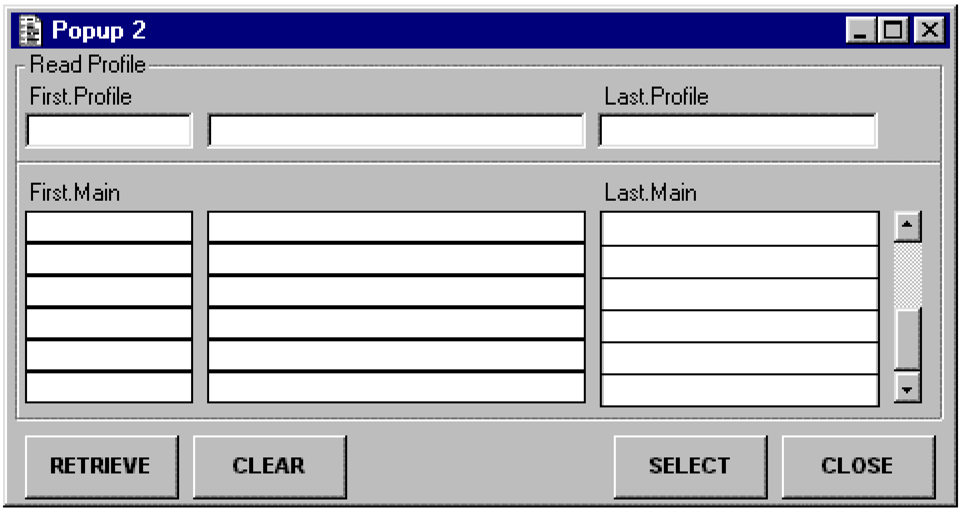
If no entries can be retrieved from the database (eg: nothing matches the retrieve profile) the function will not display a blank screen but will return immediately to the parent form with a warning message. This will allow the user to change the profile characters before making another attempt.

If the contents of the ID field are meaningless to the user then it may be omitted.

The active occurrence will be highlighted in a different colour.

Select	Will return to the parent form with the identity of the chosen entry (the current occurrence) so that the popup field can be filled in with the relevant details from that entry. Alternatively the user may activate the <detail> trigger of the chosen occurrence (refer to section *3.7 Shortcut Keys* for more details).
Close	Will return to the parent form signifying that no entry has been chosen.

4.11.2 Popup 2



This form will not perform an automatic retrieve - instead it will present the user with a blank screen and wait for a retrieve profile to be entered in the area at the top of the screen, after which the RETRIEVE button must be pressed to initiate the search. The CLEAR button is used to clear the current screen contents prior to a new retrieve profile being entered.

The Profile is limited to fields that are contained on the main outer entity.

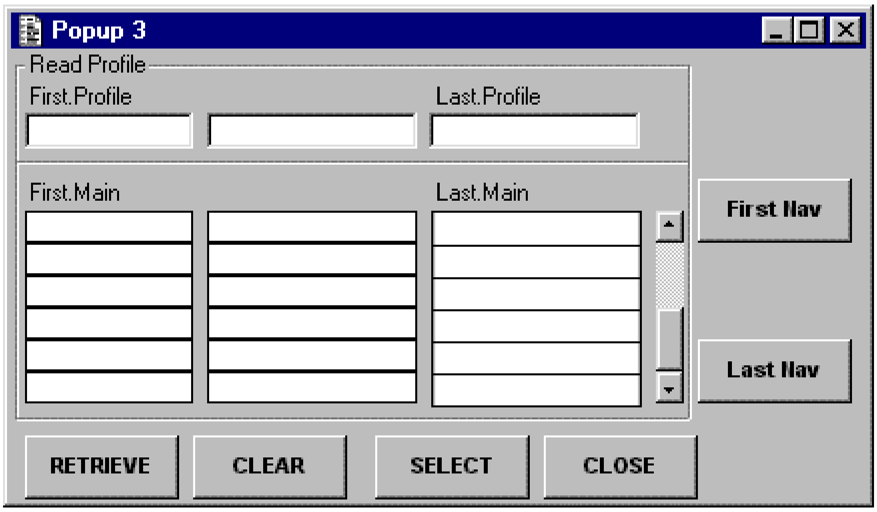
If the contents of the ID field are meaningless to the user then it may be omitted.

The active occurrence will be highlighted in a different colour.

Prompt sequence: If no data has been retrieved the prompt sequence will be from the Profile area to the Action bar, excluding the Data area. If data has been retrieved then the prompt sequence will be from the Data area to the Action bar, excluding the Profile area.

RETRIEVE	Will retrieve a new set of entries from the database using the current retrieve profile. If the profile is blank then all available entries will be retrieved. If no qualifying entries can be found then a warning message will be displayed.
CLEAR	This will clear the contents of all fields on the current screen, allowing a different profile to be defined in the area provided before attempting another retrieve.
Select	Will return to the parent form with the identity of the chosen entry (the current occurrence) so that the popup field can be filled in with the relevant details from that entry. Alternatively the user may activate the <detail> trigger on the chosen occurrence (refer to section *3.7 Shortcut Keys* for more details).
Close	Will return to the parent form signifying that no entry has been chosen.

4.11.3 Popup 3



This form will not perform an automatic retrieve - instead it will present the user with a blank screen and wait for a retrieve profile to be entered in the area at the top of the screen, after which the **RETRIEVE** button must be pressed to initiate the search.

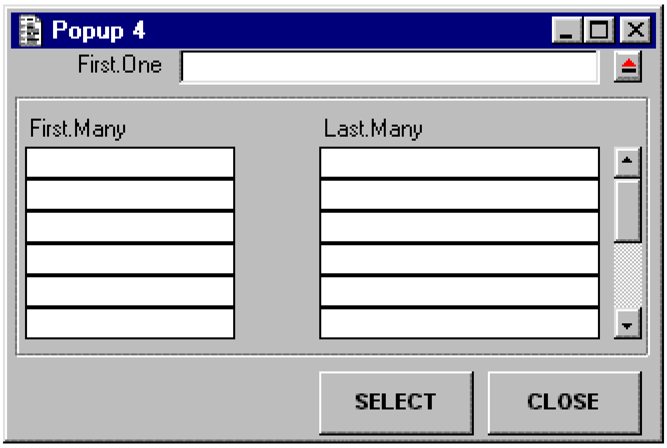
The Profile is limited to fields that are contained on the main outer entity.

The active occurrence will be highlighted in a different colour.

Prompt sequence: If no data has been retrieved the prompt sequence will be Profile area then Action bar, excluding the Data area and Navigation bar. If data has been retrieved then the prompt sequence will be Data area, Navigation bar, then Action bar, excluding the Profile area.

RETRIEVE	Will retrieve a new set of entries from the database using the current retrieve profile. If the profile is blank then all available entries will be retrieved. If no qualifying entries can be found then a warning message will be displayed.
CLEAR	This will clear the contents of all fields on the current screen, allowing a different profile to be defined in the area provided before attempting another retrieve.
Select	Will return to the parent form with the identity of the chosen entry (the current occurrence) so that the popup field can be filled in with the relevant details from that entry. Alternatively the user may activate the <detail> trigger on the chosen occurrence (refer to section *3.7 Shortcut Keys* for more details).
Close	Will return to the parent form signifying that no entry has been chosen.
First Nav **Last Nav**	Each of these navigation buttons will pass control to another form in the system, passing as a parameter the primary key of the current (highlighted) occurrence. • If an ADD function is called this will cause the new database occurrence to be inserted into the screen display after the current occurrence. • If a MODIFY function is called this will cause the screen display to be updated to reflect any changed values. • If a DELETE function is called this will cause the relevant occurrence to be removed from the current screen display.

4.11.4 Popup 4



This type of form is used in a ONE-to-MANY relationship where only those occurrences of MANY that are related to the occurrence of ONE can be displayed and selected.

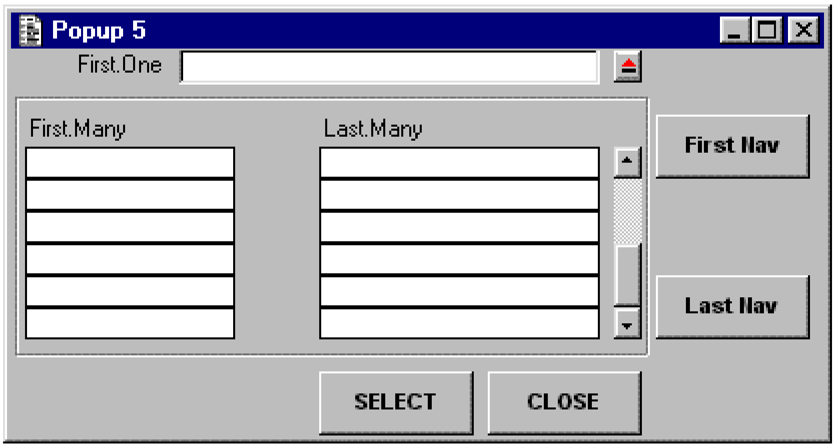
The action taken upon initial entry depends of the existence of a passed parameter:-

- If a parameter is passed an occurrence of the ONE entity will be retrieved, followed by any associated occurrences of the MANY entity. The ⏏ button will be hidden, and the cursor will be positioned on the first occurrence of the MANY entity.
- If no parameter is passed the function will not be able to perform an automatic retrieve. The cursor will be positioned on the first field of the ONE entity, and the user must select an occurrence (using another popup) before any occurrences of the MANY entity can be retrieved, displayed, and selected.

The active occurrence/field will be highlighted in a different colour.

Select	Will return to the parent form with the identity of the chosen entry (the current occurrence) so that the popup field can be filled in with the relevant details from that entry. Alternatively the user may activate the <detail> trigger on the chosen occurrence (refer to section *3.7 Shortcut Keys* for more details).
Close	Will return to the parent form signifying that no entry has been chosen.

4.11.5 Popup 5



This type of form is used in a ONE-to-MANY relationship where only those occurrences of MANY that are related to the occurrence of ONE can be displayed and selected.

The action taken upon initial entry depends of the existence of a passed parameter:-

- If a parameter is passed an occurrence of the ONE entity will be retrieved, followed by any associated occurrences of the MANY entity. The ⏏ button will be hidden, and the cursor will be positioned on the first occurrence of the MANY entity.
- If no parameter is passed the function will not be able to perform an automatic retrieve. The cursor will be positioned on the first field of the ONE entity, and the user must select an occurrence (using another popup) before any occurrences of the MANY entity can be retrieved, displayed, and selected.

If the required entry does not currently exist a navigation button may allow it to be created.

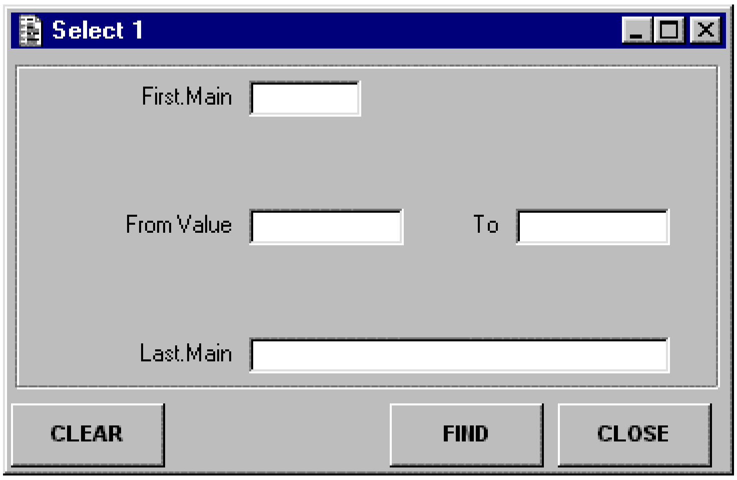
The active occurrence/field will be highlighted in a different colour.

Select	Will return to the parent form with the identity of the chosen entry (the current occurrence) so that the popup field can be filled in with the relevant details from that entry. Alternatively the user may activate the <detail> trigger on the chosen occurrence (refer to section *3.7 Shortcut Keys* for more details).
Close	Will return to the parent form signifying that no entry has been chosen.
First Nav Last Nav	Each of these navigation buttons will pass control to another form in the system, passing as a parameter the primary key of the current (highlighted) occurrence. • If an ADD function is called this will cause the new database occurrence to be inserted into the screen display after the current occurrence. • If a MODIFY function is called this will cause the screen display to be updated to reflect any changed values. • If a DELETE function is called this will cause the relevant occurrence to be removed from the current screen display.

4.12 SELECTION

This type of form is an extension of the profile area defined at the top of LIST forms. It is used when the selection criteria for data retrieval contains more data items that can fit comfortably into a single-line area. This is also useful when it is required to specify ranges of values for a single field on the database - in this type of form separate fields can be shown for the "from" and "to" values.

4.12.1 Select 1



This type of form can either be the child of a LIST form, or the parent of a LIST form. This depends on the setting of **$child_form$** (see below).

It may be activated when the Full Profile button in a LIST form is pressed, in which case any selection criteria are passed back to that form.

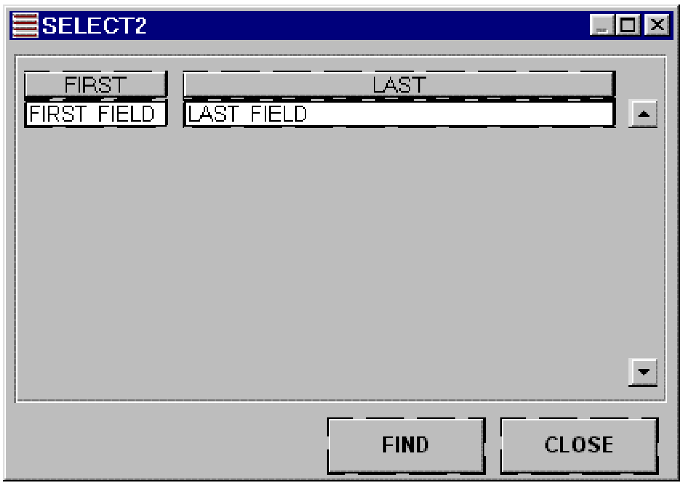
Upon initial entry the screen will be empty. Alternatively initial values can be defined on the Menu database (refer to transaction MNU_0080M in the menu and Security System for details). The user then enters whatever selection criteria are required and presses the FIND button. Any validation (such as checking for valid ranges in the "from" and "to" values) takes place before control is passed back/forward to the designated form.

CLEAR	This will clear the contents of all fields on the current screen, allowing a different profile to be defined.
FIND	If **$child_form$** is not empty then the component with this name will be activated, and the selection criteria will be passed down as the parameter. When the child form exits the original selection criteria will be shown. If **$child_form$** is empty then this form will return control to its parent form, passing back any selection criteria. In either case it will be the responsibility of the form which receives the selection criteria to actually retrieve the data from the database
Close	Will return to the parent form with a blank profile.

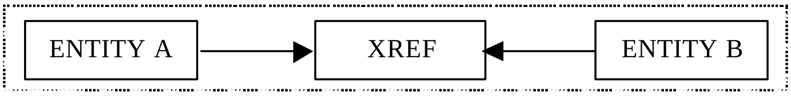
The following options can be specified in *Extra Parameters* in the transaction definition on the Menu database:-

$child_form$	The possible values are *blank* or the name of a form component.

4.12.2 Select 2

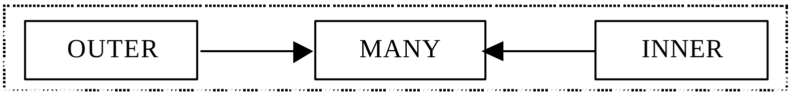


This type of form is used to select occurrences of ENTITY B based on a selection from ENTITY A, and where there is no direct relationship between these two entities, as in the following diagram:-



The primary key of the MANY entity is actually a combination of the primary keys from ENTITY A and ENTITY B. This means that only one occurrence of XREF can exist for a combination of primary key A and primary key B.

In this type of form the structure of the entities is as follows:-



Only occurrences of OUTER are displayed on the form. The user selects an occurrence and pressed the FIND button. This retrieves associated occurrences of MANY, and for each occurrence it uses the primary key of INNER to construct a hitlist of INNER entities. When this is complete it passes the hitlist of INNER entities to the form nominated in $child_form$.

Upon initial entry to this function there will be an automatic retrieve of the OUTER entity.

The active occurrence/field will be highlighted in a different colour.

FIND If **$child_form$** is not empty then the component with this name will be activated, and the hitlist of INNER entities will be passed down as the parameter. The nominated form must be capable of receiving a hitlist in this fashion (eg: List 4). When the child form exits the user will be able to make another selection.

If **$child_form$** is empty then this form will return control to its parent form.

Close Will return to the parent form.

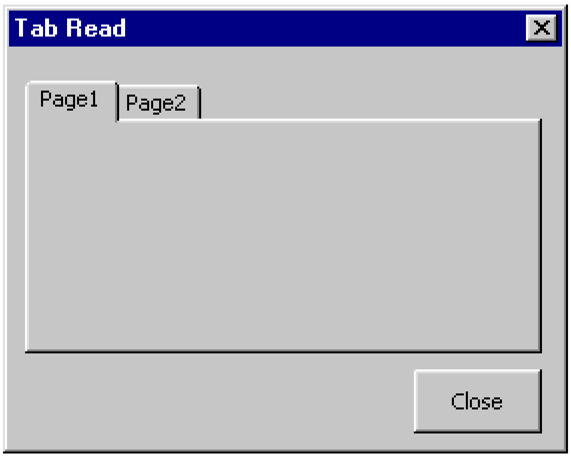
4.13 TABS

A Tab form is actually a group of forms – a Tab Parent that contains the tab widget, and a series of child forms (known as Tab Pages). One of the tab pages will be defined as the initial selection, and will be displayed when the parent form is activated. The other tab pages will only become visible when the relevant tab is selected. Only one tab page can be visible at a time.

The whole of each child form will appear inside the area of the tab widget defined in the parent, therefore the dimensions of each child form should not exceed those of the tab widget in the parent. If they do, this will cause scroll bars to become visible inside the tab widget.

Tab Pages (the child forms) do not usually have any action buttons of their own - they follow any action taken in the parent form.

4.13.1 Tab Read Parent



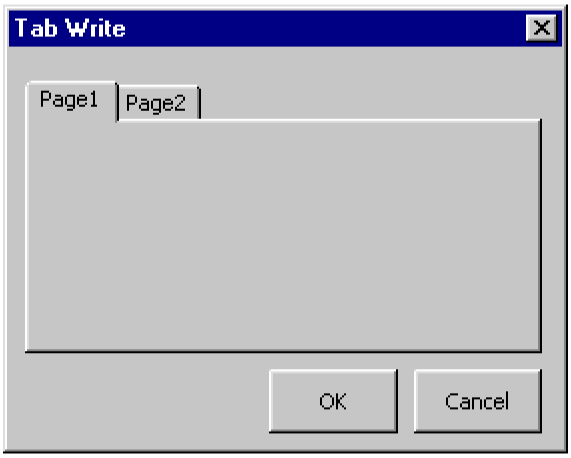
This type of form will be used to display the contents of an object. Only one occurrence of that object can be displayed with each operation of this form.

Upon initial entry there will be an automatic retrieve of the relevant occurrence using the primary key value passed down from the parent form. Data will then be passed down to a series of child forms that will become visible when the relevant tab is selected on the tab widget.

4.13.2 Tab Read Child

This type of form can only be called from a Tab Read Parent. Data to be displayed is passed down from the parent when the form is activated.

4.13.3 Tab Write Parent



This type of form can be used to modify the contents of an object. Only one occurrence of that object can be modified with each operation of this form.

Upon initial entry there will be an automatic retrieve of the relevant occurrence using the primary key value passed down from the parent form. Data will then be passed down to a series of child forms that will become visible when the relevant tab is selected on the tab widget. The user changes the relevant data, then presses the OK button to update and exit.

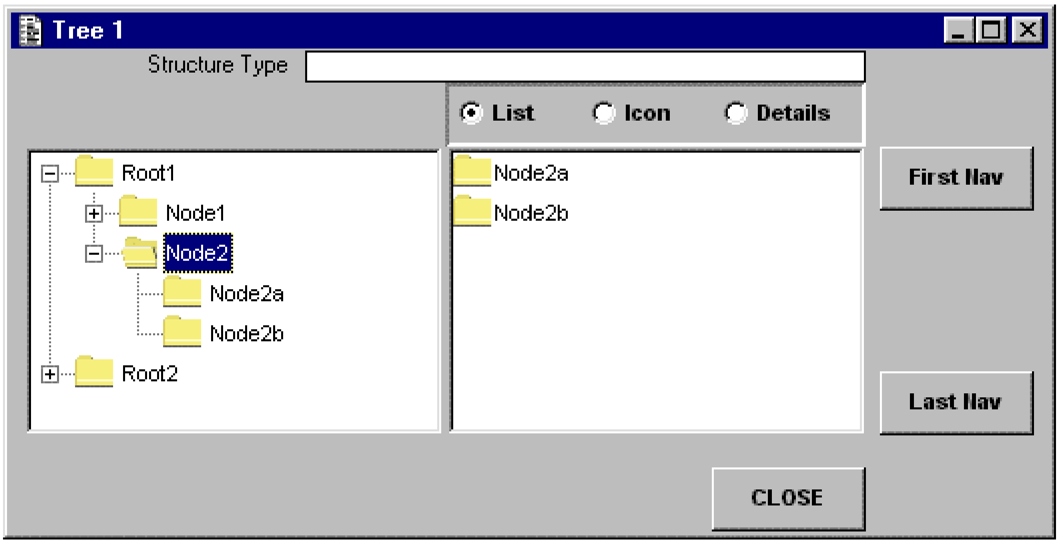
4.13.4 Tab Write Child

This type of form can only be called from a Tab Write Parent. Data to be displayed is passed down from the parent when the form is activated. When the OK button is pressed any changes made within the form will be passed back to the parent so that they may be applied to the database.

4.14 TREES

This type of function is used to display a structure hierarchy in tree form, similar to that provided by the explorer in Micosoft Windows. The user may open up any branch in the structure by clicking on the **expand** (+) symbol next to a node. Conversely, any branch may be closed by clicking on the **collapse** (-) symbol.

4.14.1 Tree 1



● List ○ Icon ○ Details

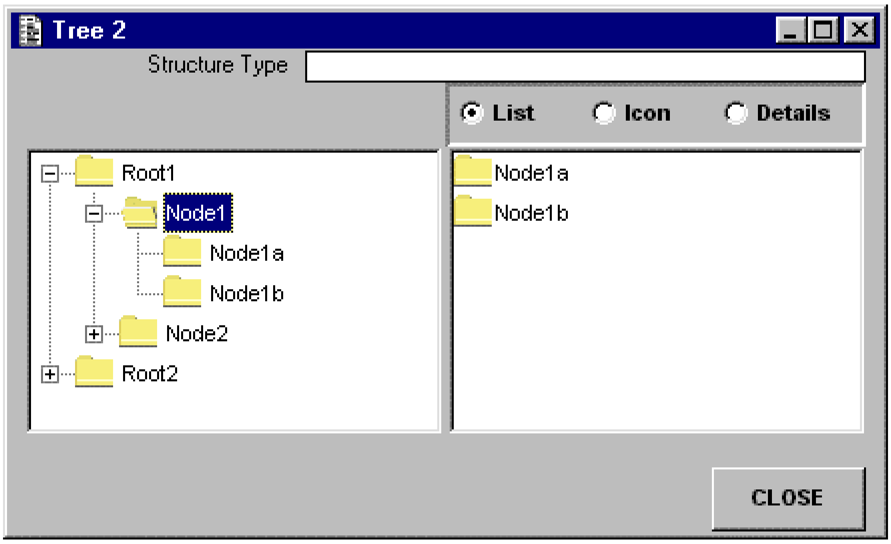
This type of form is used to display a structure hierarchy in tree form. Options on the navigation bar may allow the user to add, amend, or delete any node in the structure.

Upon initial entry the form will retrieve an occurrence of Structure Type using the primary key passed down from the parent form. The screen will then display the root node(s) for the selected structure. The user may traverse any branch of the structure as required.

The contents of the right-hand pane may be changed by selecting one of the radio buttons labelled List, Icon, or Details. These have the same effect as the options in the Windows Explorer.

Close	Will return to the parent form.
First Nav Last Nav	Each of these navigation buttons will pass control to another form in the system, passing as a parameter the primary key of the selected item.

4.14.2 Tree 2



● List ○ Icon ○ Details

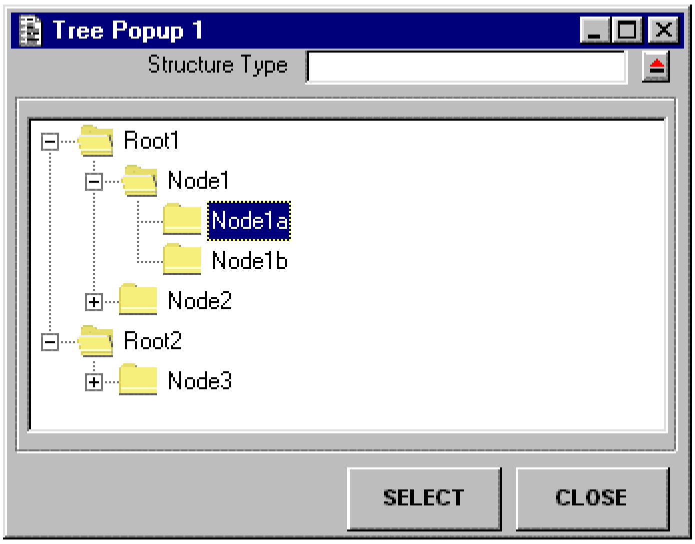
This type of form is used to display a structure hierarchy in tree form.

Upon initial entry the function will retrieve an occurrence of Structure Type using the primary key passed down from the parent form. The screen will then display the root node(s) for the selected structure. The user may traverse any branch of the structure as required.

The contents of the right-hand pane may be changed by selecting one of the radio buttons labelled List, Icon, or Details. These have the same effect as the options in the Windows Explorer.

Close	Will return to the parent form.

4.14.3 Tree Popup 1



This type of form is used to display a structure hierarchy in tree form.

The action taken upon initial entry depends on the existence of a parameter passed down by the parent process:

- If a parameter is passed this will be used to retrieve an occurrence of Structure Type, then the contents of that structure will be automatically retrieved and displayed.
- If no parameter is passed the user will be required use the relevant popup screen in order to choose an entry from those which are currently available.

the screen will display the root node(s) for the selected structure. The user may traverse any branch of the structure as required before selecting the desired item.

Select	Will return to the parent form with the identity of the chosen item (highlighted) so that the popup field can be filled in with the relevant details from that item.
Close	Will return to the parent form signifying that no item has been chosen.

4.14.4 Tree Service (Nodes)

This is a hidden form (self-contained service) that retrieves data from the database as required by any form which needs to display that data in the form of a tree structure. It will pass back the selected data in the form of a list of values that are suitable for loading into the tree widget.

This component deals with Nodes only.

This component contains the following operations:-

PROPERTIES	This is used to initialise the tree contents. It returns values for the tree properties (the number of levels in the structure, and the icons to be used to indicate the state of an entry due to user action) as well as the details of the root node(s).
EXPAND	When a Node is expanded this will supply the details of that Node's children.
SELECT	When a Node is selected this will supply the details for that Node.

4.14.5 Tree Service (Leaves)

This is a hidden form (self-contained service) that retrieves data from the database as required by any form which needs to display that data in the form of a tree structure. It will pass back the selected data in the form of a list of values that are suitable for loading into the tree widget.

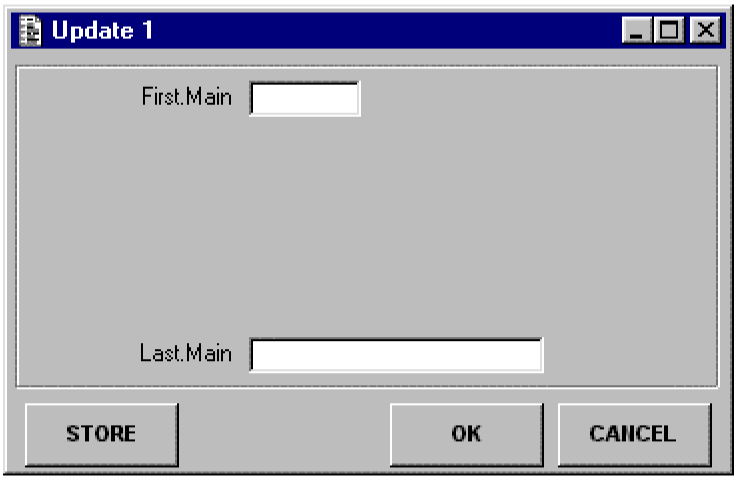
This component deals with Leaves only.

This component contains the following operation:-

EXEC	When a Node is selected this will supply the details of al the Leaf entries that are attached to that Node.

4.15 UPDATE / MODIFY

4.15.1 Update 1



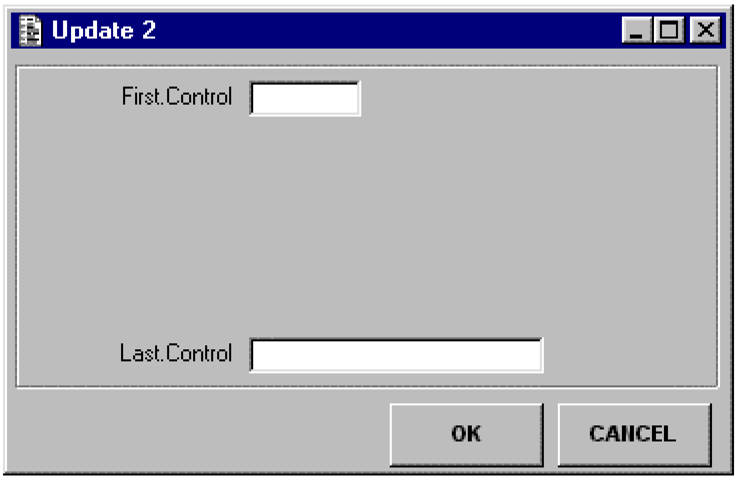
This type of form is used to modify the contents of an entity. Only one occurrence can be modified with each operation of this form.

Upon initial entry there will be an automatic retrieve of the relevant occurrence using the primary key value passed down from the parent form. The user changes the relevant data, then presses the OK button to update and exit.

The active field will be highlighted in a different colour.

STORE	Will add the current entry to the database and clear the screen, allowing details for another occurrence to be entered.
OK	Will update the entry on the database and return to the parent form.
Cancel	Will abandon all updates and return to the parent form.

4.15.2 Update 2



A lot of applications require some sort of control record which contains a series of settings which can be modified by the user to determine various options. This is usually implemented as a single record containing a separate field for each individual setting. With this structure it is not very easy to add extra settings as the record layout would have to be modified, and all forms referencing this layout would have to be recompiled.

An alternative and much more flexible approach would be to hold each setting on its own record rather than all settings on a single record. This can be achieved with a record layout containing just two columns - *fieldname* and *fieldvalue.*

This type of form is used to modify the contents of these control records. If any field painted on the form has a name which appears as *fieldname* on a control record then it is automatically loaded with the corresponding value. When writing to the database if any field on the form does not currently have a corresponding control record then one will automatically be created. This effectively allows new fields to be added to the control record without having to alter the structure of the database. The only form that needs to be recompiled is the form that references the new field.

As this entity is independent of all other entities this is a modal form. Once activated it will not be possible to switch focus to any other form.

Upon initial entry the existing values will be automatically retrieved. The user changes the relevant data, then presses the OK button to update and exit.

The active field will be highlighted in a different colour.

OK	Will update the entry on the database and return to the parent form.
Cancel	Will abandon all updates and return to the parent form.