Introduction to UNIFACE

The advanced 4th generation

application development

and run time system.

Uniface B.V. Amsterdam, April 1989

This publication was written by Jeffrey Mann and Deborah ffoulkes. Please send comments or questions about Uniface or Uniface documentation to:

Uniface B.V.
Hogehilweg 3
1101 CA Amsterdam
The Netherlands

Tel. +31 (0)20 976644   Fax +31 (0)20 912912

Introducing.......UNIFACE

Uniface, the innovative fourth generation application development and run time system produced by Uniface B.V. of Amsterdam - what is it, how does it work, why should I use it? This brochure provides you with the answers.

The product description offered here introduces you to the Uniface system and presents some insights into the concepts and philosophy which guided its makers in developing it. After reading this brochure, you should understand the terms and concepts used in the Uniface development environment.

We do not attempt to explain in detail how to use Uniface. Most of the functions available in Uniface are mentioned, but they are not explained fully, nor is complete syntax for statements or commands provided. Readers who wish to use Uniface to build an application should refer to the other Uniface publications described below.

This brochure is intended for the technical staff and management of software development houses or departments. The reader should be familiar with software development, database management and general data processing. Common data processing and database management terms are not explained, except when Uniface uses them differently from their commonly accepted meanings.

Chapter one gives a brief history and overview of the Uniface system. Chapter two describes the standard user interface used in all Uniface applications. The interface functions are described using the Uniface demo application. Chapter three describes how to use the designer's workbench to define some of the features included in the application. Chapter four looks at the system administrator's workbench to explain how the conceptual schema of the application was defined.

Although this reverses the order in which the development process is usually pursued, we have chosen this order to illustrate how to define each function directly after it has been described.

The database interface, governed by the Uniface driver, is described in Chapter five. Chapter six gives an overview of the Information engineering & Design Facility (IDF). Chapter seven concludes with a look at the planned future developments of the Uniface system.

Other Uniface publications

There are five other publications in the Uniface library.

The *Uniface Information engineering & Design Facility (IDF) Manual* provides complete documentation of how to use Uniface to develop and manage database applications. This is the main publication in the Uniface documentation library.

The *Uniface User Interface Guide* is a short booklet which explains how to use the Structure Editor and other functions of the standard user interface available with Uniface.

The *Uniface Messages Guide* explains all of the messages sent by the Uniface run time system and the IDF.

The *Uniface Installation Guide* explains how to run the installation procedures which install the Uniface system on a particular machine.

The *Uniface DMBS Driver Cookbook* gives you the basic recipe of a Uniface DBMS Driver and explains how to make one yourself.

In addition to these publications, the *Inside Information* newsletter provides news, tips and suggestions about Uniface. *Inside Information* is sent periodically to all current clients and interested parties. If you would like to receive any of these publications, be placed on the mailing list for the newsletter, or would like additional information about Uniface, contact Uniface B.V. at the address shown on page i or one of the Uniface distributors.

TABLE OF CONTENTS

Chapter 1. History & Overview of the Uniface System

The Uniface story began when a small group of developers left a larger, more traditional company to create their own brainchild. That was in 1983. Today, they are marketing a mature product, which is *unique* in the Data Processing world. This is why people have difficulty understanding what it is when they first encounter it: it's not a CASE tool, it's not a DBMS, and it's certainly not just another front-end tool for a database. So what is it ?...

...Uniface is an innovative Fourth Generation Language (4GL) environment for building and operating database applications in a wide range of application areas such as office automation, administrative data processing, document management, etc.

Uniface is a *uni*que, *uni*form, *uni*versal inter*face*. The concept of universality was of fundamental significance in its creation. Bodo Douqué, Managing Director, explains:

> "A multiplicity of DBMSs in a corporate network of mainframes, minis and PCs is unavoidable. If you want to develop applications in such an environment, it goes without saying that you have to be able to accommodate several DBMSs and machines within your strategy. The application development environment, therefore, has to be *completely independent* of both DBMS and machine."

Putting theory into practice: the ANSI report

The universality concept is built into Uniface by utilizing a 3-schema architecture, consisting of an internal, an external and a conceptual schema. The need for this sort of approach to the design of information systems was identified by ANSI/SPARC database study group, and set down in what has now become a classic report·

The International Standards Organization confirmed the findings of the ANSI report several years later. It is perhaps worthwhile to quote them at length here, since their findings are not only crucial to an understanding of Uniface but also of fundamental importance to all interested in information system design.

> "The reports of the ANSI/X3/SPARC DBSG identified the need for a conceptual schema in the context of a three-schema framework...
>
> Subsequent papers...have emphasised the importance of a conceptual schema to users and

designers...In this context, a conceptual schema comprises a unique central description of the various information contents that may be in a database. This includes the description of what actions, such as changes and retrievals, are permissible on the information content. The database itself may be implemented in any one of a number of ways. Users and application programs may view the data in a variety of ways, each described by an external schema. Each external schema is therefore derived from the common conceptual schema. The physical storage structure that may be in use at any given time is described by an internal schema that is also derived from the conceptual schema.

The conceptual view, as meant by ANSI/SPARC, concentrates on the meaning of the information. It is the conceptual schema that describes this view. The external views concentrate on the forms - the data - that represent the information to the outside. These are described in the external schemata. The internal view concentrates on the internal physical representation of the data inside the computer system and is described in the internal schema.

Such a three-schema framework is widely, but not yet universally accepted...

We believe the database user will benefit from the clear separation of the information meaning from the external data representation and the internal physical data storage layout."

As far as we know, Uniface is the only product of its kind which has put the recommendations of this report into practice.

The Uniface philosophy

It is our belief that an application can be completely defined in terms of the data it processes: application definition *is* data definition. After the information analysis phase the conceptual data structure (entities and their relationships) is stored in a central dictionary, together with referential and other integrity constraints, semantic

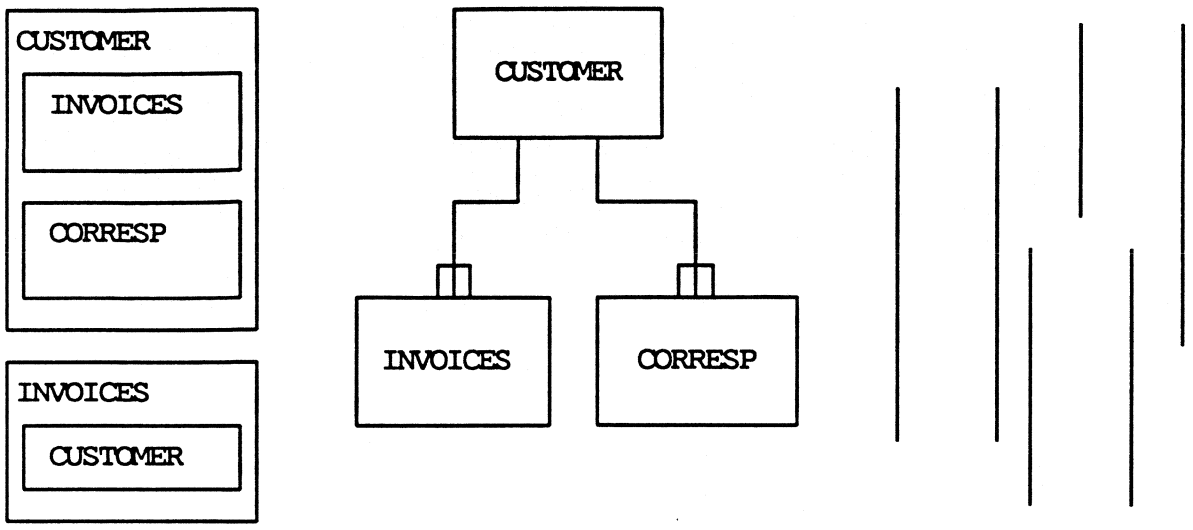


Figure 1-1. The ANSI/ISO 3-Schema architecture.

validation rules, derivation formulae, syntax checks, formatting rules, etc. We generically refer to these as "constraints". Collectively, data structures together with constraints completely define an application. This conceptual heart drives the whole application development process.

Data definition in Uniface is, however, not limited to this description of the database structures and constraints, the *conceptual schema*. End user transactions are defined in terms of other data structures and constraints superimposed, as it were, upon the conceptual "vision". These other structures are the *external schemas*, represented as forms. An external schema represents the user's view of a subset of the database for a particular transaction. An application can include any number of external schemas.

The Uniface runtime system uses these definitions to control the end-users' transactions against the database. This occurs at the lowest level, that of the *internal schema*, and it is this which allows Uniface to offer database independence: the internal schema maps entities in the conceptual structure to tables or files in one or more database management systems (DBMSs). This is done via a specially-designed driver which "speaks" to the DBMS in its own language. The DBMS is specified per entity, which means that applications can be built using several different DBMSs. The degree of independence offered by Uniface in designing applications is such that the assignment of entities to DBMSs can even be altered dynamically. An entity and its DBMS are not fated to remain together "until death us do part".

With Uniface, application development becomes a data management task. The Uniface run time system manages data held in temporary work space with the external schemas, whereas a DBMS manages permanent data stored on disk, and is directed by the internal schema. The conceptual schema straddles these two, as it were. On the one hand it defines the constraints, formatting and rules which give the data its initial structure. And on the other hand it waves the sceptre in the direction of the DBMS, leaving the internal schema to the details of executing its imperatives.

Sketch of a typical Uniface transaction

This section provides a quick overview of the Uniface run time system, i.e. what the user experiences when operating a Uniface application. The facilities and concepts mentioned here are discussed in more detail in the rest of this brochure.

Forms with Frames

Uniface presents an external structure on a seemingly 3-dimensional Form which includes Frames. These Frames offer the user a clear overview of the external structure. Frames are windows over entities and fields. Frames can show a limited number of occurrences in entities, and of characters and lines in fields. The designer determines how much data to show at a time. Further data, i.e. additional occurrences or lines of a document, are available "deeper" in the Form.

Uniface I/O

After calling a specific Form, for example, EMPLOYEE DATA, the user types in say, London & Tokyo, in the City field frame of the branch address. S/He then presses the I/O function key RETRIEVE to perform a Query-By-Form, retrieving data on all employees in the London and Tokyo branches. Uniface extracts the requested data from the database(s) using the query profile supplied by the user and the data definitions of the designer.

At the end of the Form edit session (see below) the user can press the STORE key to update the modified, added and removed occurrences in the database. Uniface performs the necessary I/O between the internal and external schemas. Whenever I/O is done, Uniface generates commands in the specific DML of the DBMS, e.g. SQL.

Editing structure.......and text

The user can browse through and edit the structured data on the Form using the Structure Editor's Travel & Edit functions. S/He can go to the NEXT or PREVIOUS OCCURRENCE, REMOVE an OCCURRENCE, move to the NEXT FIELD, ZOOM a Field Frame to make it temporarily larger, etc. The Structure Editor operates like a word processor, but applied to structured data.

The Structure Editor also includes full-screen text editing functions (e.g. REMOVE WORD, NEXT LINE, SAVE BLOCK, etc.) to edit within Field Frames. These functions are particularly useful when editing fields which contain long formatted strings, i.e. documents.

Event-triggered & data-driven processing

Constraint definitions can be defined either in declarative format, or as processing specifications (Procs). All these definitions are stored in the application dictionary and activated during the edit session. Events (e.g. leaving modified field, pressing STORE, etc.) for the current Frame or Form trigger these constraints definitions.

Procs can perform tasks such as validation, integrity constraints checks, formatting, calculation, I/O and session control, etc. Processing in Uniface is event-triggered and data-driven; the combination of an event and the current data element determines the processing which is activated.

Data-driven session control

During the Form edit session, the user can press the DETAIL key to call another Form, which appears in an overlapping window on the screen. A Proc associated with the current Frame determines which Form appears. Detail Forms are used for tasks such as querying the database for additional information, providing assistance with code lists, Index Forms, showing parts of the structure which are not normally needed by the user, etc. Other functions access standard system Forms used to define CHARACTER ATTRIBUTES such as boldface and underline, define formatting with the RULER, search for text strings within a field with the FIND function, etc.

Complex dialogues with recursive forms

Another Form can be called from any of these Forms, another Form from that one, and so on and so on. There is no limit to the recursiveness of external schemas. All Structure Editor functions are available on any Uniface Form. The ACCEPT and QUIT function keys always return the user to the previous Form, where data and context are retained.

The data-driven dialogue model sketched above provides an elegant framework to easily and naturally define the most complex and demanding transactions, which are still easy for users to understand and use.

Sketch of a typical development cycle

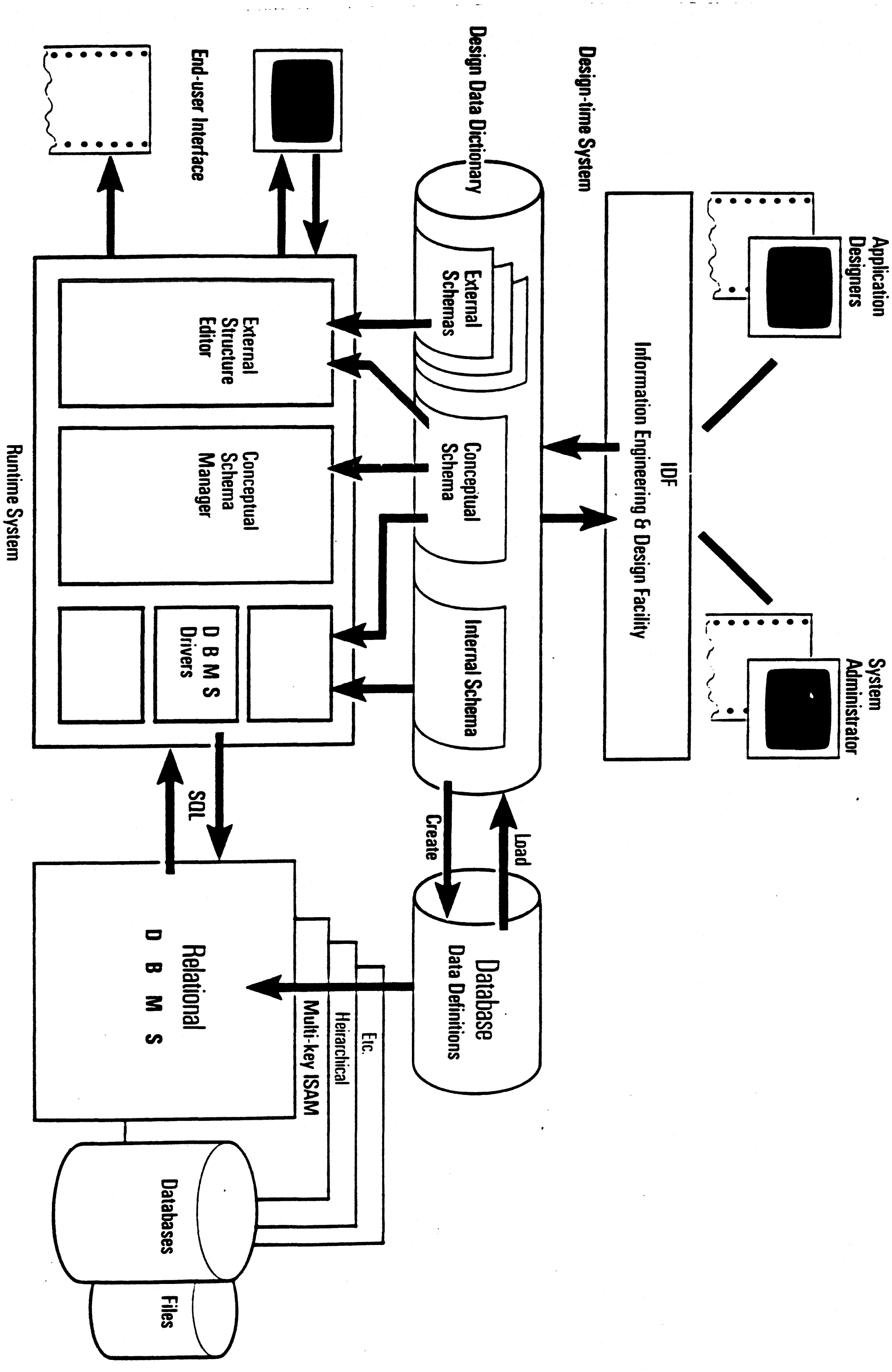
Uniface provides support for every stage of the development cycle. From the initial data model specifications to maintenance of the final application, the necessary tools are provided with the Information engineering & Design Facility, the IDF.

The development process usually starts with the system administrator defining the conceptual schema in terms of an extended relational model with one-to-many relationships between entities. If a Computer Aided System Engineering (CASE) tool has been used, the definitions and constraints can be loaded directly into the conceptual schema using the Load Defintions Utility. Information in the DBMS concerning entities, relationships and constraints can also be loaded.

The conceptual data structure and all of the constraints are defined centrally. By using the system administrator's workbench in the IDF, s/he has control during the entire development process. Application designers then use *their* workbench to define external schemas based on the definitions in the conceptual schema. They can use the FastForm facility to load fields onto the Paint Tableau and then enhance and modify the definitions provided by the system administrator. Definitions can be added which apply only to the specific Form being defined.

Although we refer to the positions of "designer" and "system administrator", it is of course not strictly necessary to assign responsibilities as described here. Many software development organizations do have specific persons responsible for these types of tasks. However, it is quite possible to have one person perform both roles, or divide the responsibilities among several people. The choice lies with your organization.

The sequence described above is also obviously not fixed. Designers can begin to define external schemas before a conceptual schema exists or both can be defined concurrently. The emphasis in the Uniface development environment is on flexibility and freedom.


Application Designers
System Administrator
IDF
Information Engineering & Design Facility
Design-time System
Design Data Dictionary
External Schemas
Conceptual Schema
Internal Schema
Load
Create
Database Data Definitions
End-user Interface
External Structure Editor
Conceptual Schema Manager
D B M S Drivers
Runtime System
SQL
Relational D B M S
Multi-key ISAM
Heirarchical
Etc.
Databases
Files

The fruits of a fundamental approach...

* The productivity of software developers is greatly increased, even when defining complex interactive applications. Independent studies have quoted productivity increases by a factor of 30.

* Uniface offers unsurpassed flexibility:

    - it can interface with several different types of DBMS, even from the one external schema. Applications developed under one DBMS can be later effortlessly transferred to another one.

    - it is written in the highly portable "C" language. It runs on several machines and under several operating systems.

    - Uniface is device independent, it can drive any printer or terminal.

* Uniface includes a fully-integrated complete data dictionary. All definitions needed to create fully-functioning application are stored in the application dictionary, which can be managed by any supported DBMS. The Information engineering and Design Facility (IDF) administers this database and "compiles" these definitions into high-performance tables used at application run time.

* The user interface is modern and PC-like, with pop-up windows for menus and indexes, a WYSIWYG (What You See Is What You Get) editor, full color options etc. Data is presented in scrollable, zoomable Frames.

* The IDF is itself a 4GL Uniface application and therefore offers designers the same modern interface as the end-user. Because Uniface was built using itself, it has the inherent flexibility of 4GL systems. It will continue to develop and improve, offering new facilities over a long product life cycle.

* End user applications have a clear, uniform way of presenting data structures and powerful, uniform functions to edit the information in these structures. The unnatural distinction between text and data editing is eliminated.

* Functions which are usually separated, like word processor, report writer, menu manager, query processor, etc. are integrated in a single user interface, increasing user-efficiency tremendously.

What more can a designer (or end-user) wish for?

Chapter 2. The User Interface

This chapter discusses the standard method of presenting external data structures (Forms) and the standard functions available to edit the data in that structure (the Structure Editor). An example application, the Uniface Demonstration application, is used to illustrate the points made. The *Uniface User Interface Guide* provides more details about how to use the functions discussed in this chapter.

Uniface uses a Form to present data in a structured, standard way. Because users need to do more than view data, Uniface includes a Structure Editor which provides functions to browse, move through and edit data, also in a structured, standard way. Forms and the Structure Editor are important parts of the Uniface model of interactive applications. Uniface applications usually consist of several Forms, each representing a separate external structure. The designer defines the user dialogue in terms of these components.

The user works with an external data structure when operating an application. This data structure can be quite complex, with many occurrences and many fields. However, s/he is often limited to a 80x24 terminal screen to perform all these operations. This is too small to show complete structures which include all occurrences.

Conventional "solutions" don't solve the problem

Conventional applications attempt to solve this problem by using several screens to present a complete data structure. They also often need different programs to enter, update or query the same information. These functions are accessed by selecting menu options. All these different screens and menu options require the user to constantly switch context. A great deal of complex programming is usually necessary.

Uniface seeks to solve this problem in a different way. The Uniface user interface utilizes several innovative concepts to help the user perform all his/her tasks while retaining context. Four specific techniques allow the designer to define a dialogue which is not constrained by the limitations of the terminal screen...

* The screen and the Form are separated. Part of the screen acts as a window over a possibly larger Form. The Structure Editor includes Form Travel functions to move across the entire Form window and to scroll the window over the Form.

* The *logical* size of a field or entity and its *physical* representation on the Form are separated. Uniface makes a distinction between the actual size of a field or entity and

the way it is presented on a Form. Uniface can condense the presentation of data by 'windowing'. The concept of windowing several programs onto different portions of the screen is well-known. In Uniface, this principle is extended to the data element. Frames are windows over fields and entities.

The Structure Editor includes Frame Travel & Edit functions to move in a structured way and edit information within Frames. A field can be a fully-formatted document. For this reason, the Structure Editor includes a complete word processor.

* The designer can "hide" less important parts of a data structure, e.g. home addresses of employees in a personnel file. The deferred structure appears on a detail Form. A detail Form appears as a temporary overlay on the main Form. The data on the detail Form is out of sight when not needed, but still easily available to the user.

* A fourth technique eliminates the need to select a separate menu option or use a different Form in order to add, delete, view or modify data. All these functions can be performed in one Form while retaining context. The importance of this unified approach can be appreciated from the significant increase in user efficiency which results from it.

The Form concept in Uniface

A Uniface Form simulates an on-screen 3-dimensional visual representation of the external structure held in working storage. A Form contains Frames which present fields and entities, and fixed text which labels the Frames and explains the Form.

An external structure (or Form) represents a path through the conceptual structure by placing Frames-inside-Frames. Entity and Field Frames appear within the boundaries of the Entity Frame to which they belong. This simple convention allows designers to make external structures which relate to complex conceptual structures. Not only is it easier for the designer to make these Forms, the end user can understand the structures they represent.

Although not seen by the user, there are extensive processing instructions and definitions connected with each Form. Uniface introduces the concept of an external schema, i.e. the visible Form and all the definitions and processing instructions which support it.

External schemas are able to serve many different purposes. Main Forms, Code Lists, Detail Forms and menus are all external schemas, even though each of these serves a very different purpose. The only difference is the way in which they are defined. The Structure Editor operates in the same way in all external schemas.

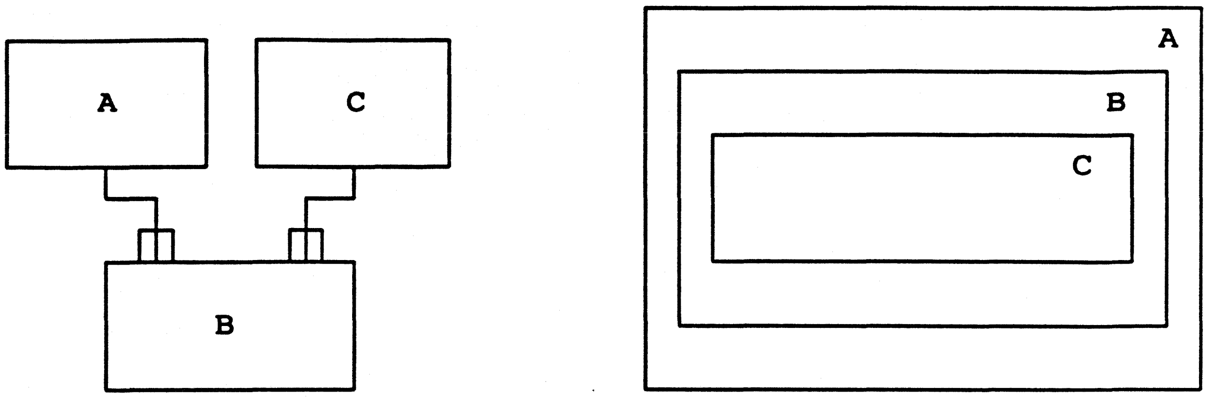


Figure 2-1. Conceptual Data Structure and an External Data Structure with Frames.

The Screen and the Form

The designer has the option to define a screen header or trailer (or both) when s/he defines an application. This space can be used to identify the application or provide other information such as date, time, messages etc. The designer also explicitly defines the Form area. This space is available for Form windows.

The size of a Form is not necessarily related to the size of the screen. The user sees a Form through a Form window. A Form window can be as small as one character, as large as the screen, or even larger. The Form window for a Code list, for example, is likely to be smaller than the window for a "main" Form.

The designer uses the Information engineering and Design Facility (IDF) to define the size and position of the Form window. Form windows can overlap in order to show several Forms at the same time.

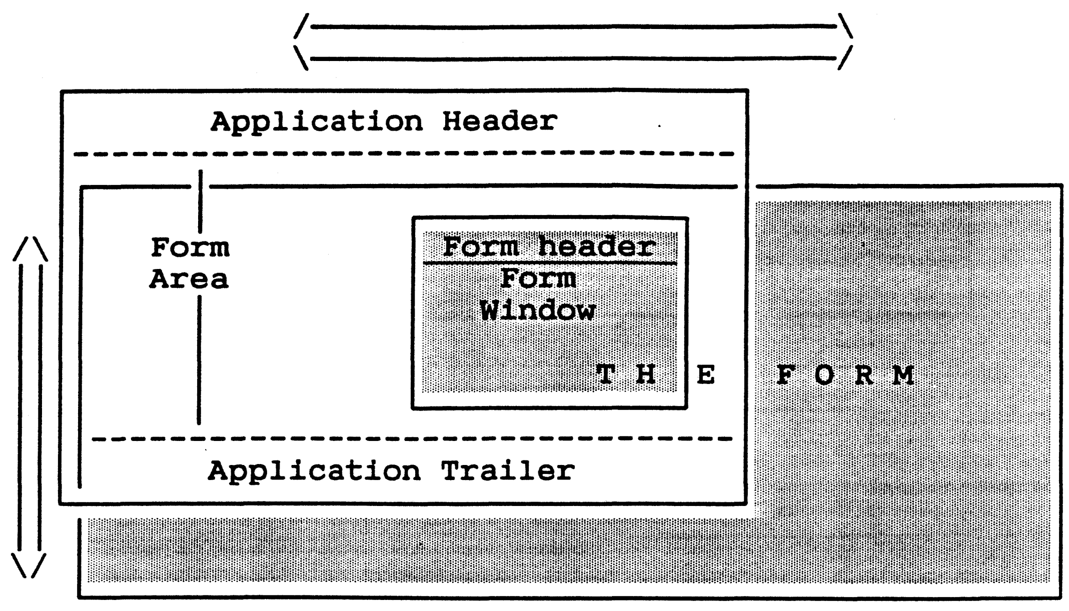


Figure 2-2. The relationship between the Screen and the Form.

The Form Travel functions of the Structure Editor allow the user to move freely across a Form. The cursor movement keys or a mouse move the cursor over the entire Form window as if it were a word processing screen. If the Form is larger than its Form window, these functions scroll the Form window horizontally or vertically over the Form.

Uniface data structure concepts

Before describing how a Form uses Frames to present a data structure, it is necessary to first explain how some data structure terms are used in Uniface...

A field corresponds to a column in a relational database. A Uniface field holds information such as a telephone number, a name, or a formatted document.

An entity is a distinguishable object represented in the database. It generally corresponds to a table in a relational database or a file in other types of systems.

A subtype entity is an alias which refers to another entity. A subtype can have its own definition and relationships, but refers to the fields and keys of its corresponding supertype.

An occurrence is one set of actual values for the fields in an entity. It corresponds to a row in relational database terms.

Data element	Synonyms	Frame
field	column, data item, attribute	Field Frame
entity	table, file, view, group, relation	Entity Frame
subtype entity	role, alias, subset	Entity Frame
entity occurrence	row, record, tuple	Entity Frame

Figure 2-3. Uniface Data Structure Concepts.

It is important to remember that...

... the structure of an external schema can be completely different from the way the data is stored in the database...

... a Form can use data from any type of database or several different databases at the same time...

... Uniface automatically performs data structure conversion when storing or retrieving data.

The Form and Frames

Uniface uses Frames to present the data elements discussed above. Each data element has a corresponding Frame which actually presents it on a Form (Fig.2-4). Each Frame can condense the presentation of its data element by windowing.

A Frame is a window over a field or entity. The user works with Frames, which can be smaller than the logical data dimensions. The designer defines the size and position of each Frame to present the data in the way which is most convenient for the user.

Figure 2-4 illustrates three examples of how a Field Frame can condense data presentation by not showing all characters or lines. If the field contains a long string, a Field Frame can show the first 15 characters, or the first 100, etc. If the field contains a document (which is actually a formatted string), the Field Frame can be as large as the designer decides is most effective.

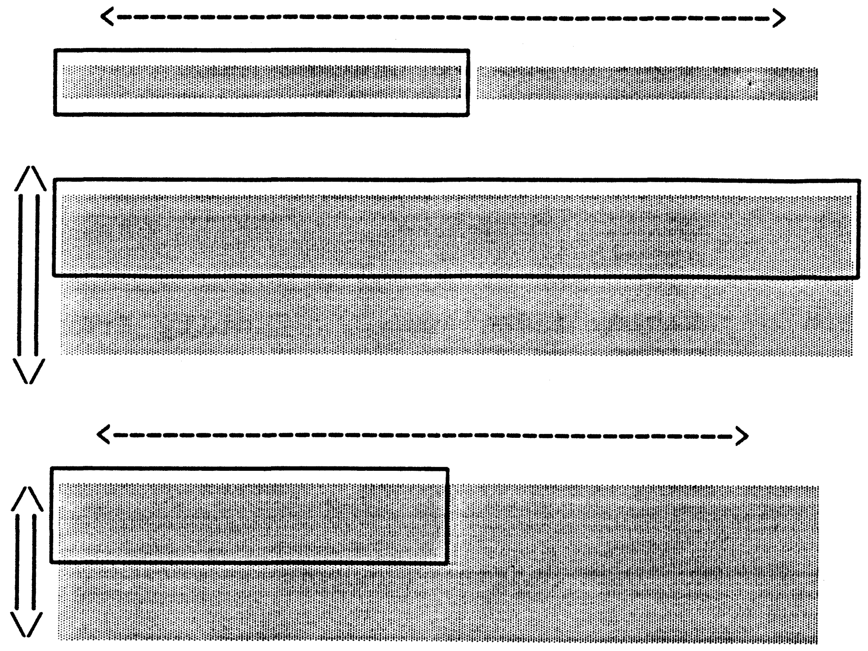


Figure 2-4. Field Frames.

Frame Travel & Edit functions allow the user to browse through and edit data in Frames. The Structure Editor contains a complete text processor for editing text within a Field Frame. See the *Uniface User Interface Guide* for a complete discussion of the Structure Editor functions. The user can scroll a Field Frame horizontally or vertically over the field or entity, or use the ZOOM function to make the Frame temporarily larger and work with more of the field.

An <u>Entity Frame</u> condenses data presentation by not showing all occurrences. The designer decides how many occurrences of an entity to show in a Entity Frame at one time. Figure 2-5 shows two examples of how Entity Frames can present occurrences.

In the first example of Figure 2-5, occurrences are arranged like a table. Each row is an occurrence, and each column is a field. The user can scroll through the visible and hidden occurrences of this table with the NEXT/PREVIOUS OCCURRENCE function. ADD and REMOVE OCCURRENCE allow the user to edit occurrences.

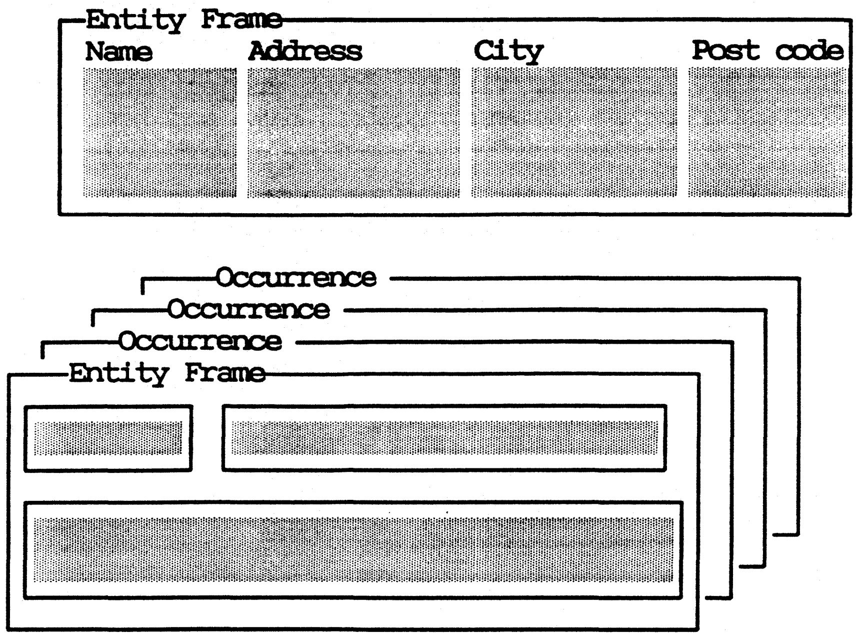


Figure 2-5. Entity Frames.

The second example illustrates an Entity Frame which shows one occurrence at a time. Other occurrences are available, but hidden. The user browses through occurrences "deeper" in the Form with the NEXT/PREVIOUS OCCURRENCE function.

Remember that the designer "paints" the Form and the data structure at the same time. A Field or Entity Frame inside an Entity Frame indicates a relationship. Uniface understands this relationship (using the information in the conceptual schema) and ensures that the data in the Form remains consistent at all times.

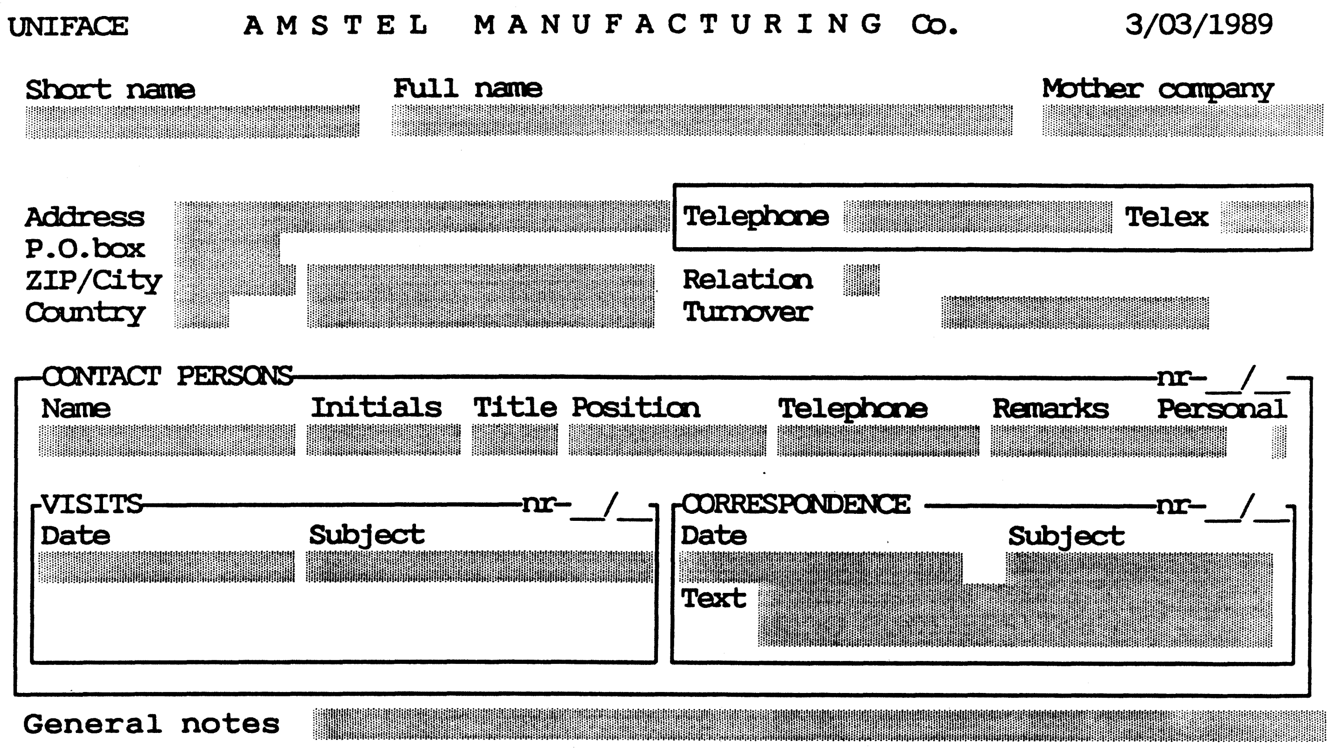
An example Form

This section illustrates the concepts described above using a Form from the Uniface demonstration (DEMO) application, which is included with distribution tapes. In the interest of clarity, this Form has been made rather simple. In an actual application, the Form would probably contain more Field Frames and function slightly differently.

Figure 2-6 illustrates the main Form for the customer information application. This Form is the template for the dialogue between the user and customer information in the database. The user can perform all the processing required for a transaction from this Form.

The application contains...

1. General company information: name, address, relation type etc.
2. Contact person information: name, address, position, notes
3. Individual contacts/visits: date, subject etc.
4. Correspondence: document number, date and text of each letter



UNIFACE A M S T E L M A N U F A C T U R I N G Co. 3/03/1989

Short name Full name Mother company

Address
P.O.box
ZIP/City
Country

Telephone Telex
Relation
Turnover

CONTACT PERSONS nr- / -
Name Initials Title Position Telephone Remarks Personal

VISITS nr- / -
Date Subject

CORRESPONDENCE nr- / -
Date Subject
Text

General notes

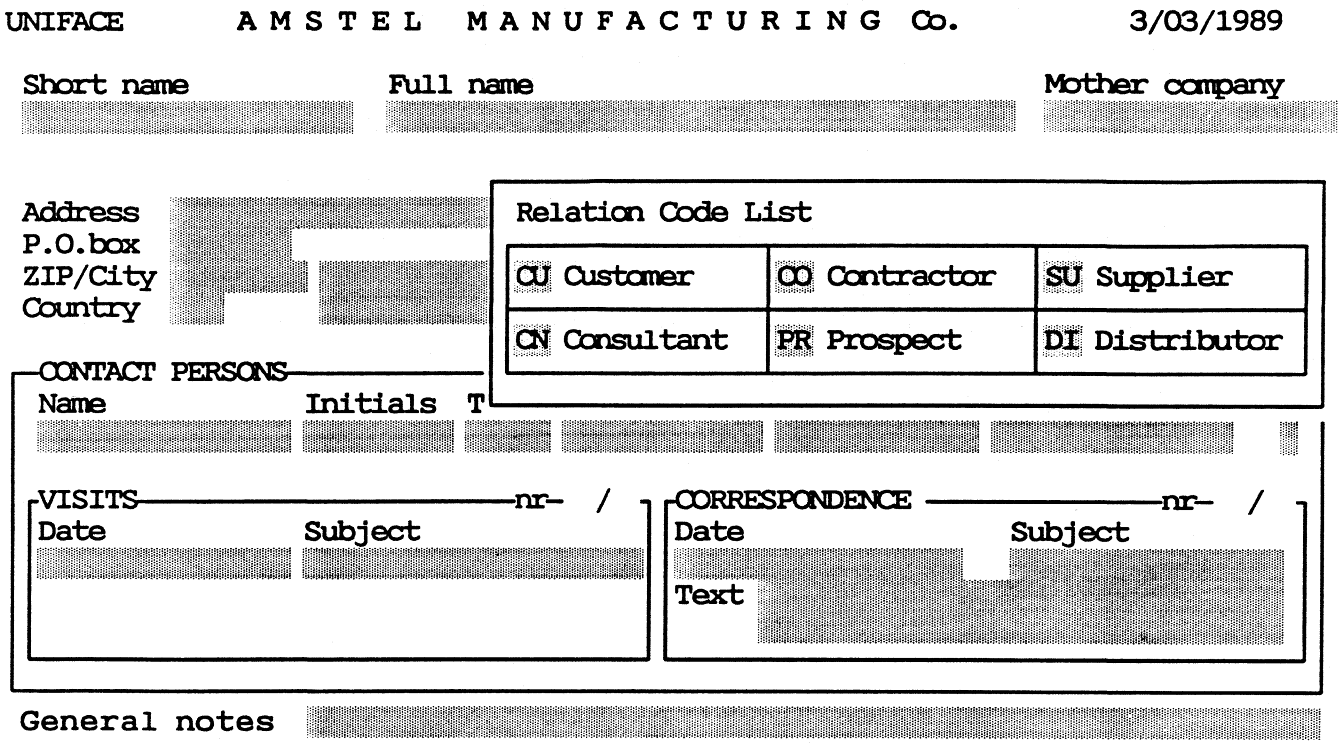
Figure 2-6. An example Form.

The top part of the Form contains general company information, belonging to the entity COMPANY. There are Field Frames for the abbreviated and full names of the company, address, turnover etc.

A complete range of text processing functions, which are part of the Structure Editor, are available when the cursor is in any field. These are described later in this chapter.

A code list Form

The designer can define a Form for the Relation field which functions as a code list (see Figure 2-8). When the user presses DETAIL with the cursor in this Frame, a Code list Form appears as an overlay which explains the codes which are available for the field.



UNIFACE   A M S T E L   M A N U F A C T U R I N G   Co.   3/03/1989

Short name   Full name   Mother company

Address
P.O.box
ZIP/City
Country

Relation Code List

CU Customer	CO Contractor	SU Supplier
CN Consultant	PR Prospect	DI Distributor

CONTACT PERSONS
Name   Initials   T

VISITS   nr- /
Date   Subject

CORRESPONDENCE   nr- /
Date   Subject
Text

General notes

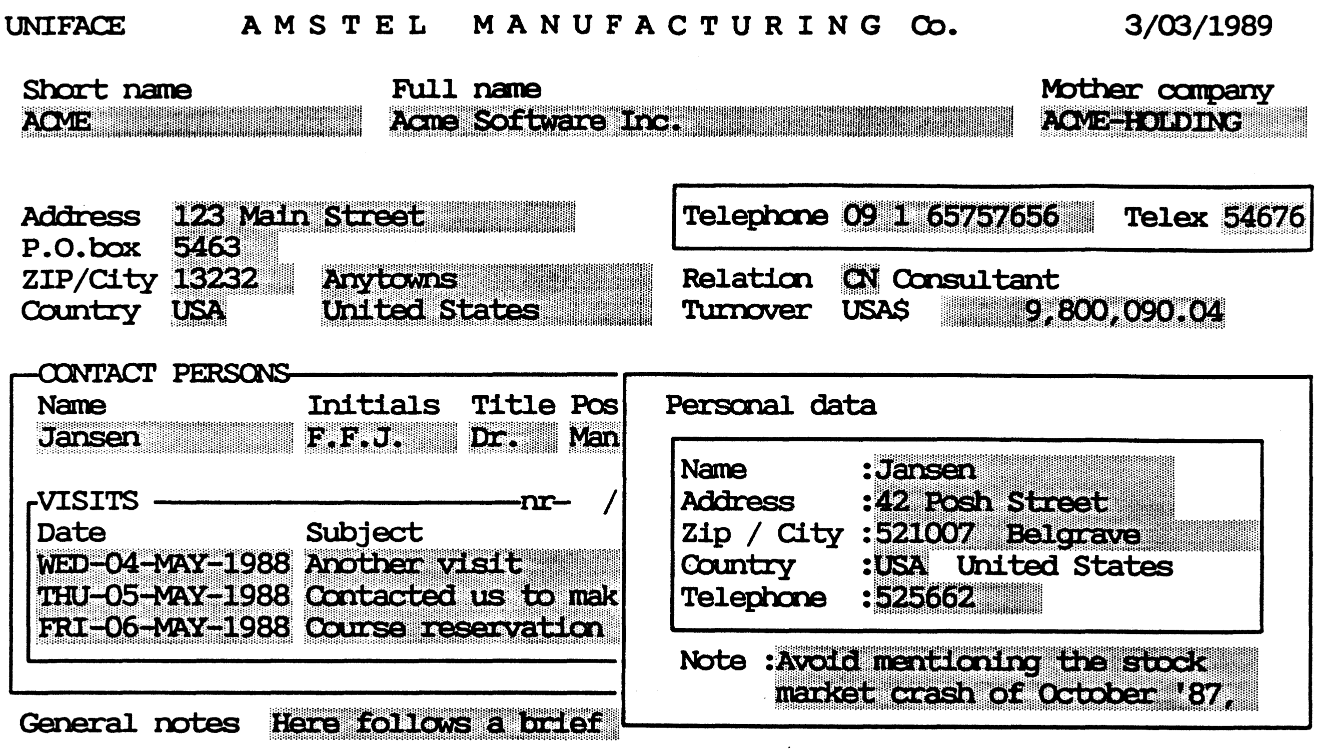
Figure 2-8. A code list Form.

The user ACCEPTS a code selected with the cursor, and returns to the main form, taking the code back to it.

A detail Form

The entity CONTPERS is in the middle of the Form (written out in full as "CONTACT PERSONS" in the fixed text).

To see more detailed information about a particular contact person, the user moves the cursor to the "Private" (i.e. private data) field and presses DETAIL. A Detail Form is overlaid on the main Form as shown in Figure 2-9. You are referred to the I/O chapter to find out where the data on this form came from, and how it got there.

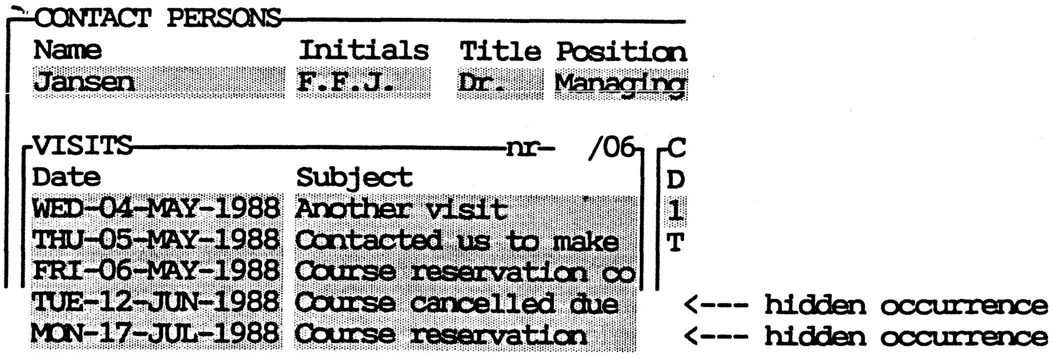


```
UNIFACE      A M S T E L   M A N U F A C T U R I N G  Co.      3/03/1989

Short name          Full name                        Mother company
ACME                Acme Software Inc.               ACME-HOLDING

Address  123 Main Street              Telephone 09 1 65757656   Telex 54676
P.O.box  5463
ZIP/City 13232  Anytowns              Relation  CN Consultant
Country  USA    United States         Turnover  USA$        9,800,090.04

CONTACT PERSONS
Name              Initials  Title Pos   Personal data
Jansen            F.F.J.    Dr.   Man
                                        Name        :Jansen
VISITS ------------------------nr-  /   Address     :42 Posh Street
Date              Subject               Zip / City  :521007  Belgrave
WED-04-MAY-1988   Another visit         Country     :USA  United States
THU-05-MAY-1988   Contacted us to mak   Telephone   :525662
FRI-06-MAY-1988   Course reservation
                                        Note :Avoid mentioning the stock
                                              market crash of October '87,
General notes  Here follows a brief
```

Figure 2-9. A detail Form.

Scrolling

There are two Entity Frames side by side at the bottom of the Form. The right Entity Frame shows information about correspondence, one occurrence at a time. The left one shows information about VISITS, three occurrences at a time. The user can **scroll** these **entities** to see hidden **occurrences** using NEXT/PREVIOUS OCCURRENCE.



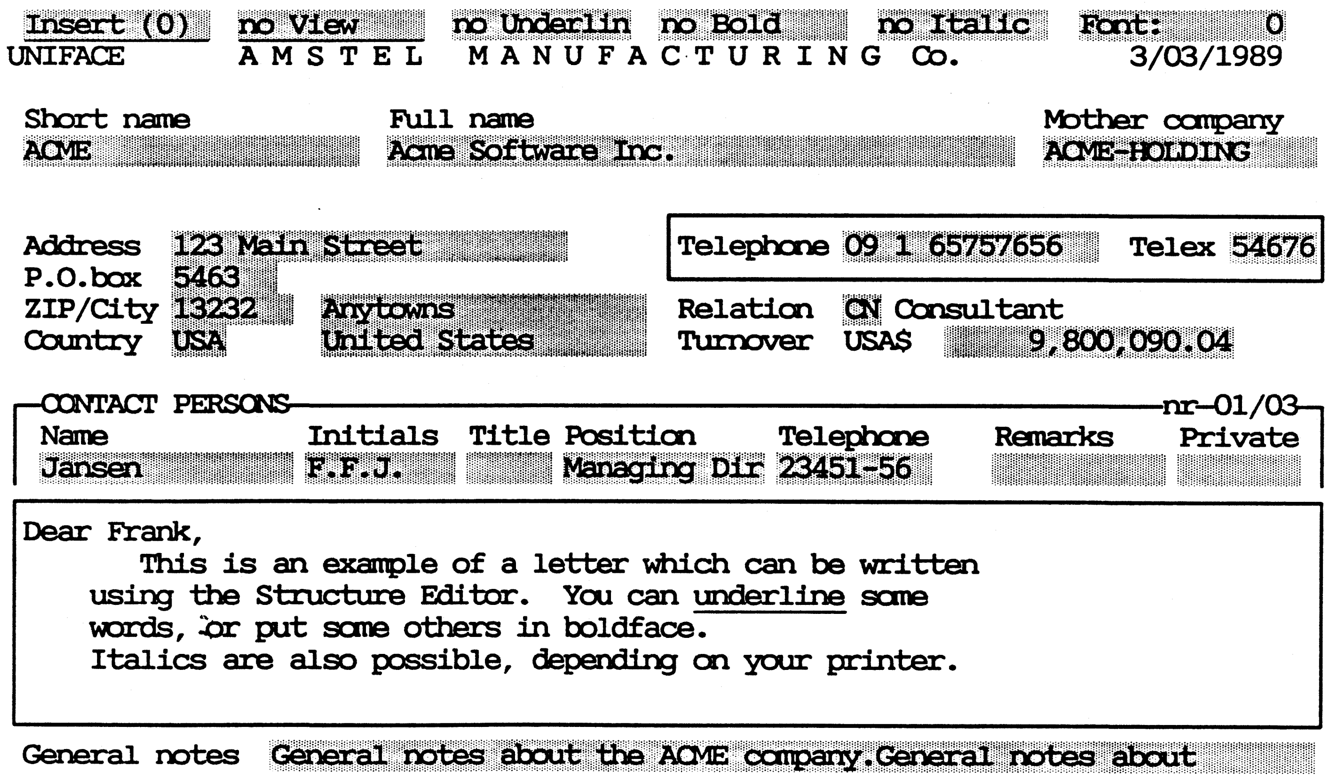
```
CONTACT PERSONS
Name              Initials  Title Position
Jansen            F.F.J.    Dr.   Managing

VISITS-------------------------nr-  /06  C
Date              Subject                D
WED-04-MAY-1988   Another visit          1
THU-05-MAY-1988   Contacted us to make   T
FRI-06-MAY-1988   Course reservation co
TUE-12-JUN-1988   Course cancelled due      <--- hidden occurrence
MON-17-JUL-1988   Course reservation        <--- hidden occurrence
```

Figure 2-10. Entities which can be scrolled.

Zooming

The last entity on the Form shows correspondence to the customer. The Entity Frame shows one occurrence (an item of correspondence e.g. fax or letter) at a time. Other occurrences are available, but hidden.

When editing the text of the document, the user can press ZOOM on the **Field Frame** to show the entire **field contents.** Pressing QUICK ZOOM makes the Frame as large as the entire screen. In Figure 2-11 below, the Text field of the CORRESPONDENCE entity has been ZOOMed. The Character Attributes Form, which allows the user to put text in bold or italic or to underline it, is also shown.



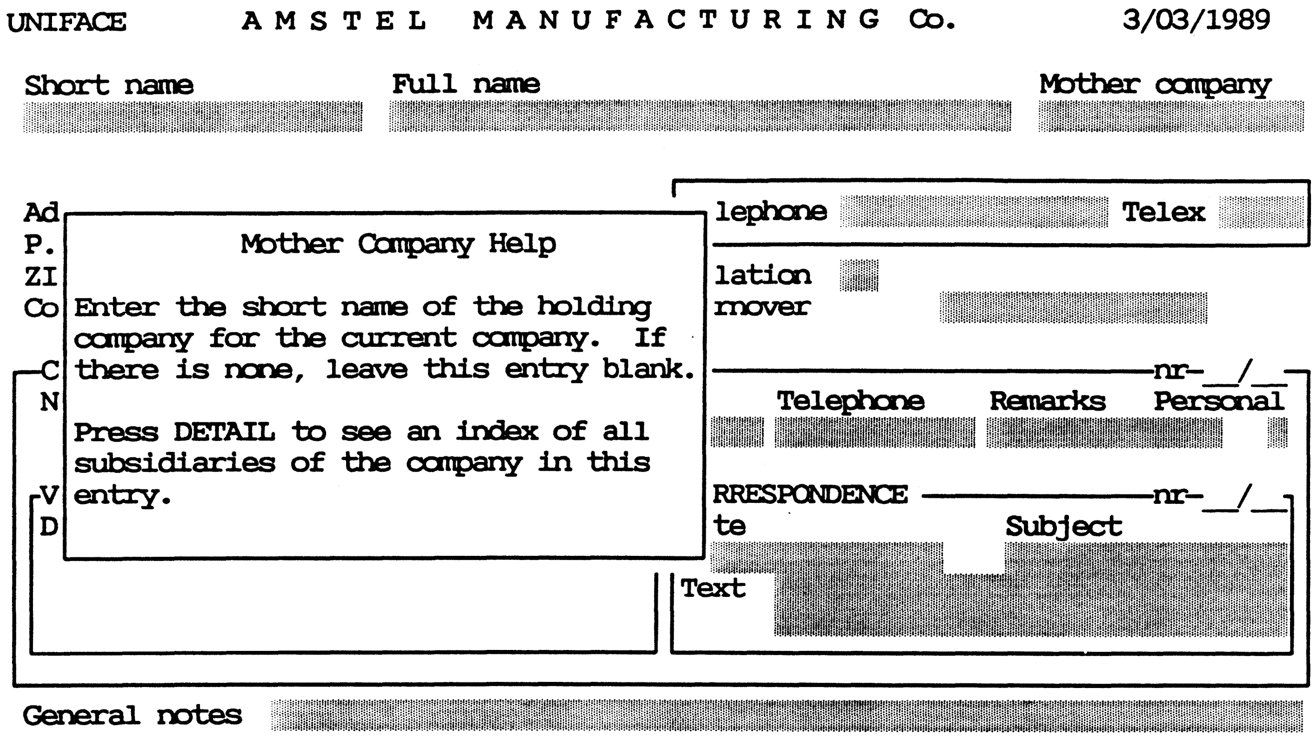
```
 Insert (0)   no View    no Underlin  no Bold     no Italic   Font:     0
UNIFACE       A M S T E L   M A N U F A C T U R I N G  Co.      3/03/1989

 Short name               Full name                          Mother company
 ACME                     Acme Software Inc.                 ACME-HOLDING

 Address  123 Main Street              Telephone 09 1 65757656   Telex 54676
 P.O.box  5463
 ZIP/City 13232    Anytowns            Relation  CN Consultant
 Country  USA      United States       Turnover  USA$       9,800,090.04

CONTACT PERSONS                                                    nr-01/03
 Name            Initials  Title Position     Telephone    Remarks  Private
 Jansen          F.F.J.          Managing Dir 23451-56

Dear Frank,
       This is an example of a letter which can be written
   using the Structure Editor.  You can underline some
   words, or put some others in boldface.
   Italics are also possible, depending on your printer.

General notes  General notes about the ACME company.General notes about
```

Figure 2-11. A ZOOMED Field Frame with the Character Attributes Form above.

The options INSERT/OVERSTRIKE mode and VIEW mode on the Character Attributes Form are not strictly character attributes but have been included on this form for the convenience of the user.

On-line help

On-line help is available on the "Mother company" (MOTCO) Field Frame. It is obtained by pressing the HELP key when the cursor is in this Frame and contains explanatory information.



```
UNIFACE      A M S T E L   M A N U F A C T U R I N G  Co.      3/03/1989

Short name              Full name                              Mother company

Ad                                          lephone               Telex
P.              Mother Company Help
ZI                                          lation
Co Enter the short name of the holding      mover
   company for the current company.  If
-C there is none, leave this entry blank.   ---------------------nr-__/__-
 N                                             Telephone   Remarks   Personal
   Press DETAIL to see an index of all
   subsidiaries of the company in this
-V entry.                                   RRESPONDENCE -----------nr-__/__-
 D                                          te                Subject

                                            Text

General notes
```

Figure 2-7. On-line help.

The designer can specify help text for every Frame. All standard word processing functions e.g. boldface, italic and underline are available when defining it. By using the central message and help text facility, help can be offered in several different languages within an application.

QUIT or ACCEPT returns you to the main Form.

National language support

The central text facility makes it easy for users to define which language they prefer for messages and help text. This facility is also used for messages and help text in the IDF and the Uniface run time system. Thus Uniface is set up to offer natural language support in an elegant way. Documentation, on-line help and messages are presently available in English. Other languages will be made available on request.

Structure Editor functions

A user edits structured data with the Uniface Structure Editor as easily as s/he edits text with a word processor. The user can browse or remove occurrences in the same way s/he scrolls a document or removes a paragraph with a word processor. This enhances ease of use considerably. The same functions are used to perform the same operations when working with a character, a word, field, an occurrence, or occurrences of several related entities.

The Structure Editor operates in a standard way in all Uniface applications, i.e. the same commands perform the same tasks in any external schema in any application. The Structure Editor has four types of functions:

- Form Travel
- Frame Travel and Edit
- Input/Output (I/O) Control
- Session Control

These functions are assigned to different function keys depending on the type of terminal used. A sample keyboard arrangement for a DEC VT200 terminal is shown in Figure 2-12.

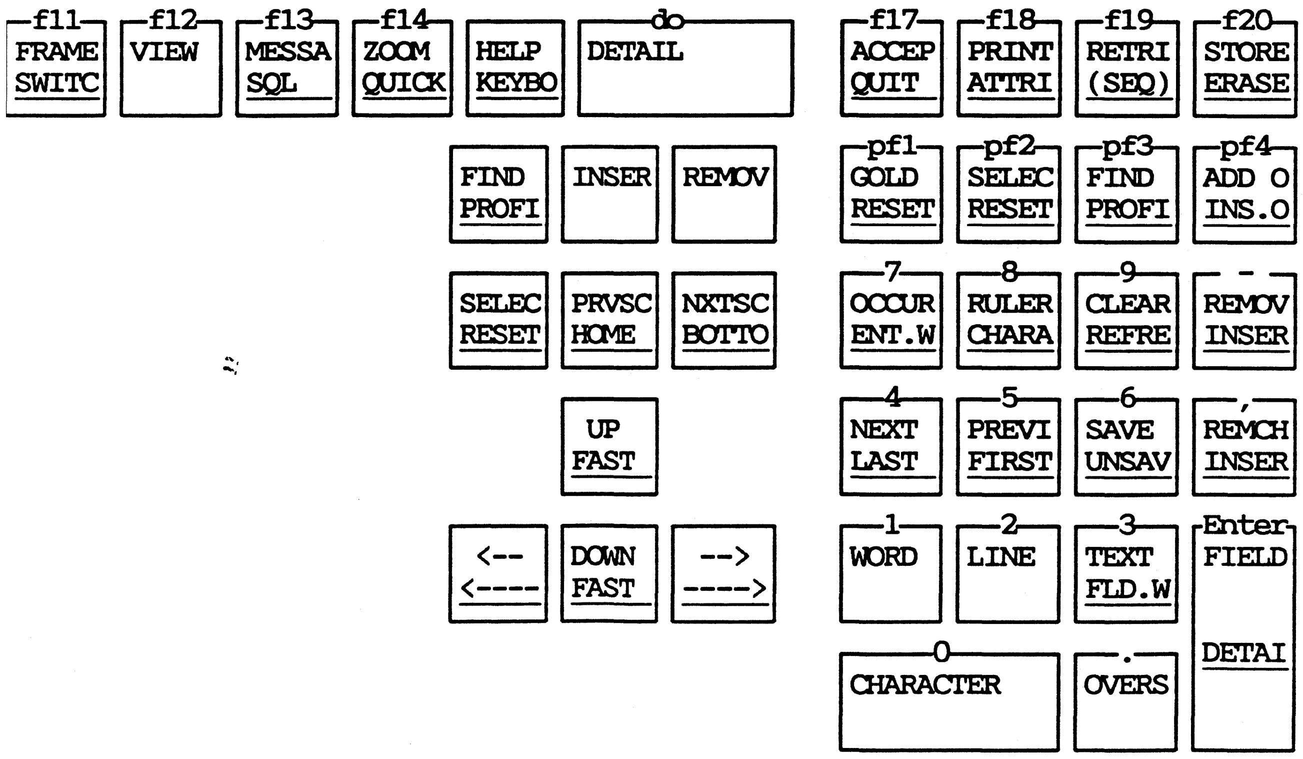


Figure 2-12. VT200 Keyboard Layout of Structure Editor Functions.

Although this arrangement has proven to be convenient for most tasks, key assignments can easily be changed to suit specific purposes. Of course, other keyboard arrangements will be offered for implementations on other hardware systems. The *Uniface User Interface Guide* contains a complete explanation of how to use these functions.

Form travel functions...

...are used to move across the entire Form and to scroll the Form window over the Form. The ordinary cursor movement keys and a few special function keys control these functions. Form travel is similar to moving the cursor across a word processing screen. The user moves anywhere within the Form window and not necessarily to the NEXT or PREVIOUS FIELD.

Frame travel and edit functions...

...are used to move through data in a Form in a structured way, i.e. to the NEXT/PREVIOUS FIELD or OCCURRENCE. The edit functions are used to edit data within Field and Entity Frames. In Field Frames, most editing functions are text editing commands. All the facilities of a full-screen WYSIWYG (What You See Is What You Get) word processor are inherent in the Structure Editor; it is not necessary to bring in an external text editor program to offer these functions. Boldface, underline, italics, different fonts, for instance, are all available.

The same commands are used for editing data and text. The user combines commands to REMOVE a WORD or ADD an OCCURRENCE, for example. The NEXT/PREVIOUS function operates as a mode; once NEXT is pressed, it functions implicitly until PREVIOUS is pressed.

To browse through occurrences, the user presses either the NEXT or PREVIOUS key and then the OCCURRENCE key. Uniface displays all the data for the next occurrence of the current Entity Frame. The function keys are context-sensitive in that Uniface knows which entity or field the user wishes to edit by the cursor position.

Other functions allow the user to move to the NEXT/PREVIOUS FIELD or to the MESSAGE FRAME. S/He can ZOOM a Field Frame to work with a larger Frame, or press QUICK ZOOM to use the entire screen as the Field Frame. Text processing functions include such facilities as REMOVE WORD, NEXT LINE and FIND [profile].

I/O functions...

...control the flow of information between the external structure and the database, i.e. between temporary and permanent storage. Uniface also takes care of concurrency control in a multi-user environment, with support for transaction definition facilities such as "commit" and "rollback".

The RETRIEVE function uses the Query-By-Form method to select data from the database. The user can select all occurrences in the DBMS, fill in a field on the outermost entity of the Form and press RETRIEVE. This field does not need to be the primary key. Uniface builds a hitlist of the selected occurrences and moves them into the external structure as requested by the user with NEXT OCCURRENCE.

Uniface retrieves data in stages when requested by the user with Travel and Edit functions (See the I/O chapter for an explanation of data-staging). The user presses NEXT OCCURRENCE in the outermost entity to browse through the hitlist of occurrences selected with RETRIEVE. S/He uses NEXT OCCURRENCE in other Entity Frames to browse through the occurrences of these entities.

The ERASE function is used to delete all the data in the external structure from the database. Depending on the referential constraints defined in the conceptual schema, the occurrences of all related entities are also automatically deleted.

The STORE function instructs Uniface to send all changes made by the user in the external structure to the database. Uniface analyzes the data in the Form to see which occurrences have been modified in the current Form edit session, i.e. removed, added or changed. Each modified occurrence is updated in the database.

The PRINT function produces a hard copy of the current Form. Because a hard copy obviously cannot reproduce the "scrolling" effect used to access hidden occurrences Uniface expands Entity Frames to show all hidden occurrences and expands Field Frames to show hidden characters. See the section "Printing and Reports" in Chapter three.

Session control functions...

...control how the user enters or leaves a Form. A Uniface application is meant to operate the same way people think. As a user works with data, s/he sometimes needs to query the database, obtain more detailed data, ask for help about a particular item, etc.

In a Uniface application, the user presses DETAIL to move forward in an application to a new external schema. An overlay Form appears, and when finished with that Form, the user presses QUIT or ACCEPT to return to the previous external schema. The overlay Form disappears, leaving the previous Form as it was originally.

In a comprehensive application, the user might move through several external schemas, each serving a different purpose. Forward movement is determined by data context, i.e. which Frame on the Form is your starting point. There can, therefore, be several "next" external schemas to choose from for a particular Form.

DETAIL is the main function which controls forward movement. One use of DETAIL is to defer part of a structure to a Detail Form. In the example above, the user presses DETAIL to see an external schema with more information about customer contacts. This information could have been included on the main Form. However, because it is not normally needed, the data was deferred to a Detail Form. Because the external schemas are recursive, another external schema can be called from the second selected external schema, and so on. This procedure can be repeated many times. Other uses of the DETAIL key are to select a menu option, to see a list of codes for a field, or to perform a parallel database query, i.e. an Index Form.

Only a few uses of DETAIL have been described in this section. The designer is provided with a great deal of flexibility, and imaginative use of this can result in truly innovative applications.

Backward movement...

...is controlled with the QUIT and ACCEPT functions. Although there can be many "next" external schemas, there is always only one "previous" external schema, i.e. the one used immediately before the current one. Uniface stacks the external schemas used in a session to retain the data and cursor position of each external schema. The user presses QUIT or ACCEPT when s/he is finished with an external schema to return to the previous one.

In the example, when the user presses ACCEPT or QUIT after editing contact information for a customer, s/he returns to the "main" Form. The cursor returns to the same position it occupied when the user left the Form; all other Frames on the main Form are exactly as they were when s/he pressed DETAIL.

The ACCEPT function leaves the current Form and accepts any changes. The user returns to the previous Form, retaining any editing changes. The QUIT function leaves the current Form and ignores any changes made. The user returns to the previous external schema, abandoning any editing changes.

This section has described the most common uses of the session control commands. Of course, the designer is free to define the application in any way s/he wishes. For example, in some applications the STORE and ACCEPT functions could be combined.

Chapter 3. Defining External Schemas

This chapter discusses how to define the external schemas described in the last chapter.

Uniface does not generate specialized software to execute specific processing tasks. All processing in a Uniface application is defined entirely by constraints specified as extended data definitions, rather than sequential programs. The application designer uses the IDF to specify these definitions, which are held in the application dictionary. These definitions specify literally the entire application.

The IDF uses these definitions to generate high-performance tables. At application run time, the Uniface run time system loads these tables into virtual working memory and uses them to implement the dialogue between the user and the DBMS. Note that Uniface is not a 3GL code generator; the definitions provided by the designer and system administrator are used to direct the run time system.

The structure of the IDF follows the division of tasks between the application designer and the system administrator. The IDF contains a system administrator's workbench to define the conceptual schema and a designer's workbench to define external schemas. Both these workbenches access the definitions in the application data dictionary. The IDF is discussed in another chapter.

Most of the definitions discussed in this chapter are actually defined by the system administrator. As s/he defines fields and entities in the conceptual schema, the system administrator centrally specifies values which are used in every external schema which contain the field or entity. The system administrator retains exclusive control of some definitions. Others can be enhanced and modified by designers to meet the needs of individual transactions. Designers use these values as defaults to provide a starting point as they paint external schemas.

Data-driven and event-triggered processing

Constraints processing in Uniface applications is determined by definitions for each field and entity. Data drives the processing. Unlike conventional applications where processing is a separate operation performed on static data, processing definitions in Uniface are inherent in the external and conceptual data structures. Most of these constraints are defined in the conceptual schema, and then locally enhanced for individual external schemas.

Field and entity definitions...

...determine all the properties, constraints, etc. of the data handled in the transaction. The definitions of these properties can be specified in a declarative format (e.g. the data type is "string" or the fixed length is "40") or as processing specifications (Procs). Procs are triggered by events; a Proc is activated when the trigger with which it is associated occurs. This means that processing is defined as data definitions, and activated by events from the user interface.

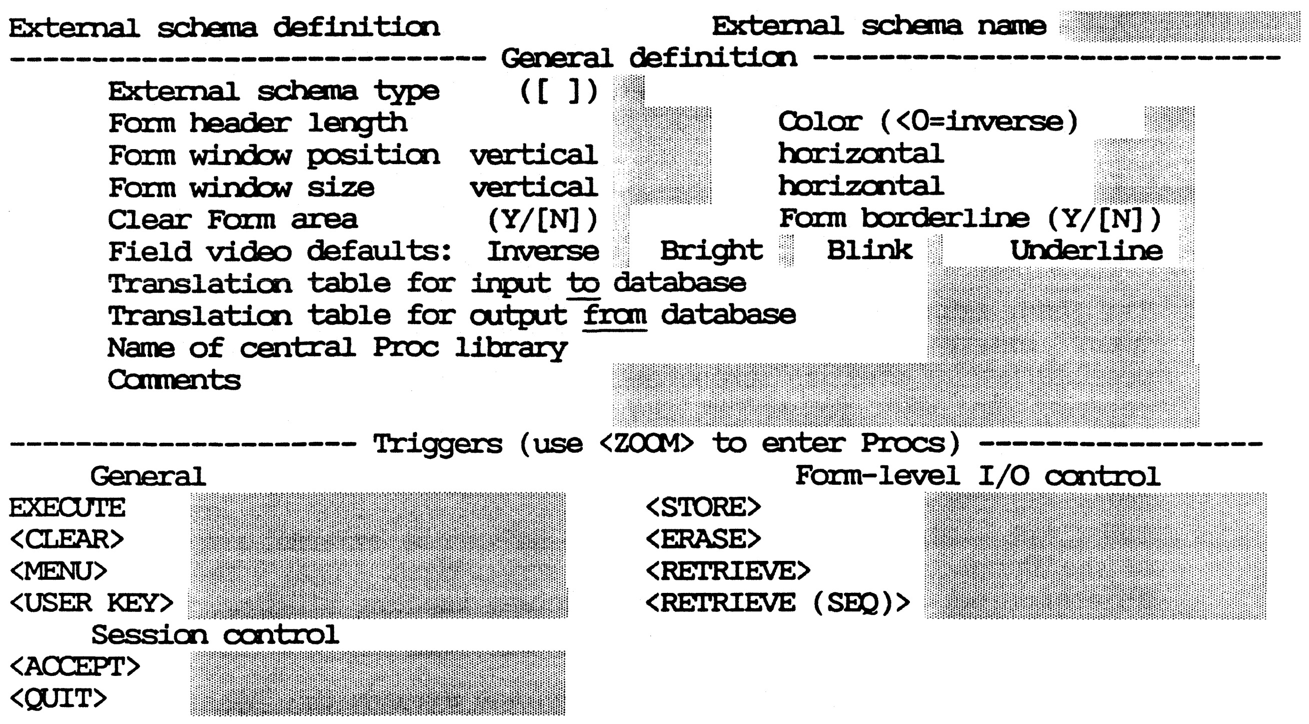
For example, the designer can define a Proc to be activated when the user presses DETAIL, or another Proc for when the user makes the first modification in a Field Frame. A specific Proc is triggered when a specific event occurs in the current Frame.

Triggers are activated either directly or indirectly from the user interface. Pressing a function key provides a direct event trigger, e.g. DETAIL. Triggers which correspond to a function key in the user interface are marked with brackets on the Forms in the IDF, e.g. <STORE>. Triggers can also be activated indirectly as part of an I/O operation, as discussed later in this brochure. Uniface provides about 40 triggers which allow the designer to specify exactly what should happen during the transaction session. Triggers are available at field, entity, form and application-level.

Form-level definitions

An external schema includes the visible Form and all the processing and specifications which define the transaction. Therefore, an external schema definition includes the painted Form, Form-level definitions and field or entity definitions. Form-level definitions include size and position of the Form window, type of external schema, borderlines, etc. The Form-level definitions are shown in Figure 3-1.

In addition to the declarative definitions supplied on the upper part of this Form, there are three classes of triggers at Form-level for defining processing specifications (Procs)...



```
External schema definition                 External schema name
---------------------------- General definition ----------------------------
      External schema type     ([ ])
      Form header length                         Color (<0=inverse)
      Form window position  vertical              horizontal
      Form window size      vertical              horizontal
      Clear Form area         (Y/[N])              Form borderline (Y/[N])
      Field video defaults:  Inverse   Bright   Blink     Underline
      Translation table for input to database
      Translation table for output from database
      Name of central Proc library
      Comments

-------------------- Triggers (use <ZOOM> to enter Procs) ----------------
     General                                    Form-level I/O control
EXECUTE                                     <STORE>
<CLEAR>                                     <ERASE>
<MENU>                                      <RETRIEVE>
<USER KEY>                                  <RETRIEVE (SEQ)>
     Session control
<ACCEPT>
<QUIT>
```

Figure 3-1. Form-level Definitions.

General triggers

A Proc associated with the EXECUTE trigger, for example, is activated when the external schema is started. This trigger can be used retrieve data into the Form, position the cursor on a specified field, etc.

A Proc associated with the <CLEAR> trigger defines the processing which occurs when the user presses the CLEAR key, a Proc defined in the <MENU> trigger is activated when the user presses the MENU key, etc.

The <USER KEY> trigger can define a new function key.

Session control triggers

Session control at Form-level occurs when an external schema ends execution, i.e. when the user presses QUIT (dropping any modifications made) or ACCEPT. Procs associated with these triggers define how the user moves to the previous external schema.

Form-level I/O control triggers

I/O triggers contain the Procs which determine what happens when the user presses ERASE, STORE etc. The Uniface I/O functions are discussed later in this brochure.

The Form Paint Tableau

The Paint tableau, as the name suggests, is an area where designers "paint" external structures. The makers of Uniface have extended the possibilities for creative screen design, while at the same time ensuring that what you see on the screen is a faithful visual representation of the logical data model.

Designers use the Paint Tableau to define not only how the Form *looks*, but - more importantly - to also define the external *structure* of the Form. Its beauty is more than screen-deep. That's the big difference between Uniface and most screen handling utilities; you paint *structure* as well as *appearance*.

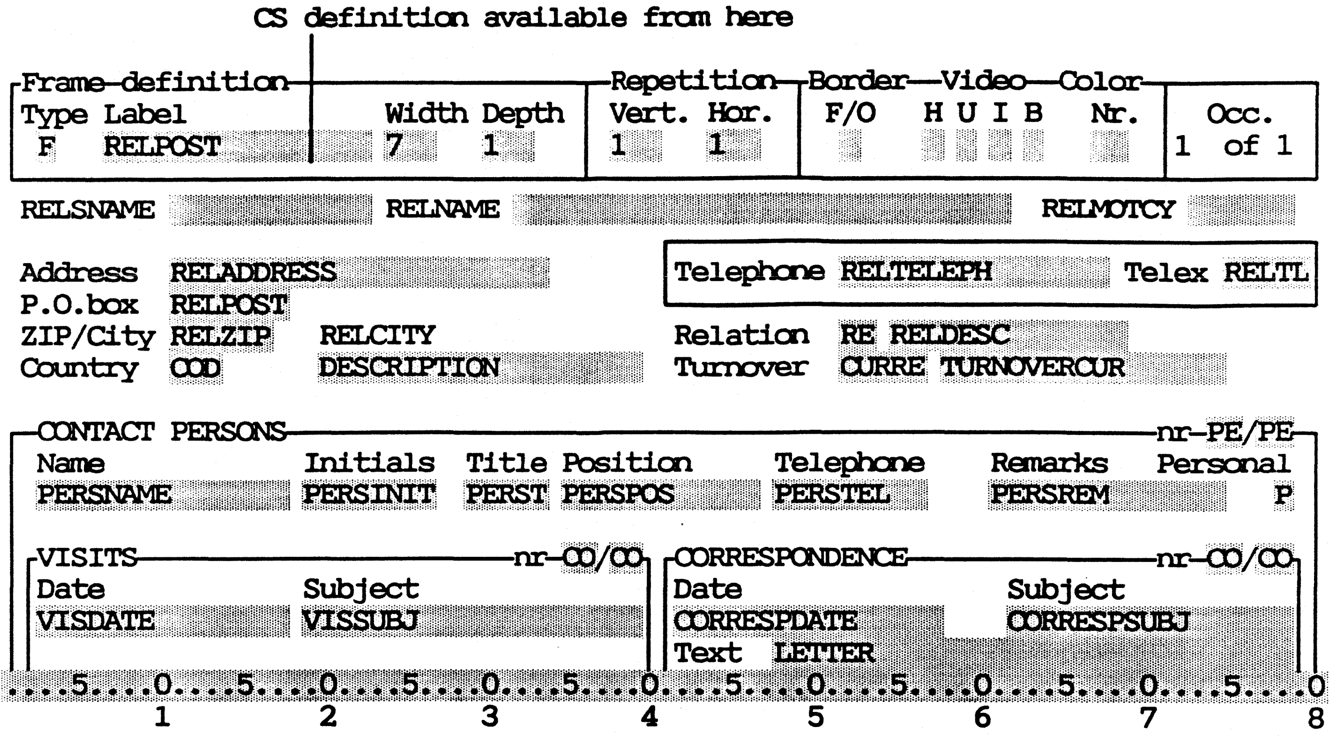
Another way of looking at it is this: Uniface extends the "What You See Is What You Get" concept into another dimension - "What You See Is What You Have Defined".

The Frame Function

Frames are positioned with the FRAME function, as shown in Figure 3-2. The designer moves the cursor to the desired spot, and presses FRAME to place either an Entity or Field Frame there.

When a designer places an Entity Frame named "CONTPERS" (short for "Contact Persons") inside of the Entity Frame named "COMPANY", s/he not only places the entity in a particular location, s/he also tells Uniface that there is some kind of relationship between the two entities. The position has meaning which Uniface understands. The conceptual schema contains a definition of exactly what that relationship is.

The relative position of Frames on the Paint Tableau is all Uniface needs to understand an external structure. Of course, a structure alone is not enough to define a complete application. Processing must be incorporated in the structure in the form of constraints. Processing definitions (both in declarative and processing specification format) are defined with the structure in field and entity definitions.



```
                CS definition available from here
-Frame-definition---------------------Repetition-Border---Video---Color---------
 Type Label              Width Depth  Vert. Hor.   F/O    H U I B    Nr.    Occ.
  F   RELPOST             7     1     1     1                              1 of 1

RELSNAME                 RELNAME                                    RELMOTCY

Address    RELADDRESS                      Telephone RELTELEPH         Telex RELTL
P.O.box    RELPOST
ZIP/City   RELZIP     RELCITY              Relation  RE RELDESC
Country    COD        DESCRIPTION          Turnover  CURRE TURNOVERCUR

-CONTACT PERSONS-----------------------------------------------------------nr-PE/PE-
 Name             Initials  Title Position      Telephone    Remarks    Personal
 PERSNAME         PERSINIT  PERST PERSPOS       PERSTEL      PERSREM           P

 -VISITS---------------------------nr-CO/CO- -CORRESPONDENCE----------------nr-CO/CO-
 Date             Subject                    Date                  Subject
 VISDATE          VISSUBJ                    CORRESPDATE           CORRESPSUBJ
                                             Text  LETTER
....5....0....5....0....5....0....5....0....5....0....5....0....5....0....5....0
         1         2         3         4         5         6         7         8
```

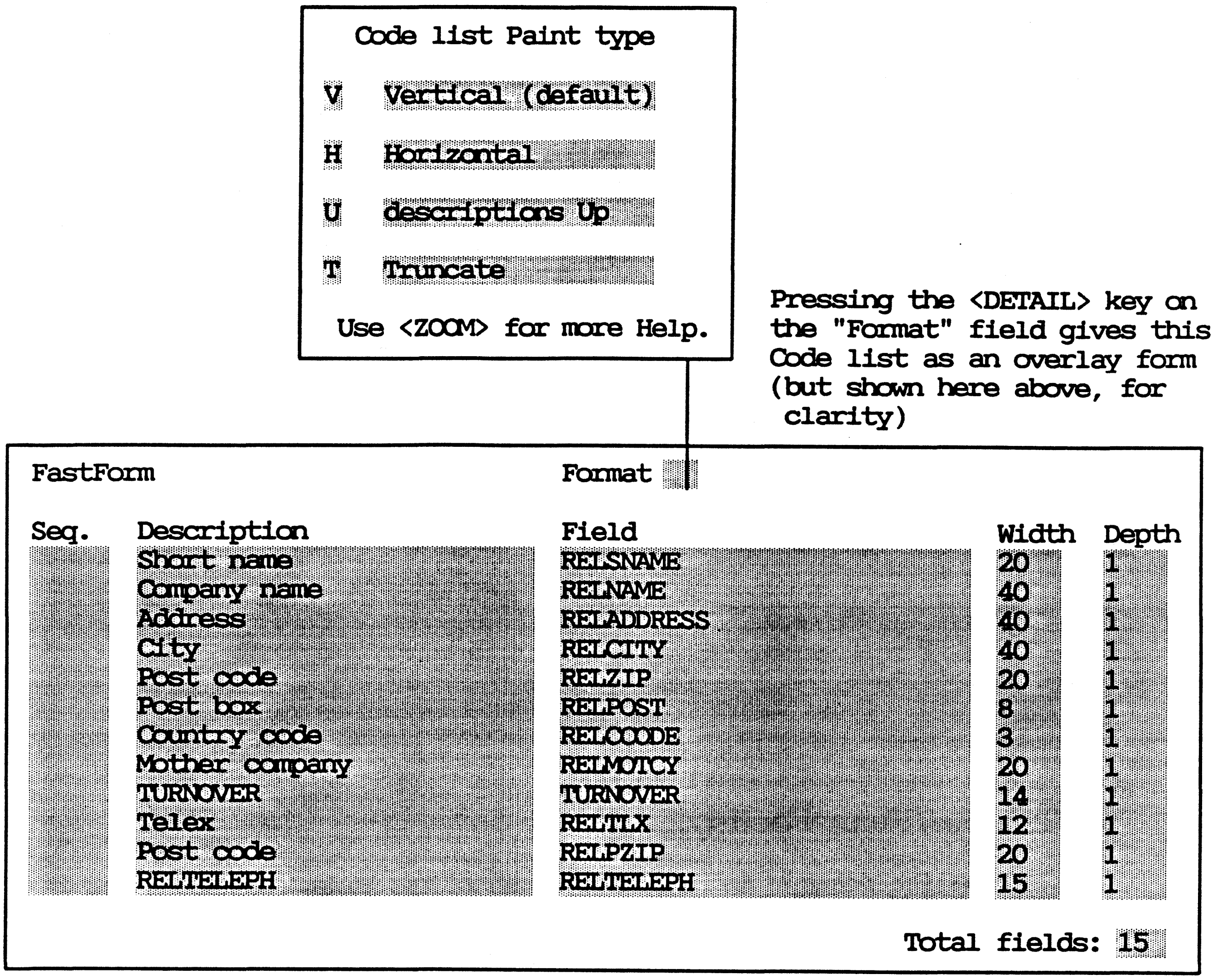
Figure 3-2. Paint Tableau with the Frame Definition Form.

The FastForm Facility...

...allows the designer to load some or all of the fields assigned to an entity in the conceptual schema quickly onto the Paint Tableau and use these fields as a basis for painting the external schema.

When the designer presses FRAME from the Frame Definition Form, the IDF retrieves information from the conceptual schema about the fields assigned to the entity being defined. This information is displayed as shown in Figure 3-3.

There is excellent integration between the various parts of the application dictionary - the conceptual schema definition for a field is accessible from the field entry.



Code list Paint type

V	Vertical (default)
H	Horizontal
U	descriptions Up
T	Truncate

Use <ZOOM> for more Help.

Pressing the <DETAIL> key on the "Format" field gives this Code list as an overlay form (but shown here above, for clarity)

FastForm Format

Seq.	Description	Field	Width	Depth
	Short name	RELSNAME	20	1
	Company name	RELNAME	40	1
	Address	RELADDRESS	40	1
	City	RELCITY	40	1
	Post code	RELZIP	20	1
	Post box	RELPOST	8	1
	Country code	RELCCODE	3	1
	Mother company	RELMOTCY	20	1
	TURNOVER	TURNOVER	14	1
	Telex	RELTLX	12	1
	Post code	RELPZIP	20	1
	RELTELEPH	RELTELEPH	15	1

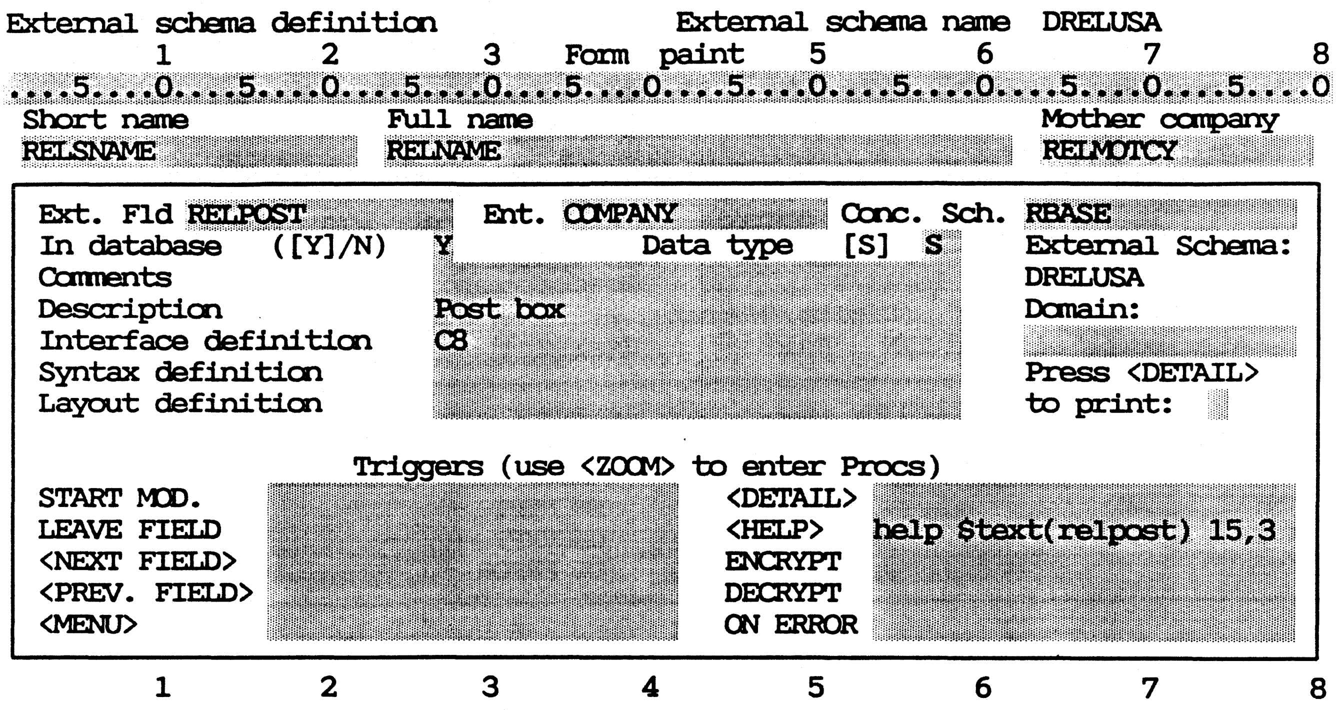
Total fields: 15

Figure 3-3. The FastForm Facility.

FastForm allows quite a bit of flexibility in defining the layout, descriptions and order of the fields so that less manipulation and editing of the Paint Tableau is necessary. A working Form can be defined in no time at all. It has to be seen to be believed.

Field and entity definitions

Designers also use the Paint Tableau to specify field and entity definitions based on the definitions provided from the conceptual schema. When s/he positions the cursor on a Field Frame on the Paint Tableau and presses DETAIL, the Field Definition Form appears in a pop-up Form as shown in Figure 3-4. The Entity Definition Form is similar to the Field Definition Form.



```
External schema definition                External schema name  DRELUSA
          1         2         3    Form paint   5         6         7         8
....5....0....5....0....5....0....5....0....5....0....5....0....5....0....5....0
 Short name              Full name                                Mother company
 RELSNAME                RELNAME                                  RELMOTCY

  Ext. Fld RELPOST           Ent. COMPANY          Conc. Sch.  RBASE
  In database   ([Y]/N)   Y          Data type    [S]  S       External Schema:
  Comments                                                     DRELUSA
  Description             Post box                             Domain:
  Interface definition    C8
  Syntax definition                                            Press <DETAIL>
  Layout definition                                            to print:

                     Triggers (use <ZOOM> to enter Procs)
  START MOD.                                  <DETAIL>
  LEAVE FIELD                                 <HELP>   help $text(relpost) 15,3
  <NEXT FIELD>                                ENCRYPT
  <PREV. FIELD>                               DECRYPT
  <MENU>                                      ON ERROR
          1         2         3         4         5         6         7         8
```

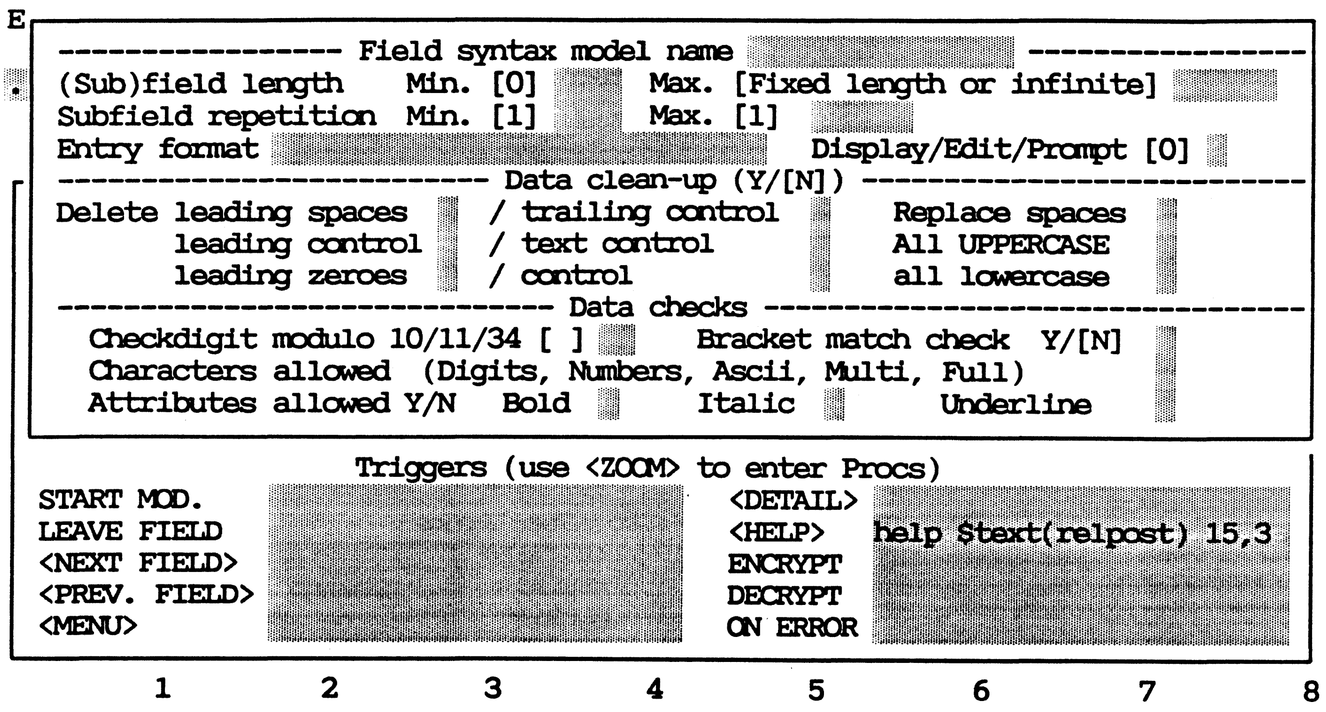
Figure 3-4. Field Definition Form.

Shorthand Codes

These definitions can be specified using a set of shorthand codes. e.g. UPC for Uppercase. The designer can simply type in the codes for the definitions directly, or, if these are not known, s/he can press DETAIL to call up a Code list form, go to the desired option with the cursor and press the ACCEPT key to insert the relevant code.

Global Definition Models

Several of these definitions can be collected under a single model name. By using model names, standard checks which are used often can be assigned a name and simply referred to as necessary. For example, a field syntax model can be defined with the name @PCODE_UK which defines pattern matching, length and data cleanup checks for British postal codes. Another model can define the checks for Dutch postal codes, @PCODE_NL for example. Model names always start with the "@" symbol.



```
E
  ---------------- Field syntax model name                ----------------
• (Sub)field length     Min. [0]        Max. [Fixed length or infinite]
  Subfield repetition   Min. [1]        Max. [1]
  Entry format                                  Display/Edit/Prompt [0]
  ------------------------ Data clean-up (Y/[N]) -------------------------
  Delete leading spaces     / trailing control       Replace spaces
         leading control    / text control           All UPPERCASE
         leading zeroes     / control                all lowercase
  ---------------------------- Data checks -------------------------------
    Checkdigit modulo 10/11/34 [ ]      Bracket match check  Y/[N]
    Characters allowed  (Digits, Numbers, Ascii, Multi, Full)
    Attributes allowed Y/N   Bold       Italic          Underline

                    Triggers (use <ZOOM> to enter Procs)
 START MOD.                             <DETAIL>
 LEAVE FIELD                            <HELP>    help $text(relpost) 15,3
 <NEXT FIELD>                           ENCRYPT
 <PREV. FIELD>                          DECRYPT
 <MENU>                                 ON ERROR
        1         2         3         4         5         6         7         8
```

Figure 3-5. Field Syntax Model Form.

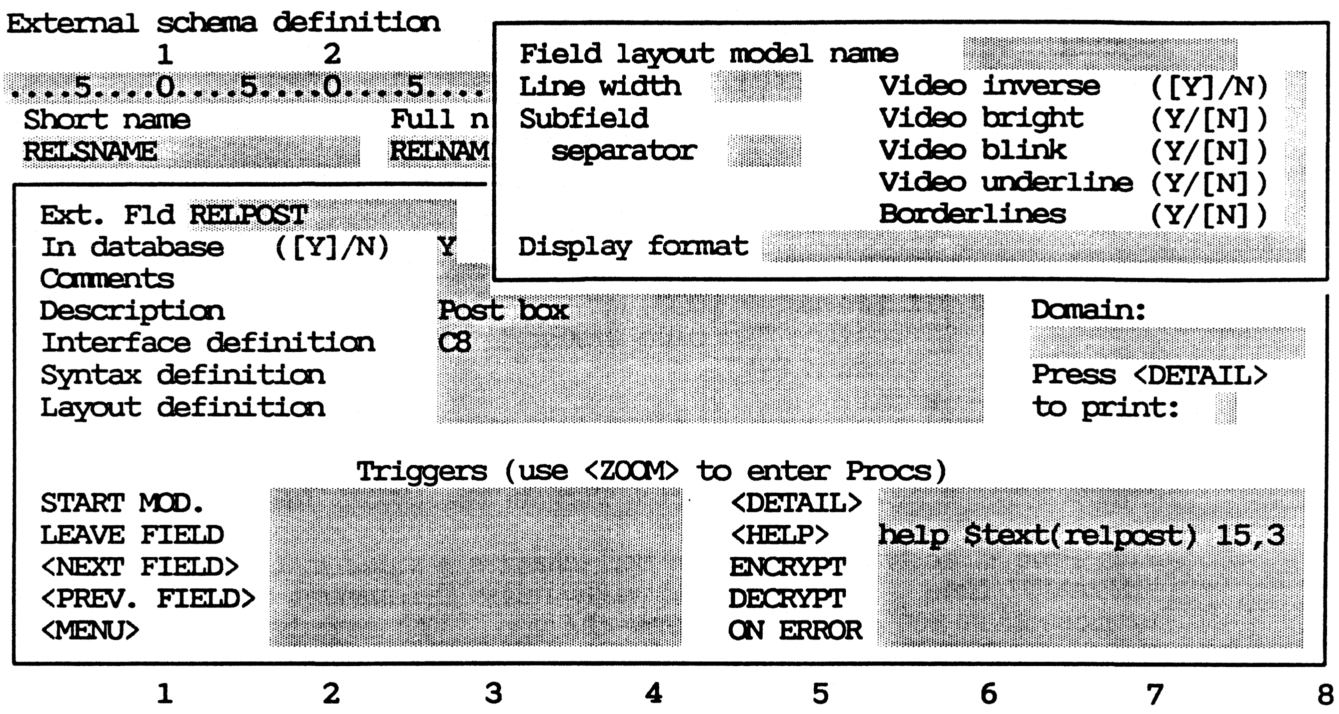
The field syntax model...

... (shown above) defines data cleanup and syntax validation checks on data entered by the user. These operations occur when the user leaves a field s/he has modified. Data cleanup converts an entered value into a desired format, i.e. convert entries to upper or lower case, remove leading zeroes, etc.

Syntax validation compares the format of an entry with a specified syntax. Syntax definitions can be used to define the maximum and minimum number of characters in a field. Other checks will scan for mismatched brackets, validate an entry with a checkdigit algorithm or check data against specified entry formats for data such as postal codes or telephone numbers.

The field layout model...

...determines the layout of the field frame, i.e. what it will look like. Line widths etc. can be defined here, as well as the date display format: you can choose whether you want your dates to be displayed as 11/15/89 or 15 November 1989, for example.



```
External schema definition
        1         2        ┌──────────────────────────────────────────────────────┐
....5....0....5....0....5..│ Field layout model name                              │
 Short name          Full n│ Line width          Video inverse    ([Y]/N)         │
 RELSNAME            RELNAM│ Subfield            Video bright     (Y/[N])         │
┌──────────────────────────│   separator         Video blink      (Y/[N])         │
│ Ext. Fld RELPOST         │                     Video underline  (Y/[N])         │
│ In database    ([Y]/N)  Y│                     Borderlines      (Y/[N])         │
│ Comments                 │ Display format                                       │
│ Description             Post box                               Domain:          │
│ Interface definition    C8                                                      │
│ Syntax definition                                              Press <DETAIL>   │
│ Layout definition                                              to print:        │
│                                                                                 │
│                  Triggers (use <ZOOM> to enter Procs)                           │
│ START MOD.                             <DETAIL>                                 │
│ LEAVE FIELD                            <HELP>    help $text(relpost) 15,3       │
│ <NEXT FIELD>                           ENCRYPT                                  │
│ <PREV. FIELD>                          DECRYPT                                  │
│ <MENU>                                 ON ERROR                                 │
└─────────────────────────────────────────────────────────────────────────────────┘
        1         2         3         4         5         6         7         8
```

Figure 3-6. Field Layout Model.

To summarize - constraint definitions are made up of parameters, definition models and Procs. Each model and Proc controls a specific aspect of the processing for that field or entity. However, Uniface offers a great many options which are, of course, optional. If you don't need checkdigit modulo, variable length techniques, etc. simply leave the entries blank. Also, because the IDF compiler assumes a default value for every entry, it is not necessary to completely define every field and entity. The choice is up to you.

Field and Entity-level triggers

The lower portion of the Field and Entity Definition Forms contains Proc triggers. These triggers determine what happens when a particular event occurs in the context of the specific field or entity. For example, what processing should occur when the user begins to modify the data in a field, or what (if anything) should happen when s/he presses HELP.
There are three kinds of Proc triggers at this level:

Semantic validation & calculation...

...concerns the value of an entry, not only its format. Types of semantic validation include existence checks, range checks, code list checks, referential integrity checks and cross-field validation.

Figure 3-7 shows a zoomed Proc which checks that the value in, say, an invoice amount field ("INVAMT") does not exceed $5,000 i.e. the Proc performs a range check.

```
LEAVE FIELD   if (invamt > 5000)
                message "Invoice amount is too high!"
                return (-1)
              endif
```

Figure 3-7. A Semantic Validation Proc in the LEAVE FIELD Trigger

This Proc is activated after the user changes a value in the "INVAMT" field. If the changed value raises the amount outstanding to more than $5000, Uniface warns the user. The negative value in the "return" statement prevents the user from leaving the field until s/he corrects the value.

Existence checks query the database directly. Normally I/O with the database is transparent to the user and even to the designer. Uniface controls I/O automatically. Figure 3-8 illustrates a Proc which ensures that the user does not specify a document number (docno) which already exists.

```
LEAVE FIELD   lookup
              if ($status > 0)
                 message "Document number already exists!"
                 return (-1)
              endif
```

Figure 3-8. An Existence Check Proc Using "lookup".

When the user leaves the "Document #" field after making a modification, Uniface builds a "select" statement based on the current key values for the DBMS which manages the current entity. If the DBMS reports that one or more rows were selected via the $status register, the Proc displays a warning message and returns the cursor to the beginning of the Frame. This class of Procs is also used for calculating derived fields, e.g. subtotals fields.

Field prompting sequence...

...can override the default field prompting order of left to right, top to bottom for the NEXT/PREVIOUS FIELD function. Prompting sequence can also be defined conditionally.

For example, a company can decide that when the average of a customer's invoices is greater than $10,000, s/he should receive a reminder letter. When the user presses NEXT FIELD key, the Proc in Figure 3-9 uses the "sql" instruction to instruct the Oracle DBMS to compute the average of the customer's invoices. If the average is greater than the allowed limit, $prompt positions the cursor on the "Corrtext" field in the CORRESP Entity Frame so that the user can send a reminder.

<NEXT FIELD>

```
sql "select avg(invamt) from invoices where custnum = '%%custnum'","ora"
if ($result > 10000)
     message "Invoice average too high.  Send reminder."
     $prompt = corrtext.corresp
endif
```

Figure 3-9. A Conditional Field Prompting Proc Using SQL.

Session control...

...at Frame-level controls forward movement through an application. This definition determines which external schema is started when the user presses DETAIL. Session control for backward movement, i.e. to the previous external schema, is controlled at Form-level. In our example, the user sees a code list for the "Document Code" Field Frame when s/he presses DETAIL with the cursor in that field. Figure 3-10 shows the Proc which calls that code list.

<DETAIL>

```
run "cldoc"
```

Figure 3-10. Code List Proc.

Input/Output (I/O) Procs...

...are described in Chapter five, "The Database Interface".

The Proc language

A Proc (PROCessing specification) is a series of 4GL statements. They are instructions about what to do if a certain event occurs in a certain data context, i.e. when a Frame is current. Procs include high-level statements, which initiate complex processing with a single statement e.g. "run", "edit", "store", "retrieve" and familiar statements such as "if", "else", "return", etc.

The structure of the Proc language allows powerful instructions to be defined with simple pseudo-English statements. A chart showing Proc instructions and facilities is shown in Figure 3-11.

Procs which are commonly used can be placed in a central Procs library. The Procs in this library can be called by name from any trigger.

Conditional	if, else, endif
Assignment and calculation	(assignment), compute
Message	message, askmess, putmess, clrmess, help
Transaction control	commit, rollback, store, retrieve, erase
I/O control	read, write, delete, lock, reload, open, sql, print, release, lookup
Branching and linking	goto, label: perform, spawn, run, call, entry,
Exit	return, done, exit, apexit, end
Occurrence control	creocc, setocc, remocc
Structure Editor	edit, display, $prompt
ASCII file manipulation	file_dump, file_load
Debug	debug, nodebug
Miscellaneous	clear, blockdata, refresh, numset, numgen, set, reset, $keyboard, $variation, $prompt

Figure 3-11. Examples of Some 4GL Proc Instructions and Facilities.

The link with 3GL

Although virtually all processing can be defined using Proc statements, there are some procedures in which a 3GL routine is more appropriate; complex statistical calculations for example.

The "perform" Proc instruction calls an external 3GL routine for use within a Proc. A routine from most 3GLs can be used in this way, e.g. C, COBOL, FORTRAN, etc. External routines can be used without pre-compilation. Uniface automatically generates all software links when the application is compiled. Existing 3GLs thus serve as extensions of the Proc 4GL.

SQL, in all its different dialects, can also be seen as an extension to the Proc language, via the "sql" Proc instruction (if, of course, the DBMS used supports SQL). The ramifications of the Proc language are many and varied. It remains, however, an intrinsically elegant construction.

Printing and reports

All Forms in Uniface are defined in exactly the same way, no matter what their function. External schemas for data input, retrieval, reports, etc. are all created using the Paint Tableau. Therefore, Uniface does not a have a separate "report writer" facility. It's simply not necessary, because...

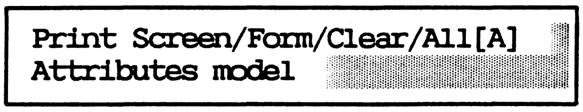
...there is almost no difference for Uniface between sending data to the terminal screen and to paper. By using the Paint Tableau to define Frames-inside-Frames, attractive, structured reports can be created easily and quickly. There are, of course, a few different formatting considerations, which Uniface supports.

For example, paper cannot "scroll" to show hidden occurrences. Therefore, Entity Frames are automatically expanded to show all occurrences as the Frame is printed. Uniface also allows the designer to define Header and Trailer Frames. Any text or field frames located within these special frames will be printed at the top and bottom of each page, respectively.

Header and Trailer Frames are very similar to Entity Frames. They are defined in the same way and are identified with the same marker on the Paint Tableau. However, Header and Trailer Frames have no connection with the database and no relationships with any of the entities painted on the form. Dummy fields can be painted to show computed subtotals and other calculations.

The PRINT key is available from every Uniface external schema (unless disabled by the designer). With this function, Uniface retrieves data into the external structure, formats the retrieved data and sends it to a printer. The designer can also use the "print" Proc instruction to make the process more automatic, or to execute it in batch mode.

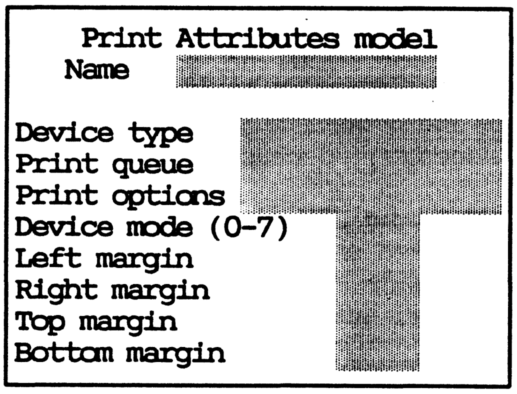
The user has four options. S/He can choose to print the data shown on the Form, on the Screen or All data in the external structure. In the latter case, Uniface expands Entity Frames to show all hidden occurrences and expands Field Frames to show hidden characters. To increase performance this data can be Cleared from memory immediately after it has been printed.



```
Print Screen/Form/Clear/All[A]
Attributes model
```

Figure 3-12. The Print Form.

The PRINT ATTRIBUTES function allows the user to define print attributes. The user can define many different models and select one at the time of printing.



```
        Print Attributes model
      Name

Device type
Print queue
Print options
Device mode (0-7)
Left margin
Right margin
Top margin
Bottom margin
```

Figure 3-13. Print Attributes Model Form.

Uniface can support any device

Uniface makes no distinction between display device types and printer device types. Uniface can support *any* device. Several standard printer and display types are supported directly. Designers can, however, use the Device Definition Facility to create their own definitions, i.e. define a table which converts character attributes codes to the appropriate code for an individual device type.

All interaction with output devices is table driven. For example, when a word has to appear on the screen in **boldface**, Uniface looks up the codes for this feature in the device table for the display terminal being used. Another display or printer has its own set of codes, provided in a different device definition.

If a desired device is not currently supported, the designer can use the Device Definition form to create one for it. The existing device tables can be used as a base for this, if the device has only slightly different specifications to one already supported.

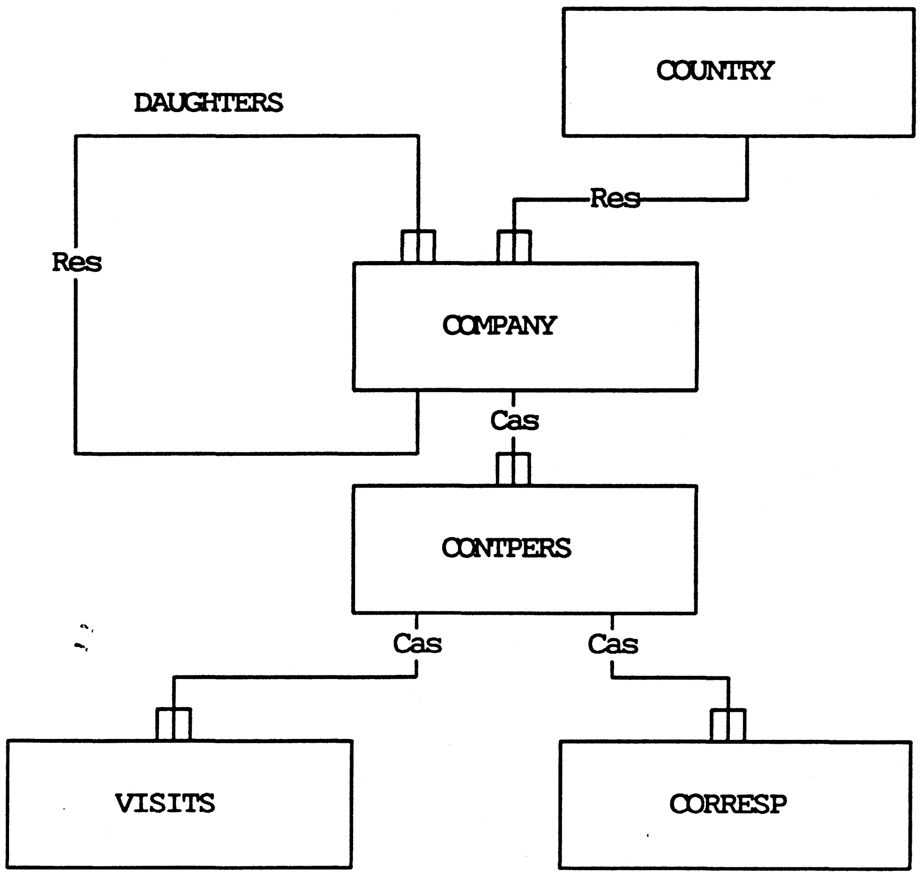
This facility also allows users to adapt a standard device definition for particular needs or preferences, eg. national character sets, color preferences, etc. If a particular device does not support a certain feature (such as graphic borderlines or color) you can define a substitute approximation for the device or leave the feature out.

Device independence is a unique feature of Uniface - one which frees you to concentrate on the real job in hand, the design of your applications.

Chapter 4. Defining the Conceptual Schema

The application development process usually begins by the system administrator defining the conceptual schema. This is the central description of all of the rules and constraints for each entity and each field in the database, and the relationships between entities. Thus it is a complete, relatively abstract description of the entire database. The conceptual schema provides a powerful way to both control and simplify the application development process.

To begin with, the system administrator uses his/her workbench in the IDF to describe the structure of the database. Figure 4-1 shows a conceptual data structure diagram of the example used in this brochure.



Delete constraints: Nul (= nullify), Cas (= cascade), Res (= restricted)

Figure 4-1. Conceptual data structure diagram.

Frames-Inside-Frames

Designers create external schemas based on the definitions in the conceptual schema. The Frames-Inside-Frames convention defines a path through the conceptual schema. Figure 4-2 shows an external schema which uses most of the entities defined in the conceptual schema shown in Fig. 4-1.

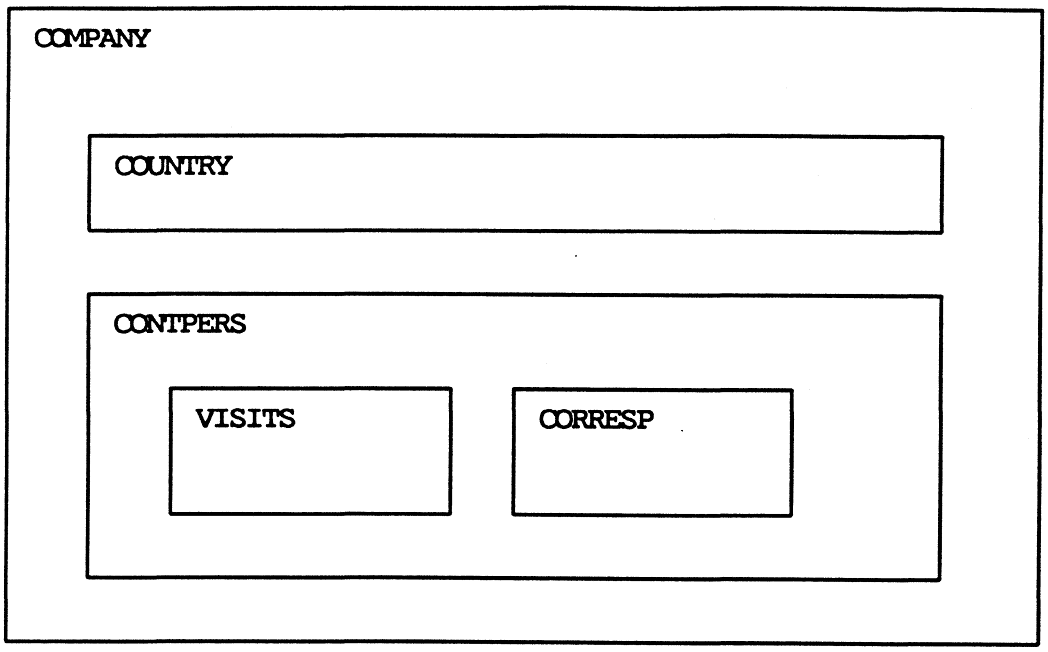


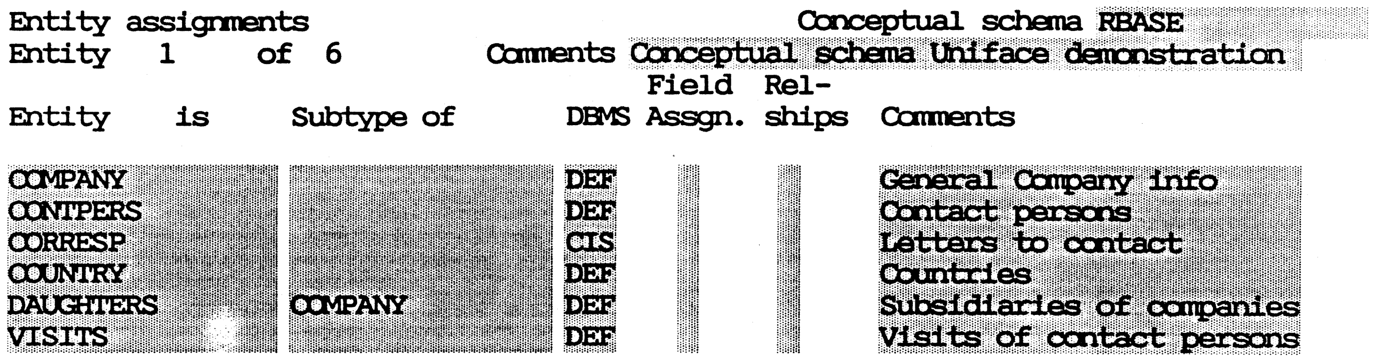
Figure 4-2. External schemas diagram.

Defining entities

The system administrator usually begins by assigning various entities to a conceptual schema. An **entity** is a distinguishable object represented in the database. In Uniface, an entity usually corresponds to a file or table in the internal schema, unless it is a subtype entity. Subtype entities are essentially aliases which can be used for entities in a specific role relationship, etc.

Figure 4-3 shows the Entity Assignment Form. This Form shows an overview of all of the entities in the conceptual schema. Notice that the DAUGHTERS entity is defined as a subtype of the COMPANY entity.

The DBMS entry is an important one for Uniface. The system administrator uses this entry to tell Uniface which database management system should administer each entity. This is all that Uniface needs to know about which DBMS should be used. Uniface knows from this entry whether SQL must be generated for I/O, or "reads" and "writes" via an index, to name but two examples. Beyond this point, Uniface is completely database independent. A mixture of any of the supported systems can easily be defined within a conceptual schema. The designer hardly needs to be concerned about which DBMS administers each entity.



```
Entity assignments                                   Conceptual schema RBASE
Entity   1    of  6          Comments Conceptual schema Uniface demonstration
```

Entity is	Subtype of	DBMS	Field Assgn.	Rel-ships	Comments
COMPANY		DEF			General Company info
CONTPERS		DEF			Contact persons
CORRESP		CIS			Letters to contact
COUNTRY		DEF			Countries
DAUGHTERS	COMPANY	DEF			Subsidiaries of companies
VISITS		DEF			Visits of contact persons

Figure 4-3. Entity assignment form.

Another big plus point: the DBMS assignment can be altered at runtime *without* having to change definitions at all and *without* having to recompile. And benchmarking is effortless. For example, a designer can switch an application which normally stores data in Oracle to RMS in order to test differences in performance, simply by changing assignments:

```
        $DEF=$ORT
```

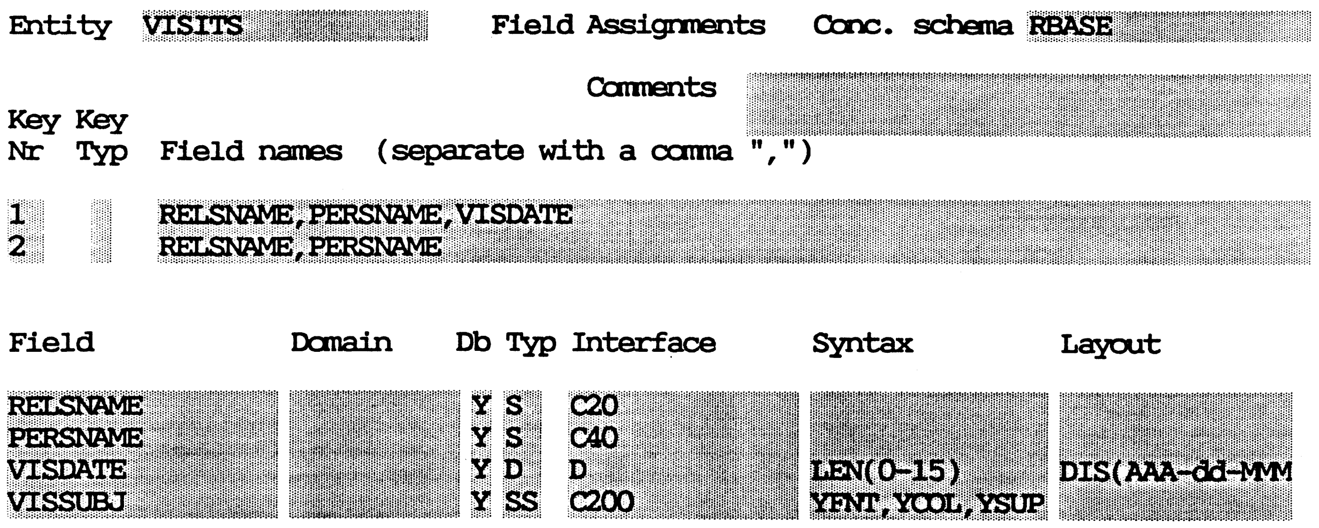
This serves to illustrate even more emphatically the total independence of Uniface applications.

Defining fields

Each entity defined in the above Form has **fields** assigned to it. To see the fields assigned to an entity, the system administrator positions the cursor on the <u>Field Assgn.</u> option opposite the desired entity on the Entity Assignments Form and presses DETAIL. The Field Assignments Form for the VISITS entity is shown in Figure 4-4.

This Form shows the keys for an entity, the fields assigned to the entity, and definitions for each field. The keys for this entity are

defined at the top of this Form. Candidate keys, consisting of one or more fields, uniquely identify each occurrence of the entity. One of these is chosen as the primary key (Key Nr 1).
Index keys can be included for performance reasons or to allow more flexible retrieve profiles.



Entity VISITS   Field Assignments   Conc. schema RBASE

Comments

Key Nr	Key Typ	Field names (separate with a comma ",")
1		RELSNAME,PERSNAME,VISDATE
2		RELSNAME,PERSNAME

Field	Domain	Db	Typ	Interface	Syntax	Layout
RELSNAME		Y	S	C20		
PERSNAME		Y	S	C40		
VISDATE		Y	D	D	LEN(0-15)	DIS(AAA-dd-MMM
VISSUBJ		Y	SS	C200	YFNT,YCOL,YSUP	

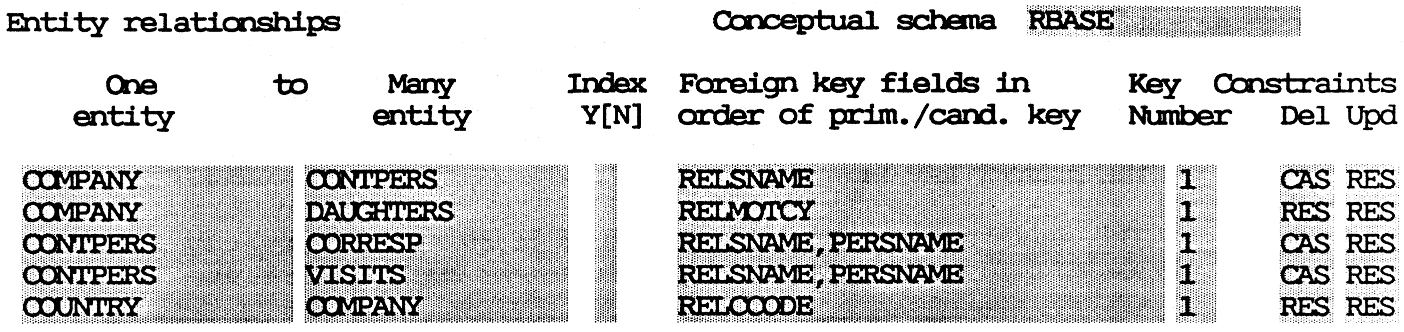
Figure 4-4. Field assignment form.

When the system administrator defines a field as part of the primary key, Uniface always prevents the user from modifying that field in an occurrence which has been retrieved from the database. This constraint is implicit in the definition of a primary key.

The fields in the entity are listed at the bottom of the Form. Next to the name of each field are the most important definitions about each field. To see the complete field definition, position the cursor on the name of one of the fields and press DETAIL. The Conceptual Field Definition Form appears. This Form is almost exactly like the External Field Definition Form explained in Chapter 3. The difference here is that when accessed from the conceptual schema, the system administrator creates central definitions which are used whenever the field is used, on any Form.

Defining relationships...

...between entities is the last important step in the definition of the conceptual schema. Figure 4-5 shows the Relationship Definition Form.



Entity relationships   Conceptual schema RBASE

One entity	to	Many entity	Index Y[N]	Foreign key fields in order of prim./cand. key	Key Number	Constraints Del	Upd
COMPANY		CONTPERS		RELSNAME	1	CAS	RES
COMPANY		DAUGHTERS		RELMOTCY	1	RES	RES
CONTPERS		CORRESP		RELSNAME,PERSNAME	1	CAS	RES
CONTPERS		VISITS		RELSNAME,PERSNAME	1	CAS	RES
COUNTRY		COMPANY		RELCOODE	1	RES	RES

Figure 4-5. Relationship definition form.

Each one-to-many relationship is defined by migrating the primary key of one entity to another entity, where it associates the two entities as a foreign key. Referential integrity constraints can also be defined with each relationship. These constraints define what should happen to occurrences of the "many" entity when an occurrence of the "one" entity is deleted.

For example, if a company is deleted, should the deletion "cascade" (CAS) to delete the contact persons working for that company? Should it allow them to continue to exist in no-man's land, without a company (the nullify (NUL) constraint)? Or should the delete operation be "restricted" (RES), i.e. a company may not be deleted if contact person data for that company still exists?

If the DBMS checks this sort of thing, Uniface doesn't, but can generate the data definition language (DDL) to define the constraint. Uniface ensures automatically that these constraints are enforced, even when the entities are stored in different DBMSs. This mechanism conforms to the latest ANSI/ISO proposals for SQL 2.

In addition, subtype entities can be used to define relationships using different roles. This allows the system administrator to define one-to-many relationships within an entity, multiple relationships between entities, many-to-many relationships within an entity, etc. You recall that in the example above, DAUGHTERS is a subtype of the COMPANY entity. The relationship between these two entities defines a one-to-many relationship within an entity; one company can have many subsidiaries, each of which is, in turn, also a company.

Loading conceptual schema definitions from another source

In the current version, Uniface can load definitions from the Oracle, Sybase, Ingres and Rdb relational DBMSs, and, in principle, any CASE tool.

...Database management systems

When using the load facility for a DBMS, the IDF retrieves the internal schema definitions of tables stored there. It can read in the name, length, and data type of each column in a table. The IDF also looks for a UNIQUE INDEX to see which columns probably make up the primary key for a table.

This facility means that the definitions of all existing tables can be loaded into Uniface in one step. From that starting point, the system administrator can complete the conceptual schema by defining relationships, the additional constraints and rules supported in the Uniface application dictionary, etc..

...and Computer Aided Software Engineering tools

There is growing tendency of systems analysts to reach for CASE when confronted with the ever-increasing demands of a DP department. It makes their job so much easier. CASE tools are, however, limited to the analysis phase. The development cycle is halted abruptly by a mountainous stack of paper which then has to be made into programs. Unless, of course, you have Uniface.

When using CASE tools with Uniface, the entities, fields, keys and relationships are loaded effortlessly into the conceptual schema, together with information about which fields and which keys belong to which entity, which entities are related to each other, etc.[1]

The CASE Tool Interface utility imports and exports definitions into the conceptual schema, to a documented standard interface format. Conversion programs then convert definitions from this format to one accepted or generated by various CASE tools. Bridges are currently available or planned to Software Through Pictures (available on Sun workstations) Excelerator, SDW and IEW.

With Uniface, you're free to use *any* CASE tool-DBMS combination you want. You can even load definitions from one CASE tool into another, via Uniface.

[1] See the section headed "The application dictionary and the CASE interface" in Chapter 6.

Chapter 5. The Database Interface

Uniface is not a DBMS tool which later added links to other DBMSs. Right from its conception, Uniface was fully DBMS-independent. This remarkable feature is not something that just happened "along the way" but is an integral part of the design - the Uniface driver.

The driver allows Uniface to interface with virtually any DBMS, taking full advantage of the facilities offered by each of them, regardless of their widely-differing internal architectures.

A unique freedom

Drivers have been developed to allow Uniface to interface with relational and hierarchical DBMSs, Indexed Sequential Access Method (ISAM) systems, and to manipulate simple sequential files of data. Drivers for other DBMSs can be developed on request, or enterprising programmers can make their own, using the *Uniface DBMS Driver Cookbook*. DBMS's currently supported or in the pipeline are...

ORACLE

BRITTON-LEE

INGRES

FOCUS

SYBASE

dBase III/IV

Rdb

SQL Server

RMS

C-ISAM

TRIP/TDBS

SQL/RT

INFORMIX

BASIS

sequential files

Actual I/O is a fully-automatic and invisible operation for the user, and even the application designer rarely needs to be concerned with the details of how it is accomplished. The opportunity to influence how I/O is performed is, nevertheless, always available.

As well as generating data manipulation language (DML) commands, Uniface offers the capability to generate data definition language (DDL) commands which create tables and files at run time. If Uniface finds that an entity does not exist in the database, it can use the definitions in the conceptual and internal schemas to create the tables or files.

What is a Uniface DBMS driver ?

A driver is a layer of software which acts as a mediator between the Uniface run time system and a specific DBMS. The Information engineering & Design Facility (IDF) generates applications following definitions provided by designers. The DBMS driver is an integral part of the resulting executable program. And new drivers can be linked *without* modifying the application.

Whenever an I/O operation is necessary, the run time system calls the driver routine for the appropriate DBMS. The driver routine translates the generic function call from Uniface (e.g. "lock", "open", "update", etc.) into the instruction or series of instructions necessary to have the DBMS perform the required operation.

The Uniface environment makes a great deal of information available to the driver routine via a set of standardized data structures. The driver uses the information in these structures to build the I/O statements necessary to interface with the DBMS. In turn, the driver is responsible for making information needed by Uniface available in these structures. Thus the driver provides information about the capabilities of the DBMS, formats and limitations which apply to the DBMS, etc.

How does a driver work ?

Whenever I/O to a DBMS is required, Uniface issues an I/O function call to that DBMS's driver. This call addresses the "front end" routine for the driver with a request code filled into an available data structure. The driver examines the request code, and branches to a routine which handles the specific request.

There is a Uniface function in the driver interface for various basic database operations (eg. OPEN, CLOSE, UPDATE, SELECT, FETCH, COMMIT, etc.), a function for initiating and terminating a DBMS session, and a function for identifying the characteristics of a DBMS to Uniface.

Retrieving and editing data

The figure below shows how Uniface interfaces with two different databases at the same time. In this simplified illustration of our example application, a relational database manages contact person and visits information, and an ISAM file holds documents.

The external structure is the Uniface Form as it appears to the user. The data structure is depicted and defined using Frames-inside-Frames as explained earlier.

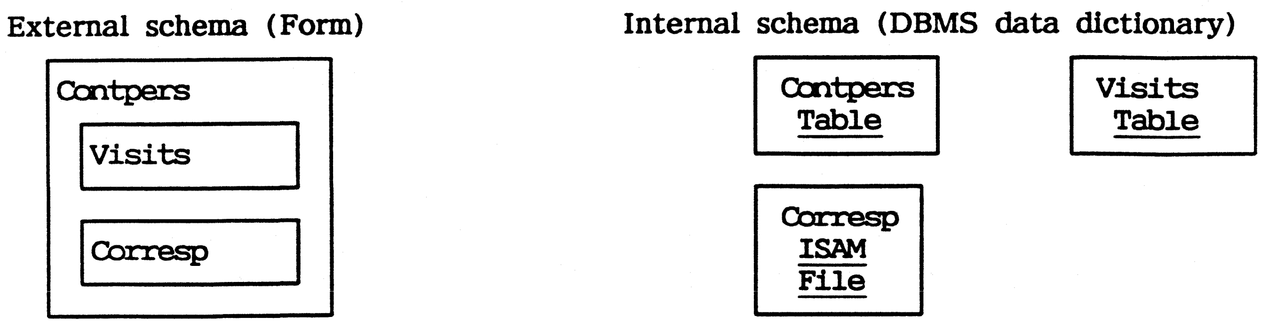
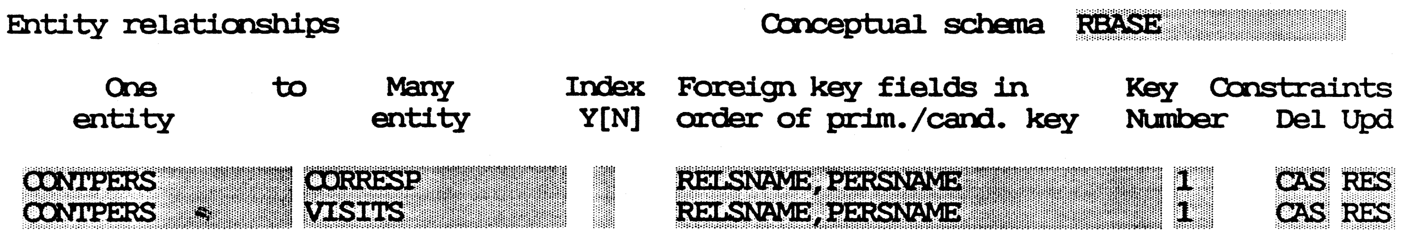


Figure 5-1. External schema and corresponding internal schema.

The relationships between the entities are defined in the conceptual schema, using foreign keys. The VISITS entity is associated with the CONTPERS entity via the "Relsname" (Short name) and "Persname" fields, which appear in both tables.

Because Uniface contains a conceptual schema of the data structure, Uniface can check referential integrity constraints during editing to maintain the integrity of the database, even across different DBMSs. If the user deletes a company record, the designer can, for example, define whether or not the company's contact persons should also be deleted.



Entity relationships   Conceptual schema RBASE

One entity	to	Many entity	Index Y[N]	Foreign key fields in order of prim./cand. key	Key Number	Constraints Del	Constraints Upd
CONTPERS		CORRESP		RELSNAME,PERSNAME	1	CAS	RES
CONTPERS		VISITS		RELSNAME,PERSNAME	1	CAS	RES

Figure 5-2. Relationship definition via foreign keys.

To retrieve the data for a specific contact person, the user types in a name (the primary key field) and presses RETRIEVE.

The user can also perform a "Query-By-Form". To retrieve all of the contact persons whose name begins with a "J", s/he types "J*" in the NAME field and presses RETRIEVE. Driver routines issue the necessary Structured Query Language (SQL) commands to retrieve the contact persons' data and visits data from two tables in a relational database. Another driver issues ISAM I/O statements to retrieve the contact persons' correspondence from the ISAM file.

The occurrences retrieved from the database remain in working storage during editing. The user edits the data in the external structure with the Structure Editor.

Storing data

In a typical transaction, the user retrieves data, e.g. all the visits and correspondence associated with Dr. F.F.J. Jansen. S/He adds some new letters, deletes a few old ones etc.. The user presses STORE to send the updated occurrences to the database.

Because editing takes place in working storage, the database remains unchanged until the "store" Proc instruction is issued, usually by pressing the STORE key. The user can press QUIT at any point to abandon all editing changes.

Storing the updated occurrences is a two-stage process. In the first step, Uniface analyzes the external structure to see what alterations have been made. Uniface determines which occurrences have been added, deleted or modified.

In the second step, Uniface creates an internal event to trigger the WRITE Proc for each modified or inserted occurrence and the DELETE Proc for each removed occurrence. These Procs contain statements which call the appropriate driver routines to update the database. These driver routines generate and execute the required update statement in the DML of each DBMS, e.g. SQL.

How the designer influences I/O

The I/O mechanism explained above is automatic and entirely based on structure definitions. Uniface could have been designed to make I/O a completely automatic and invisible operation. However, the decision was taken to create "store", "retrieve" and "erase" Proc instructions and STORE, RETRIEVE and ERASE triggers to provide maximum flexibility to the designer.

Level 1 (Form) Event	Proc inst.	Level 2 (DBMS) Internal Event	Proc Inst.
<RETRIEVE>	retrieve	READ	read
<STORE>	store	WRITE	write
<ERASE>	erase	DELETE	delete
(no specific trigger)	commit	LOCK	lock
	rollback		

Figure 5-3. I/O Proc instructions and events.

Figure 5-3 illustrates triggers and Proc instructions on both I/O levels. These triggers provide the flexibility to influence I/O in three ways. The designer can:

* Include pre- or post- I/O Proc instructions, which perform operations like clearing the Form and sending a message to the user after a "store" operation.

* Omit an I/O instruction to prevent I/O from occurring, e.g. omit the Form-level "erase" instruction to prevent the user from erasing records from the database.

* Put I/O instructions in different triggers, e.g. include a "store" instruction with an ACCEPT trigger rather than with the STORE trigger.

Figure 5-4 shows a Proc which includes post-I/O instructions which check whether the "store" operation succeeded, and if so clears the Form and sends a message to the user after the "store" operation.

<STORE>

```
store
if ($status < 0)
  message "Store error!"
  rollback
else
  message "Data stored.  Ready for next entry."
  commit
  clear
endif
```

Figure 5-4. A STORE Proc.

Data-staging

Uniface can retrieve data in several stages when there is a great deal of data for the external structure. Uniface initially retrieves only enough data to fill the visible part of the Form. When the user requests more data, e.g. presses NEXT OCCURRENCE to see hidden occurrences, Uniface automatically fetches the next stage of data from the database.

The operation is completely invisible to the user; s/he sees data when s/he needs it. No additional definitions from the designer are necessary. Data-staging is a fully automatic feature of the Structure Editor.

Uniface makes full use of the optimizing strategies of the individual DBMS. Cursors, stored procedures, indices are all utilised to the full. Another example is the segmenting of long data strings. Any field can be of "infinite" size. Uniface automatically retrieves very long fields in segments, as triggered by the Structure Editor. Again, this feature is completely automatic.

Transaction processing

A transaction in Uniface is the processing which occurs between two "commit" Proc instructions, for example, or between the start of the application and the first "commit" instruction. Uniface allows the designer to explicitly define what constitutes a transaction. A transaction can easily be performed using several different Forms, depending on where the "commit" or "rollback" instructions are placed.

The precise definition of a transaction allows Uniface to use fairly straightforward concurrency control mechanisms. Different users cannot be permitted to modify the same data at the same time. Therefore, most DBMSs offer a mechanism for locking an occurrence when a user wishes to modify it. Uniface uses generic Proc instructions which use these facilities. The I/O driver routines translate the generic Uniface instructions to the specific DML commands for each DBMS.

Uniface supports two kinds of locking

Uniface can adapt locking strategy to suit various environments, e.g. to support various types of high performance on-line transaction processing (OLTP) database management systems.

Cautious locking...

...attempts to lock an occurrence when the user first modifies the data in that occurrence. If the occurrence has been locked by another user, Uniface sends a message to the user telling him/her so. S/He can then continue editing and return to this occurrence later.

This type of locking is used in a normal environment where maximum protection is needed and where transaction rates are low.

If the occurrence has been deleted by another user, Uniface sends a message to the user. If the occurrence has not been deleted and is not locked, Uniface compares the occurrence in the database with the occurrence in the external structure to ensure that the data on the Form is still current. If the occurrences no longer match, i.e. another user has modified the occurrence, Uniface can send a message to the user, read the new occurrence into the external structure and lock it. If the data is still current, Uniface locks the occurrence and processing proceeds. The occurrences remain locked until a "commit" or "rollback" instruction is executed.

Optimistic locking...

...postpones locking the occurrence until the last possible moment, which can result in considerable performance improvements in multi-user environments.

Optimistic locking is desirable in processing environments in where the probability of two users simeoultaneously modifying the same occurrence is very low.

With optimistic locking, the "lock" instruction is ignored. Instead, Uniface locks an occurrence automatically when the "write" or "delete" instructions are activated. Uniface also checks at that time whether the occurrence is still current; i.e. whether another user has modified or deleted the occurrence. If the occurrence is no longer current, then the user receives an error message and must start the transaction again.

If a problem occurs during an I/O operation Uniface can automatically instruct the DBMS to roll the transaction back to the point at which the last "commit" instruction was executed (if the DBMS concerned supports "rollback"). The database remains as if all the processing from that point forward had never been attempted.

Chapter 6. The Information Engineering & Design Facility (IDF)

Uniface is itself a Uniface application. The IDF, a sub-set of the total Uniface system, is a sophisticated information management system of design definition. It was also the first application built using Uniface. The origins of Uniface are shrouded in the mists of metaphysics - not only is one unsure as to whether the chicken or the egg came first, one also has some difficulty in determining which is the chicken and which the egg. Software developers move in mysterious ways...

The database managed by the IDF is called the *application dictionary* (see the end of this chapter for how it connects with CASE). It is a normal database and can be managed by any of the DBMSs supported by Uniface.

The workbench approach

The IDF provides a system administrator's workbench to define and manage conceptual schemas and a designer's workbench to define and manage external schemas. The main working space of these workbenches is a tableau used to define structures. IDF Detail Forms appear as overlays over the structure to add and modify the definitions in the application dictionary.

Both workbenches have access to the definitions in the application dictionary. Constraints defined as declarative definitions and processing specifications (as described earlier) can be defined centrally in the conceptual schema, and then enhanced or narrowed locally in an external schema. This mechanism greatly reduces duplicated effort and improves standardization of data definitions, e.g. display format, messages, etc.

When the system administrator describes the structure of the database on his/her workbench, s/he also specifies a definition for each field and entity. These conceptual definitions are used in effect as defaults when the designer includes the field or entity in an external schema. The designer is free to modify these definitions as needed.

For example, the system administrator can specify that a date field should usually be displayed as **01-JUN-1987**. A designer can decide for a particular external schema that s/he would prefer to have the name of the month spelled out, or include the day name, etc. S/He can create an external variation of the conceptual definition which will be used for that external schema only. The conceptual definition remains unchanged for use in other external schemas.

Of course, some definitions cannot be modified locally. These definitions are under the exclusive control of the system administrator. For example the Interface definition. It is impossible for a field to have a fixed length of 10 in one external schema, and a length of 20 in another. Therefore, this definition is conceptually controlled.

IDF functions

The first three options on the left side of the IDF Main Menu, shown below, are used to define the structures described earlier.

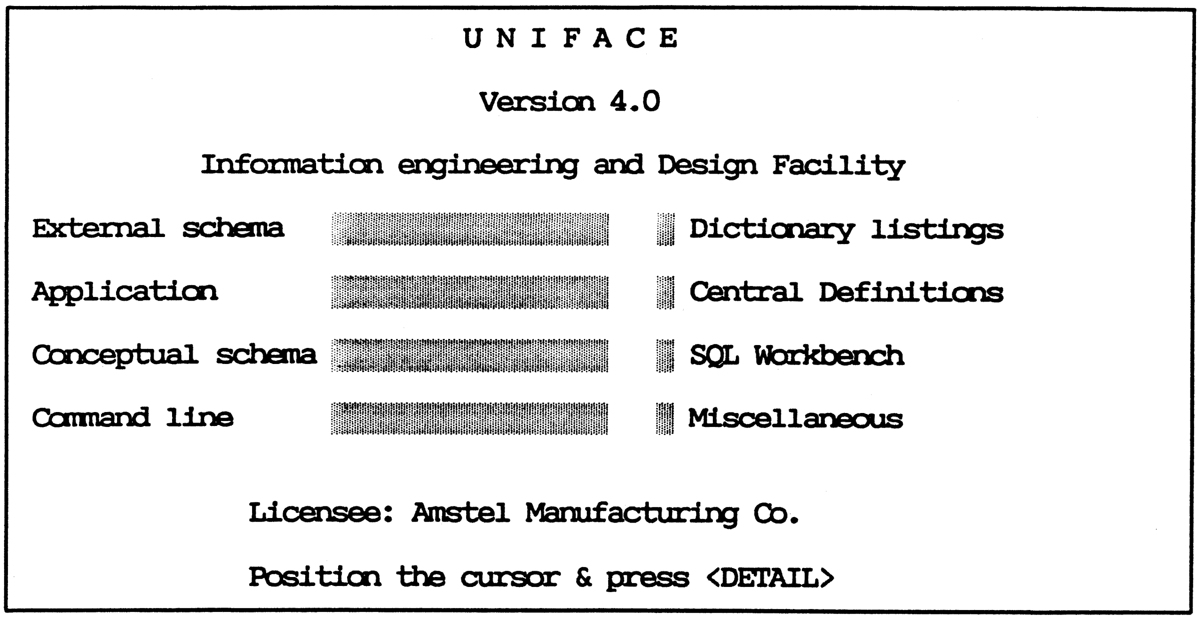

```
                         U N I F A C E

                          Version 4.0

           Information engineering and Design Facility

External schema                    Dictionary listings

Application                        Central Definitions

Conceptual schema                  SQL Workbench

Command line                       Miscellaneous


             Licensee: Amstel Manufacturing Co.

             Position the cursor & press <DETAIL>
```

Figure 6-1. The IDF main menu.

The External schema option calls a menu used to define external schemas and compile the definitions into the high-performance tables used at run time. Other functions on this menu allow you to test an external schema, print a definition for documentation purposes, see an overview of all of the Procs used in an external schema, and clean out definitions from the application dictionary that are no longer needed.

The Application option is used to define application-level definitions such as 3GL and DBMS driver routines to be included, the position of the message area, etc. The Application Definition Menu is also used to link the routines used in the final application.

The Conceptual schema accesses a menu with which the system administrator defines the conceptual schema. This menu accesses

Forms to define entities, fields, domains, relationships, global definitions models, etc.

The Dictionary Listings menu prints cross-reference report listings of the information held in the application dictionary. These listings document and give an overview of the definitions specified with the IDF.

The Central Definitions option is used to define Proc libraries where commonly-used Procs can be centrally maintained, a central text system for messages and help text in several languages, and translation tables used to define alternative keyboard layouts.

The SQL Workbench option provides the developer with the facility to issue Structured Query Language (SQL) commands directly from the terminal for DBMSs which support SQL.

Options on the Miscellaneous menu include a text file editor, the language setup and code list and license maintenance facilities.

Like every Uniface application, the IDF uses the standard user interface functions described earlier. Code list, Index, on-line help, and Detail Forms and all the facilities of the Structure Editor are available to aid the designer and the system administrator. Several of the pop-up Forms used in the IDF have already been shown in other chapters.

The application dictionary and the CASE interface

Because the application dictionary is simply a normal database, the relationships between all of these definitions can be diagrammed like any conceptual schema. The conceptual schema of the application dictionary is shown in Figure 6-2, along with the names of each entity.

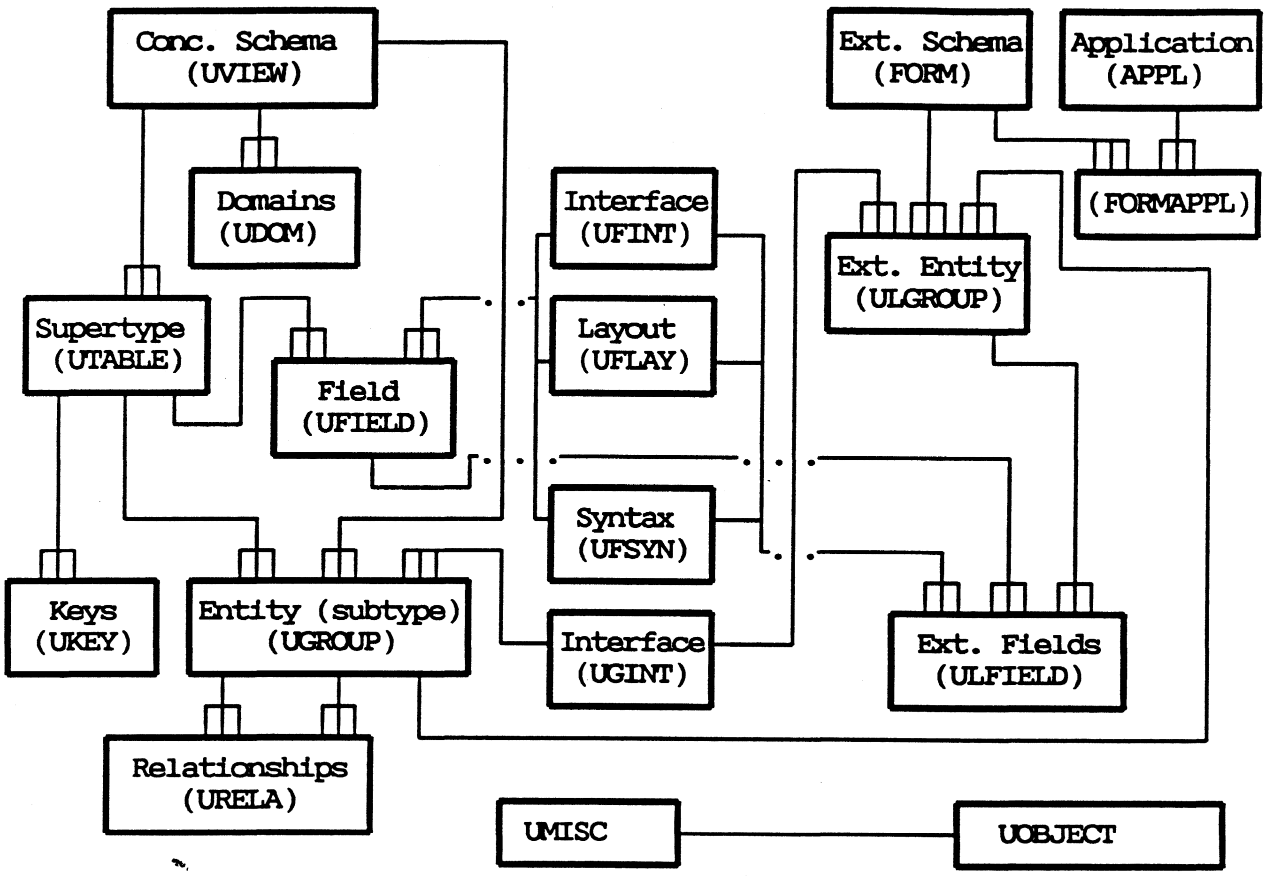


Figure 6-2. Conceptual schema of the application dictionary.

Nearly all CASE tools have an import-export format with the contents of the data dictionary: the entities, the fields, the keys, which fields and which keys belong to which entities, what the relationships between the entities are, etc.

The output of a CASE tool is usually an ASCII file containing a representation of the data model. A Uniface conversion program converts the various CASE interface formats to the Uniface interface format. The data model can then be read in, translated directly as the records in the diagram below. New entities are created, if necessary, by making extra UGROUP or UTABLE records.

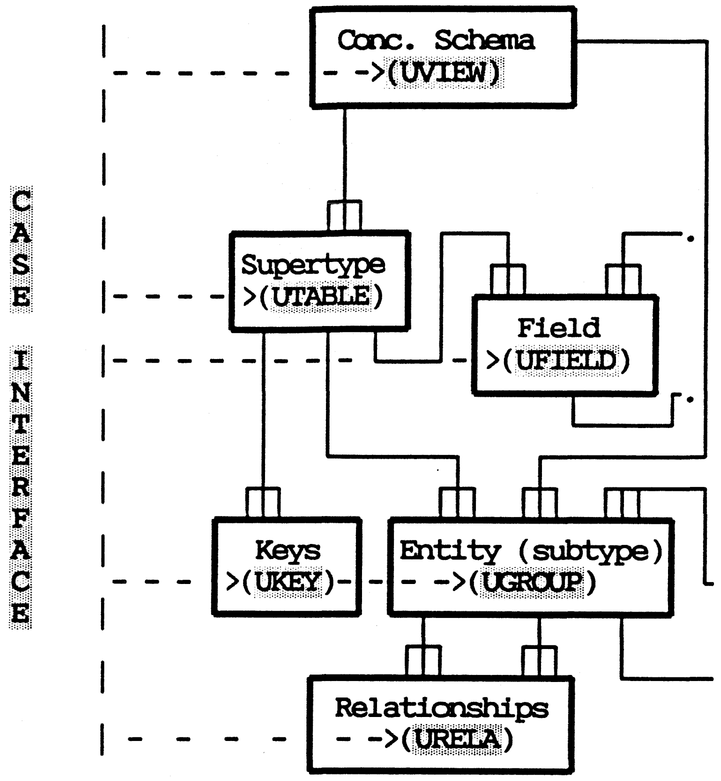


Figure 6-3. The IDF and CASE.

Chapter 7. Future Directions

This chapter describes the positions adopted with regard to several important current industry trends. These positions represent longer term directions for Uniface.

Graphic interfaces

The acceptance of graphic interfaces has clearly been one of the most significant developments in the computer industry in the last year. The success of Sun, Apple and PC workstations is undeniable. Any supplier of front-end user interface software must react to this trend. Adapting Uniface to use a graphic interface is an important direction for the development process in the long term.

However, low cost character-based terminals currently dominate the market and we feel that will continue to be the case for some time to come. Uniface is designed to provide attractive, user-seductive applications on these terminals, something our competitors cannot provide. The character-based market will continue to be Uniface's main market niche for the next several years.

Mainframes, networks and distributed processing

Despite the growing importance of mini and micro computer systems, mainframe systems will remain a very important part of the market-place for many years to come. However, the way mainframe resources are used is changing.

More and more often the mainframe forms a central backbone around which a variety of other equipment is clustered. VAXes, UNIX machines, and PC LANS are no longer looked at with distaste by MIS department heads. Integration rather than pure horsepower is increasingly important. Mini and micro systems are best suited for front end and departmental processing while mainframes hold central data needed at all levels of the organization.

We do not believe that a product like Uniface belongs on a main-frame. A character-based user interface like Uniface's is relatively CPU intensive and not appropriate for a block-mode environment. Uniface applications must be able to access mainframe data, but from a more appropriate platform.

Measured in cost per machine cycle, mainframe processing power is far more expensive than on a personal computer or a mini system. Therefore, it makes much more sense to run the front end user interface on a separate machine than the corporate mainframe. By removing this class of processing from the mainframe, the client's hardware investment is made that much more valuable.

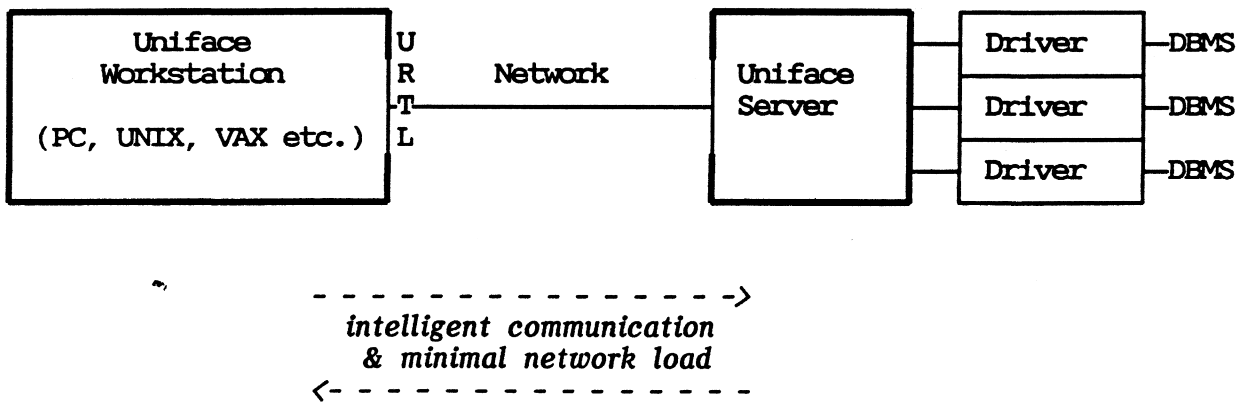
Our strategy is to implement connections to mainframe environments based on the Uniface Network architecture described below. We hope to implement the first connections sometime in 1990.

Uniface Network

Our network support is designed to implement distributed processing using a "Uniface workstation". The user's application can run on a relatively small machine, while accessing data on a larger dedicated database server.

Uniface network support is independent of any network support offered by individual DBMSs or operating systems. It is currently possible for Uniface to use facilities supported by the DBMS or operating system, if the interface is transparent to our software. However, we feel that this arrangement does not offer enough flexibility or performance.

We intend to split Uniface into two parts. The larger part of the run time system resides on the workstation (or "client"), and a smaller part (the "Uniface Server" and DBMS drivers) resides on the host.



URTL = Uniface Run Time Library

Figure 7-1. Uniface network support.

Before accessing an entity, the workstation sends a structure description over the network to the Uniface server on the host. Each I/O command then refers to this information. The Uniface server directs the DBMS driver to get the data from the DBMS and then sends it back over the line to the workstation. The advantages of this approach can be summarized as follows:

Performance

Splitting the processing load between two machines allows users to make use of the cheap processing power offered by personal computers. The value of the client's hardware investment is therefore substantially increased.

Network load

This architecture requires far less network load than most existing distributed processing environments. A preliminary measurement against Oracle's SQL*NET showed that we used 1/10th the network resource. Our architecture has a certain amount of intelligence built into it; the Uniface server understands how to communicate optimally with Uniface on the workstation. It is not just a communications protocol. Therefore, the network only needs to carry relatively small I/O commands rather than complete SQL instructions. Automatic field subsetting (supported since V4.0.p) also ensures that the network carries only the necessary data.

Flexibility

Uniface has always offered complete hardware and software independence but the workstation concept extends this independence even more. Each workstation can direct several different Uniface servers on different machines, or on the same machine. This means that the user can edit data from a variety of different backends. There can even be several different DBMSs on each backend.

System independence

Uniface is written in the "C" language with few system dependencies. Therefore, Uniface can be configured relatively easily to operate on different systems. Uniface currently runs on VAX-VMS and VAX-ULTRIX, PC/MS-DOS, STRATUS (VOS) and a large variety of UNIX machines:
ACER 19K
BULL DPX2000
CADMUS 9600
HP9000/300, /800
IBM RT (AIX)
NCR Tower 400, 600, 800
NIXDORF TARGON 31
SIEMENS MX500
SUN/3
XENIX 286,386

OS/2 will be available on request, and a port to the IBM AS/400 is planned for later this year.