DIALOGUE: Providing total terminal independence*

by DAVID VASKEVITCH

Standard Software Limited
Toronto, Ontario

ABSTRACT

A software tool, called DIALOGUE, makes application programs completely terminal-independent so that they can be used from both formatted screens and character mode terminals. The independence is achieved by providing programmers with a high-level record definition language for describing the data. This language isolates the programmer from the details of terminal interaction so they can be automatically handled in the most appropriate fashion at execution time.

To support various terminal types most effectively, DIALOGUE may generate a different interface for each brand of terminal. A unique aspect is its typewriter interface which supports character mode terminals so that they are indistinguishable (to the application) from formatted screen terminals. Devices such as light pens and OCR readers are also supported.
1.0 INTRODUCTION

In a transaction-oriented environment, formatted screen terminals offer the potential of a user interface superior to that which has been historically possible.

Special control characters and escape sequences must be sent from the computer to the terminal to take advantage of screen formats.

This raises several issues:

1. Writing programs to support a formatted screen is technically complex.
2. Terminals made by different vendors are incompatible with each other.
3. Older terminals and hard-copy terminals do not support screen formats.

The result is that using formatted screens is complex, locks the user into particular equipment, and renders existing terminals useless.

A software tool, called DIALOGUE, solves the problems associated with formatted screens by doing the following:

1. Providing a high-level interface that makes it easy for the applications programmer to develop user interactions that use screen formats.
2. Supporting a variety of formatted screen terminals so that no lock-in takes place.
3. Supporting character mode terminals so that the application cannot distinguish them from screens, ensuring that any application can be accessed from any terminal.

2.0 THE RECORD DEFINITION LANGUAGE

The central concept in DIALOGUE is that transactions are described in terms of records and that the description occurs at a high enough level of abstraction that the details associated with terminal support are concealed. Later, DIALOGUE can take a record description and decide how best to interact with a user based on the facilities at the user's terminal.

A record definition consists of two components:

Form: The form definition describes the fields making up the record, along with "decorations," such as titles that make the record more understandable.

Commands: Associated with every form is a set of commands representing the valid actions for a user. A form may be associated with several command sets over time (one at a time). DIALOGUE does not actually execute the commands; it uses them to establish valid function keys (at a screen) and signals the application when a command has been entered.

2.1 Describing Records

An example of a record definition is shown below. Several key points will be of interest to most readers:

Fields: The most important elements in the record are fields. Each field has a "prompt" and a "value" and various other optional qualifiers.

Edit Checks: Both alphabetic and numeric fields are supported, along with various other edit checks and transformations (such as shifting to upper case, e.g., SHIFTED).

Groups: Fields may be grouped for users' convenience. On the screen, groups of fields are surrounded by boxes (when possible). On the typewriter special commands allow groups of fields to be skipped in one step.

Syntax: The definition language is free format. Special care was taken to make statements self-identifying so that delimiters, such as semicolons, are not required. All keywords may be abbreviated as long as they are unique.

One of the key design criteria for the definition language was that it be easy to use for both programmers and end users. In this way, record definitions could serve as part of the design interface between an end user and an analyst. The use of English words and the simple syntactic rules have helped make this possible. (See Figure 1.)

2.1.1 Automatic positioning

When describing a form to DIALOGUE, the programmer does not need to specify where the fields and titles are to be placed on the screen. DIALOGUE will automatically position the fields so that they look good to the human eye—a tricky proposition.

The rules used to lay out the form consist of a set of heuristics dealing with the eye's tolerance for misalignment along vertical sight lines. The rules work completely about 80% of the time; and when they do not work, usually only one or two fields need to be adjusted. Naturally, the programmer can override DIALOGUE for one or more fields at any time.

Automatic positioning is important for three reasons:

• Terminal independence—Different-size screens (e.g.,
FORM 'MAILFORM' (Comments are enclosed in brackets).
TEXT 'MAILING LIST QUERY' CENTER UNDERLINE.
GROUP 'IDENTIFICATION' CENTER
  FIELD 'SURNAME' ALPHA 20 MANDATORY SHIFTED
  FIELD 'NAME' ALPHA 15
  FIELD 'INITIALS' ALPHA 3
GROUPEND
GROUP 'ADDRESS' CENTER INVERSE BLINK (A STANDOUT).
  FIELD 'CITY' ALPHA 20 NOCLEAR
  FIELD 'STATE' ALPHA 20 NOCLEAR
  FIELD 'COUNTRY' ALPHA 20 NOCLEAR
    SELECT 'CANADA USA'
GROUPEND
FIELD 'AGE' NUMERIC 2 RANGE 20 . .45
END
COMMANDS 'MAIL'
  NAME 'ADD' KEY 1 KEY 9 EXPLANATION "Add record to database"
  NAME 'QUERY' KEY 2 POINTER EXPLANATION "Retrieve record"
  NAME 'END EXIT QUIT LUNCH' KEY 3 NOREAD EXPLANATION "End of Program"
END

Figure I—A sample record

24 × 132 and 33 × 80) may contain similar amounts of information and yet require totally different screen layouts. In addition, the presence or absence of boxes and display attribute characters may require differing field positioning from one terminal to another.

- Programmer productivity—Calculating field positions is time-consuming, tedious, and error-prone. Furthermore, the addition or deletion of a field may make it necessary to reposition all subsequent fields.
- User readability—Not having to specify x and y coordinates makes a form definition easier to read (and write) for the end user.

On balance, of course, automatic positioning is effective only if it does produce good-looking forms; and experience has shown that it does.

2.2 Commands and Function Keys

Given a form definition, a transaction can still not be completed until the user enters a command signaling the application that a record is ready to be processed. Commands are defined in sets. An example of a command set was shown as part of the "Mailform". Command sets may be written as part of a total transaction definition (Form and Commands) or separately. When written separately, the Form and the Command set become associated by application-level subroutine calls at execution time.

At a screen, a command becomes associated with one or more function keys (e.g., QUERY = F1 or F9), while at a character mode terminal the command is invoked by name (e.g., "QUERY").

3.0 THE SCREEN INTERFACE AND DEVICE INDEPENDENCE

At a formatted screen DIALOGUE interprets the record definition automatically to generate a form on the screen based on the capabilities of the terminal. In providing this degree of device independence two key design principles were followed:

- GREATEST COMMON MULTIPLE—One way to support a variety of devices involves the lowest-common-denominator approach—i.e., support only features found on all terminals; the more terminals handled this way, the fewer the features supported. DIALOGUE takes the opposite approach: an honest attempt is made to support all the useful features found on each type of terminal. In large part, this is possible because the Record definition language leaves most of the decisions about form presentation to DIALOGUE.
- FEATURE INDEPENDENCE—Programmers should not be able to build feature dependencies into applications. Thus, when a feature is supported that is not universally present on terminals, it is always supported in such a way that applications are not restricted by the absence of the feature. A particularly striking example is given later in the discussion of pointers.

3.1 Display Enhancements, Edit Checks, and Boxes

The visible appearance of the form on the screen is established through a series of control characters and escape sequences which call on various features found in the terminal. These features include:

Display Enhancements:
- Fields (and other areas) may be highlighted by using display enhancements, such as INVERSE VIDEO, DIM, UNDERLINE, SECRET, BLINKING. These may be used alone or in combination, and they may be used to show a field in its normal state (e.g., the "blanks" in a form) or to flag errors (e.g., BLINKING). When available, unusual enhancements, such as color, may be used as well (e.g., flag a field in RED). Installations may choose the best enhancements for each terminal type to take advantage of its features.

Edit Checks:
- When possible, the terminal is asked to enforce edit checks directly, reducing the load on the computer and providing the user with more immediate feedback. However, if the terminal does not support an edit check, or supports it incompletely, it does not matter, because DIALOGUE will perform it instead.

Boxes:
- When supported by the terminal, boxes are always drawn around groups of fields. When available, vector drawing and repeat instructions are used to draw the boxes faster.

Pages:
- Multiple pages of memory are automatically used to store forms for reuse. This
considered to be a typewriter by DIALOGUE. Typewriters

4.0 THE TYPEWRITER INTERFACE

Any terminal that is not supported as a formatted screen is considered to be a typewriter by DIALOGUE. Typewriters include

1. Hard-copy terminals
2. Dumb screens (e.g., glass teletypes, word processors, etc.)
3. Screens operating at speeds too low for forms (e.g., 300 baud)
4. Screens not currently supported in forms mode

Users may also choose to run in typewriter if they prefer it.

The typewriter proceeds by prompting the user for each field (in a fashion familiar to any timesharing user). It should be immediately obvious that this process alone provides functional equivalence to the screen, because the end user can enter all the fields in a record using both interfaces (typewriter and screen). Further, the application cannot distinguish between the two interfaces: in either case it receives a complete, edit-checked data record.

The typewriter interface must do more than provide a means for entering data; it must allow the user to quickly and easily modify previously entered fields, as would be possible on the screen. Furthermore, it must allow the user to request a formatted display of his/her context (the record) and provide facilities for expert users to speed up their interaction. The mechanism used to accomplish this is the typewriter command.

4.1 Typewriter Commands

DIALOGUE’s typewriter interface allows the user to enter a command at any time. Commands are distinguished from field values by an installation-defined unique character, typically the period. Special care is taken to ensure that this character can still be used in other contexts (e.g., entering numbers), since otherwise it would be impossible to choose a character without conflict.

Commands are divided into two categories: internal (or DIALOGUE) commands directed to DIALOGUE, and application (or external) commands that provide the user with function keys.

Help facilities allow the user to see a list of commands (internal, external, or both) and ask for an explanation of each. The previous example showed how explanations are specified for application commands. Commands may always be abbreviated as long as the abbreviation is unique within the current command set.

4.1.1 Typewriter Commands

A set of approximately 35 commands provides the user with complete facilities for examining and modifying records as and after they are entered. For example, when receiving an error message, the user can display the entire record (e.g., DISPLAY), modify a field (e.g., MODIFY SURNAME), and then reenter the record using an application command.

Special provision is made for the first-time user, who may not know any commands. First, the programmer can designate a default application command which is used if the user simply enters all the fields in a record and “falls off the end.” Second, if one or more fields are then flagged, DIALOGUE will cycle the user through the flagged fields showing him the value in each (e.g., SURNAME (WASHINGTON) - ). The user can keep the value or replace it by typing in a new one. When the user has cycled through all the flagged fields, the record is re-entered for him/her. Thus the user need not know commands or have to reenter the entire record for one mistake.
4.1.2 Advanced Commands

Several advanced facilities can make the typewriter interface uniquely efficient for expert users:

Automatic Responses: If a field does not change, the user can establish an automatic response with the “remember” command (e.g., REMEMBER SEATTLE). Once an automatic response is set up, the associated field literally disappears from the transaction so that the user no longer needs to deal with it.

Resequencing: Using a single command (DETOUR), users can rearrange the sequence of fields to please themselves. The application still sees the original record and field sequence.

Typeahead: DIALOGUE’s typeahead feature allows the user to anticipate any number of fields and enter their values, all at once, separated by commas (e.g., WASHINGTON, GEORGE, H). Once a field is anticipated by the user, its prompt is suppressed. This allows very terse input, since the user can enter any amount of information at any time. Typeahead may cross record and program boundaries.

4.2 Relative Efficiency

Although formatted screen mode is always easier to use, typewriter can actually be more efficient for the expert user because

1. Extraneous or constant fields can be totally suppressed with automatic responses.
2. Input may occur in any sequence as a result of detours.
3. Multiple records can be entered at one time using typeahead.

One net result of the efficiency of the typewriter is that expert users may use it in preference to block mode. This ensures that screen formats can be designed for ease of use, knowing that the typewriter is there for experts.

5.0 ALTERNATE INPUT DEVICES

A variety of devices may be attached to a terminal, allowing input alternatives to the keyboard. These include OCR wands, bar code readers, magnetic-stripe readers, and voice recognition units. These are essentially “field input” devices that can cause one field at a time (typically) to be entered and are known as readers.

DIALOGUE allows a field to be designated as READABLE. If a form contains a readable field, the reader can be used if present; otherwise the keyboard is used. If only one field is readable, reader input is always placed there. If several fields are readable, the cursor position is used to determine where to place a reader input.

This scheme allows readers to be used when present without either requiring their presence or requiring special application code. A particularly interesting scenario can even be imagined involving both a touch-sensitive screen (a pointer) and a voice recognition device (reader).

6.0 USING DIALOGUE

To the programmer DIALOGUE has three major components:

Forms: Form definitions are usually written in edit files. These files are then referenced by name in the application. This degree of separation allows simple form changes to be made without even recompiling the application.

In addition, form definitions may also be generated programmatically by the application.

Commands: Although usually written as part of the form definition, command sets may be established separately.

Subroutines: The subroutine gives the programmer control over the transaction. Most of the subroutines have self-explanatory names; the key ones are listed below:

- SETUPFORM (editfilename)
- READRECORD (data-area)
- WRITERECORD (data-area)
- ERROR (fieldname, message)
- MESSAGE (message)

Other routines provide explicit cursor control and other detailed functions used less frequently than those above. Great care was taken to ensure that most work could be done with fewer than 10 subroutines.

A particularly attractive aspect of DIALOGUE is its natural-language structure. It appears equally attractive to programmers in COBOL, FORTRAN, Pascal or even BASIC. Unlike many other terminal handlers, DIALOGUE has no bias toward any one language.

7.0 CONCLUSION

DIALOGUE has been running in a commercial environment on the TANDEM computer for about two years and has been very successful there. Over 20 terminal types are supported, and both screen and typewriter mode are in active use. Both end users and programmers seem to like DIALOGUE, and the concept of terminal independence has proved workable.