Introduction

xmbase-grok was born as "grok", but the name was changed later to this more descriptive variation. It is a simple desktop database manager, intended to manage small databases such as phone lists, diaries, todo lists, URL lists, etc. A user interface builder is integrated in xmbase-grok to allow customization of the presentation of the data in the main window. A variety of user interface elements such as text entry fields, checkbuttons, function buttons, and bar charts ca be placed in the database-dependent part of the main window, called the *card*. xmbase-grok comes with a selection of pre-built applications.

The database itself is organized in a table with rows and columns of strings. A *card* presents one row of the database. For example, the phone list stores one person per row, with various columns for name, address, phone number, and so on.

In addition top the card, the main window also displays a scrollable *summary* listing containing one line per card, allowing fast lookup of cards. Pressing on a line in the summary puts the corresponding row into the card part of the window. The main window also contains controls for searching of cards, and for adding and deleting cards. There are several methods for searching: by keyword, using the built-in query language, or using customizable standard queries.

This chapter is intended as a user manual. For details on the implementation of databases, and for advanced operations such as building new applications with xmbase-grok, refer to the chapter "Creating and Editing Forms".

Starting xmbase-grok

From the command line, grok can be started as `grok` with no arguments. This will bring up an empty card; a database must then be chosen from the *Database* pulldown. The database to load can be specified on the command line, such as `grok phone`.

The `-t` and `-T` options allows queries printed to stdout, without starting the interactive graphical user interface. Queries may use simple keyword lookups, or may use the query language:

```
grok -t phone thomas
grok -t todo '({_status == "t"} && {_assigned == user})'
```

For details on the query language used in the second example, refer to the "Expression Grammar" chapter. The first example looks for the given keyword; case distinctions are ignored. The author of the application controls which columns of the database are searched for keywords. Note the single quotes enclosing the query to protect blanks, parentheses, and double quotes from the shell.

xmbase-grok supports the following command-line options:

`grok` [*options*] [*database* ['*query*']]

`-h` print a short usage message listing the available options.

`-d` print the default X resources to stdout and exit. The output can be appended to the local `~/.Xdefaults` file or other resource files, after changes to customize xmbase-grok's behavior.

`-v` print the version number of xmbase-grok.

`-t` Perform *query* and print the matching cards to stdout. If *query* is missing, print all cards. The *database* argument is mandatory for `-t` and `-T`.

`-T` Same as `-t`, but omit the header line. This is useful when the result is piped to other Unix utilities.

`-f` Do not fork on startup. This is mainly a debugging tool; it prevents xmbase-grok from putting itself into the background.

`-r` Restricted `grok`. The form editor is disabled to prevent inexperienced users from inadvertently starting the user interface builder (called the "form editor") and modifying the current database format. The same effect can be achieved by linking `grok` to rgrok; running `rgrok` has the same effect as running `grok -r`.

The Main Window

When xmbase-grok is started, it presents a four-part window: a search area for entering keywords and query expressions, the summary, the card, and a row of buttons for creating, deleting, and modifying cards. There is also a menu bar with pulldown menus.

When xmbase-grok is started up without arguments, it is first necessary to choose the database to display from the `Database` pulldown. This pulldown presents all files in the `~/.grok` directory that end with the extension `.gf`, with the extension removed. (If a new database is created, it is not necessary to choose a database, see next chapter.) Sometimes, the Database pulldown shows some databases more than once. The reason is usually that the database is found in two different places or using different paths; the duplicates can be eliminated by enabling the "Don't show duplicate databases" in the preferences menu available under the "File" pulldown.

The next step is choosing a *section* of the database. Not all databases have section; if the `Section` pulldown in the menu bar is grayed out, this step can be omitted. xmbase-grok normally loads its string table containing the cards from a file ending with the extension `.db`, usually with the same directory and root name as the `.gf` file. If this `.db` file is a directory, xmbase-grok recursively loads all files ending with `.db` in this directory, and combines them into one database in memory. Each file becomes a *section*. By default, all sections are displayed after loading a database, but the Section pulldown allows restricting the summary listing to a single section (file). To create a new section, choose "New..." in the Section pulldown; if the database did not have sections until now all its cards are put into the new section. Otherwise, the new section is created empty. There is no way to delete sections, other than using a shell to delete the file named *sectionname*`.db` in the database directory.

The main window now displays all cards in the chosen database, or section of the database if chosen, in the summary listing. To display any card listed in the summary, press on this line. The card appears in the bottom part of the window and can now be edited (unless the database file was read-only, or the application author has restricted write access). Text entry fields normally have a pink background; read-only text fields have a gray background.

There is a special multi-line version of text entry field called a *note*, which always has a gray background regardless of its writable status. On some systems it is necessary to press in the upper left corner of the inset area, or on already existing text to get a cursor.

Below the card, there are buttons to skip to the next or previous card, to create a new card (*New*), to duplicate the currently displayed card, or to delete the currently displayed card. New cards may be created with defaults specified by the application designer. If the database has sections, there is an additional buttons that moves the card into a different section.

xmbase-grok can optionally display a row of letter buttons below the summary area that restrict the summary to display only cards whose first text field begins with the selected letter. Case is not distinguished; leading white space and punctuation is ignored. The row of letter buttons can be enabled or disabled with the "Enable search by initial letter" mode in the preferences menu that can be called with the "Preferences" choice in the "File" pulldown. The "Letter search checks all words" mode in the same menu extends the letter search to check all words in the first text field of each card, instead of just the first.

Searches and Queries

The Search text entry field near the top of the window allows entry of a search string. Case is not significant. The entire string is searched for; the number of blanks and punctuation is significant. The search string is *not* a sequence of keywords. This type of search is equivalent to what the Unix command `fgrep -i` does.

More sophisticated queries can be performed by entering a query expression beginning with (or {. For a description of the syntax and the difference between parenthesized and braced expressions, the the "Expression Grammar" chapter. To get a feel for expressions, enable "Show query search expressions" in the preferences menu and choose a canned query from the "Query" pulldown. The query expression will be shown in the Search text field and can then be edited there.

The Query pulldown always begins with the "All" choice that puts all cards in the database, or all cards in the current section if one was selected with the Section pulldown, into the summary listing. The application designer may predefine canned queries (using the "Queries" button in the form editor) that appear in the Query pulldown.

If the "Incremental searches and queries" mode in the preferences menu is enabled, each new search or query only checks the cards already in the summary, thus narrowing the previous search or query. To return to a complete listing, choose "All" in the Query pulldown. If the incremental mode is disabled, all cards are checked.

Printing

The File pulldown has a "Print...' choice that pops up the printing menu. It allows selecting the cards to print, the output format, the output quality, and the output device. Medium quality uses overstrike Ascii to simulate boldfacing and underlining as supported by many programs such as *more* and most printer drivers. It uses character-backspace-character or underbar-backspace-character sequences to highlight field names and header lines.

If "Printer" is chosen as output device, the printout is sent to the Unix command line specified in the preferences menu. Typical print spooler strings are "lp" (on System V) or "lpr" (on BSD systems). It is also possible to enter shell commands such as "cat ⟩ /dev/tty". The PostScript printer is specified separately because a RIP program or special lp options may have to be specified; this is currently not used because the "PostScript" quality choice in the print menu is not yet implemented.

Miscellaneous

When a database is changed, the changes are not immediately written to the database file (with the `.db` extension). It is written when "Save" or "Quit" in the File pulldown is chosen, or when a new database is chosen from the Database pulldown. Under mwm-based window managers, double-clicking the top left corner button in the window decoration also saves, but there may be window managers that do not support the necessary WM protocols for this. The status line below the search text field displays "(modified)" if the database has been changed but not written back.

Note that xmbase-grok does not attempt to sequence database accesses. It does not protect databases against simultaneous accesses. Although standard file locking is used during read and write operations (which may or may not work across NFS), nobody stops two users from reading the same file, then both modifying it, and writing it back; one of the two changes will be lost.

A useful feature is the "Current database" choice in the Help pulldown. It lists all known information about the current database, including the names and paths of all loaded files and their sizes.

In case of trouble, it is **strongly** recommended to read the other help texts available under the Help pulldown, especially the "Troubleshooting" section. The Help pulldown will only work if the `grok.hlp` file is installed in the directory specified in the GLIB macro in the Makefile (Imakefile, or Makefile.alt), by default /usr/local/lib.

Creating and Editing Forms

xmbase-grok distinguishes databases and forms. A database is an array of unformatted data, while the form specifies the structure of the data and describes how to display it in a card. The standard user interface that comes up when xmbase-grok is started deals with database presentation and modification; to edit the presentation, or to create a new database, start the form editor from the File pulldown in the main window.

The form editor is a separate window that is rather more complicated than the standard database user interface. It is basically a simple UI builder that allows the user to create and position UI elements in a card.

General Setup

The first step when creating a new database is choosing a form name. This is the name that will appear in the Database pulldown in the main window (actually, the pulldown should be called Form, but I fear that would confuse casual users). Every form references a database whose contents it presents; this name must also be chosen. Typically, both names are the same.

Both the form name and database names are also the file names the form and the database will be stored in. The form name gets the extension ".gf", and non-procedural databases get the extension ".db" tacked on if the names are not fully qualified (i.e., do not begin with "/" or "~"). If the database is procedural, the database file is a script, and has no .db or any other extension. This script is executed to read or write data.

When a database or form is read, the path it was read from is stored; when the database or form is changed, it is written back to that path. When a new database is created, and its name does not begin with "/" or "~" as defined in the second line of the form editor, it is stored in the same directory as its form file. The default is always ~/.grok. The Help→Database popup shows which paths are actually used.

The database is a two-dimensional array of strings. The rows are called *cards*, and the columns are called *fields*. Rows are separated by newlines, and columns are separated by the field delimiter. The field delimiter is a colon by default, but can be changed to any character. The button accepts characters, octal constants *nnn*, and the tab character `\t`. Any character other than *0* and newline can be chosen; xmbase-grok will properly escape the character when it appears in a database string. Severe chaos may result if the delimiter character is changed when the database already contains data.

Databases can be marked read-only. A user accessing a database through a form that is has the read-only flag set will not be able to change any cards, and will not be able to write back.

A procedural database does not read a file, but calls a script that provides the data in the same format that the file would contain. If the `procedural` button is turned on, the referenced database name is the name of the script, not of any database file. It may contain options. When the database is read, xmbase-grok appends the option `"-r"` (read) and the form name as shown in the `Form name` button (without prepending a path or appending an extension). When writing, `-w` is appended instead of `-r`. The script must print the data to stdout if `-r` is specified, or must accept the data from stdin if `-w` is specified, separating columns with the field delimiter character and separating rows with newlines. If the delimiter or newlines appear in as part of a data string, it must be escaped with a backslash.

A comment can be specified that should give the name of the author of the form, or special caveats. The comment is displayed only in the form editor window.

Creating a Card

After the general setup is done, fields can be arranged on a card "canvas", which has the same size and layout as the final card will have, but doesn't look as nice and shows extra information. Fields appear as blue rectangular boxes, some of which are divided in the middle (depending on the field type). The current field, whose specification is displayed in the form editor window. A field can be moved by left-clicking somewhere inside the field (but not too close to an edge) and dragging. The size can be changed by dragging one of the four edges, and the divider can also be dragged. Fields should not overlap. The card canvas can be resized; fields should not overlap the canvas window edges.

The canvas is divided in two parts by a horizontal fat line. The divider can be moved vertically by dragging the little square that initially appears near the top right corner of the canvas. Everything above the divider is the *static part*; everything below the divider is the *card part*. The card part displays one row of the database if one is selected; this information changes frequently whenever a search is performed or a row is chosen from the summary.

The static part does not normally change, it is intended for static data such as the average of all fields, a chart displaying statistics, or buttons. This part is not entirely static because entering new cards or resorting the database may change data, but it is not bound to a particular card, and it remains accessible if no card is chosen. This makes it a good place to place form switch buttons that would otherwise become unavailable when no card is displayed.

There are several types of fields. Not all of them store data in the database; some are decorative or display computed information.

Input This is the main type of field. It displays an editable (unless turned off) string in the database, along with a label. Input fields should not be put into the static part of the canvas.

Time A variation of the Input field. The database representation is a number of seconds. It is displayed as a date, as a time, as both date and time, and as a duration. The first three assume the database string to be a number of seconds since January 1, 1970; the last simply assumes a number of seconds up to 86399 (one day minus 1 second). When a string is entered into a Time field, it is converted to the numeric representation, reformatted, and reprinted. Time fields are useful because they can be used in expressions for calculation; expressions always see the numeric database string. Time fields should not be put into the static part of the canvas.

Note A note is a multi-line Input field. It should be used only for multiline text input because it cannot be tabbed over, and because pressing Return when entering data into the card into a Note actually inserts a newline, rather than skipping to the next field as an Input-type field would. Note fields should not be put into the static part of the canvas.

Choice Unlike all other types, many choice fields reference the same database string. They all must have the same summary column, the same database column, and the same internal field name (these three are buttons in the form editor). They differ only in the Choice/flag code. xmbase-grok always makes sure that only one of the choice fields with identical internal field names can be active at any time; the database string then matches the Choice/flag code of that field. Most attributes of a `Choice` item, when changed, are copied to all other `Choice` items that have the same internal name. Choice fields should not be put into the static part of the canvas.

Label Labels are purely decorative. They print an arbitrary one-line string at a position in the card. There is no associated database string. Labels are rarely needed because most of the other types come with their own built-in label parts. The label is static, expressions cannot be used.

Print Print fields are like Input fields, but no text can be entered. Unlike labels, they can display an expression specified in the Input Default button of the form editor. This can be used to display a running average or sum in cards, or display other computed information. There is no associated database string. Print fields are useful in both the static and card parts of the canvas.

Flag Flags are boolean database strings: the string either matches the predefined string (*true*), or it is empty (*false*). (In fact, a string that doesn't match is also considered *false*, but this is not part of the normal operation.) The string that constitutes *true* is specified with the Choice/flag code button in the form editor.

Button Buttons have an associated action expression that is executed when the button is pressed. This action could start a shell script, for example. Buttons are not associated with any database string, but the expression can access one. For example, a database of demo programs can have a button that executes the program. The returned string is executed, there is no need to use the `system` keyword unless nesting is desired. Note that the action expression is the only type of expression that may contain `switch` statements; see the Expression Grammar chapter for details. It is often a good idea to put buttons in the static part of the canvas.

Chart Charts display data as bar or line charts. The X axis is divided into one slot per row in the database; the Y axis depends on the values computed from those rows (X and Y may be exchanged). Each chart contains one or more *components*. A component computes values that is plotted in the chart; a chart may display more than one value. For example, an expense account chart may display a stacked bar chart consisting of different color-coded types of costs, each described by a component. There are many variations for configuring charts and their components. Charts should be put into the static part of the canvas.

Each field has a number of parameters that depend on the type. The most important is the internal field name. It must be unique, except for choice fields which are grouped by assigning common internal field names. If the field references a database string, the internal field name also names the database string. The internal field name can be used in expressions to read the database string. For example, suppose you have a database of backup tapes, you may have an `Input` field with an internal field name *capacity*, and another `Input` field named *used*. You could then add a `Print` field whose Input Default expression is `(_capacity - _used)`. The `Print` field then displays the remaining free space on each tape, even though the database only contains total capacity and used capacity. Another `Print` label may have an expression `(sum(_used))`, which displays a running total of all tapes' contents. `sum` is one of a group of functions that loop over all cards rather than just referencing the current card; see the Expression Grammar chapter for details. It is also possible to reference a database field for which there is no field description in an expression; in this case, the field is referenced by number. Fields in the database are numbered left to right, beginning with 0.

The next button in the form editor is the database column. It needs to be specified only for field types that display the column and allow entry into the column. These fields are "windows" into the database; there is normally one field for each database column. All fields that do not reference a database column are merely decoration, no change of the database is possible through such a decorative field (although the decorative field may read the database, as `Print` fields do). This relation between fields and database columns also serves to give a symbolic name to database columns; these symbolic names can be used in expressions by prefixing them with an underscore. (It is also possible to use the column number in expressions, but that is less convenient).

The main window has three parts, a summary, the static area, and the card. The summary contains one line per card, while the card contains the entire card's information as defined with the form editor and the card canvas. The static part is optional and programmed in the same way as the card. The Summary column and Width in summary buttons in the form editor determine which fields also appear in the summary; this is a subset of the fields that reference a database column (decoration fields can not be put into the summary). The two buttons specify the order in database column and the width in characters. Two blanks are inserted between fields in the summary automatically. The summary has a title; it can not be specified directly but is taken from the Label text of the field.

Here is a brief summary of all buttons in the form editor that specify a field in the card:

Field type	The type of a field is entered here. See above for a list of available types and what they do.
Searchable	The main window contains a Search input button. It searches through all cards and puts all cards containing the search string into the summary. Fields that are not searchable are excluded from the search.
Read only	The user cannot change the database string referenced by a field that is read-only. This is useful if there are two forms referencing the same database, one for you and one for the unwashed masses with lots of read-only flags set. The read-only flag can also be set for the entire database with the button near the top of the form editor window; setting that flag overrides all field read-only flags.
Not sortable	The field will be omitted from the Sort pulldown in the main menu.
Default sort	When the file is read in from disk, it is sorted by the field that has this flag on. Setting it in any field will clear the Default sort flag in all other items automatically. If no field has the Default sort flag, the file will not be sorted when it is read. It is possible but not recommended to have both the Default sort flag and the Not sortable flag on in the same field.
Internal field name	All fields have an unique name. Choice names are not unique, choice fields are grouped by a common name. If the field references a database string, the internal field name also names the database column, which can then be accessed in expressions symbolically.
Database column	If the field references a database column, this button says which one, 0 being the first column. If not, this button is grayed out. The column number must be unique, except for choice fields which are grouped by a common database column (and a common internal field name, too).
Width in summary	If the width is nonzero, the database string referenced by the field will appear in the summary, with as many characters as specified. Two blanks are appended. The summary always uses monospaced Courier to make columns line up vertically.
Summary column	If the width is nonzero, this value specifies the order of fields in summary lines. No two fields may have the same summary column number, but there may be gaps.
Choice/flag code	The string that Flag and Choice fields store in the database, if active. No two Choice fields with the same internal name may have the same code.
Shown as	If this string is set, it will be displayed in the summary in place of the choice/flag code. Basically, it is a mnemonic name for the choice/flag code that a user can understand.
Time format	Time fields have four different formats, as described above. The format controls what gets printed into the card, and how user input is interpreted.
Label text	All field types come with some kind of text string that is printed into the field in the card. This string is always literal, it cannot be an expression.
Label justification	Labels can be centered, left-aligned, or right-aligned. This is not shown in

the card canvas, press the Preview button to see the effect.

Label font The font used for the label. Five fonts are available.

Max input length The maximum number of characters than can be entered into an Input, Time, or Note field. The default is 100 for Input and Time fields, and 10000 for Note fields. Always make sure that note fields have a sufficient maximum length. This number is passed to the Motif widget to limit input length, but does not lead to increased memory usage for the database.

Input default For Input, Time, Flag, and Choice fields, this field provides the defaults when a new card is added to the database. It can be an expression. For Print fields, the Input default specifies what gets printed into the inset area of the field; input default is actually a misnomer because Print field texts cannot be input and are evaluated whenever the database changes, not just when a new card is added. In general, Choice fields should always have a default. If the field has type Time, the input default expression should evaluate to a number of seconds, not to a string containing a date. For example, to make the Time field default to today, use (date), not date.

Input justification Input can be centered, left-aligned, or right-aligned. This is not shown in the card canvas, press the Preview button to see the effect.

Input font The font used for the input area. Five fonts are available. It is recommended to use Courier for Note fields (and, by extension, for Input and Time fields) because printing functions print notes using a fixed-width font.

Grayed out if If the named expression evaluates to *true*, the field is grayed out and cannot be used to alter the database. The expression is evaluated every time the database changes.

Invisible if If the named expression evaluates to *true*, the field is excluded from the card. The expression is evaluated only once, when the database is read from disk. This can be used to hide entries if the wrong user has read the database. Invisibility does not affect the summary.

Read-only if If the named expression evaluates to *true*, the field is read-only. The expression is evaluated only once, when the database is read from disk.

Skip if Normally, pressing Return in an Input or Time field advances the cursor to the next field (fields are ordered by their bottom left corner, in Y-major order). If the named expression of the next field evaluates to *true*, the field is skipped and the cursor is put elsewhere. This expression is evaluated every time return is pressed in the previous field. A constant expression such as `true` is also useful.

Action when pressed If the button is pressed, this expression is evaluated. The result is ignored. Typically, the expression is the name of a shell script. The expression may use the `switch` statement, which switches to another database and/or performs a query on all cards.

Chart flags Not documented yet. This part of the menu is still under development.

Some of the above accept expressions. An expression begins with a parenthesis, a brace, or a dollar sign. Everything else is a literal string. Parentheses and braces are numeric and string expressions, respectively; a dollar sign followed by an environment variable is a shortcut for the same sequence enclosed in braces. The `system` statement should be used sparingly, because some expressions (such as the grayed-out-if expression) are evaluated frequently. See the Expression Grammar section for details.

Buttons

There is a row of buttons in the form editor for various operations:

Queries Starts up a window that allows entry of standard queries, as name/expression pairs. The name is what will appear in the Query pulldown in the main menu; the expression is what gets executed if the name is selected in the pulldown. When a name is selected, the expression is applied to all cards in the database, and those that return *true* are put into the summary. For example, assuming your database has an Input field with the internal name `value`, the query expression `(_value ⟩ avg(_value))` will select all cards whose value is above average.One of the queries can be selected as the default query that will be performed when the database is read from disk.

Def Help The main window has a help button in the lower left corner. This button pops up a help window with some generic info about xmbase-grok. With the Def Help button, more text can be entered that will be appended to the generic help text. The text should explain the card, how to use it, and what the fields mean.

Debug This button checks the consistency of all fields, and reports conflicts such as non-unique internal names or redundant choice flags. At this time, expressions are not checked. If the Debug button reportrs nothing, the no problems were found. The Done button always does a debugging run first, and refuses to exit if errors were found.

Preview The card canvas shows the layout of fields in the card, as boxes that show additional information such as type, database column, flag/choice code, and summary column. This does not reflect the final card that the user will see very well; in particular, whether a label string fits into the field on the card canvas does not mean that the same label will fit into the final card. Preview shows precisely what the card will look like.

Help Print general help information.

Cancel Discards all operations done with the form editor since it was installed, and removes the form editor window after asking for confirmation.

Done Check all fields for consistency. If no problems are found, the form file is written. The file name is taken from the Form name button at the top of the form, with `~/.grok` prepended and `.gf` appended if appropriate.

Add Adds a new field to the card. Its type, parameters, and position on the card canvas are chosen based on the currently selected card, so it's a good idea to select a field that is similar to the new one before pressing Add. If the card canvas has no free space below the bottom field, the new field may be placed under the bottom field where it can't be seen; it is generally a good idea to start with a card canvas that is too large and resize it to the correct size after all fields have been added and positioned.

Delete Delete the currently selected field. There is no Undo function to get it back.

Expression Grammar

Expressions are used for queries, for defaults of card items, and for printing expressions into cards. They are set in Database Edit mode; a normal user does not deal with expressions directly.

Expressions deal with two data types, *strings* and *numbers*. Expressions or sub-expressions returning strings are enclosed in braces; expressions or sub-expressions returning numbers are enclosed in parentheses. There are many built-in operators and functions; most of them can be used only in either string or numeric context.

Numbers begin with a numerical digit or a period, and are in standard integer, floating-point, or exponential notation. String literals are enclosed in double quotes. Conversions from numbers to strings use the `%g` format (unless `printf` is used); conversions from strings to numbers skips leading blanks and converts like `atof`. Trailing non-numeric characters are ignored.

Expressions are interpreted, not compiled. This means that all parts of the expression are evaluated, `?:`, `&&`, and || do not short-circuit.

In the following tables, *n* stands for a number ot a numerical expression, and *s* stands for a literal string or a string expression. Note that some operators, such as `==` and `date`, appear in both contexts.

Numerical Operations

Divisions by zero return 1. Arithmetic operators use standard C precedences. Bitwise operations operate on 32 bits only.

<table>
<tr><th>Operator</th><th>Type</th><th>Operation</th></tr>
<tr><td>(n)</td><td>n</td><td>Number</td></tr>
<tr><td>{ s }</td><td>n</td><td>In number context, convert string to a number</td></tr>
<tr><td>n ? n : n</td><td>n</td><td>If the first number is nonzero, return the second number;
otherwise, return the third number</td></tr>
<tr><td>n , n</td><td>n</td><td>Evaluate both numbers, return second</td></tr>
<tr><td>- n</td><td>n</td><td>Unary minus</td></tr>
<tr><td>! n</td><td>n</td><td>Unary boolean NOT</td></tr>
<tr><td>~ n</td><td>n</td><td>Unary bitwise NOT</td></tr>
<tr><td>n + n</td><td>n</td><td>Add two numbers</td></tr>
<tr><td>n - n</td><td>n</td><td>Subtract two numbers</td></tr>
<tr><td>n * n</td><td>n</td><td>Multiply two numbers</td></tr>
<tr><td>n / n</td><td>n</td><td>Divide two numbers</td></tr>
<tr><td>n % n</td><td>n</td><td>Calculate modulo of two numbers</td></tr>
<tr><td>n & n</td><td>n</td><td>Calculate bitwise AND of two numbers</td></tr>
<tr><td>n && n</td><td>n</td><td>Calculate boolean AND of two numbers</td></tr>
<tr><td>n | n</td><td>n</td><td>Calculate bitwise OR of two numbers</td></tr>
<tr><td>n || n</td><td>n</td><td>Calculate boolean OR of two numbers</td></tr>
<tr><td>n ^ n</td><td>n</td><td>Calculate bitwise XOR of two numbers</td></tr>
<tr><td>n << n</td><td>n</td><td>Calculate bitwise left shift</td></tr>
<tr><td>n >> n</td><td>n</td><td>Calculate bitwise right shift</td></tr>
<tr><td>n == n</td><td>n</td><td>1 if both numbers are equal, 0 otherwise</td></tr>
<tr><td>n != n</td><td>n</td><td>1 if both numbers are not equal, 0 otherwise</td></tr>
<tr><td>n < n</td><td>n</td><td>1 if the first number is less than the second, 0 otherwise</td></tr>
<tr><td>n > n</td><td>n</td><td>1 if the first number is greater than the second, 0 otherwise</td></tr>
<tr><td>n <= n</td><td>n</td><td>1 if the first number is less than or equal to the second, 0 otherwise</td></tr>
<tr><td>n >= n</td><td>n</td><td>1 if the first number is greater than or equal to the second, 0 otherwise</td></tr>
<tr><td>sqrt (n)</td><td>n</td><td>Square root of a number</td></tr>
</table>

`exp` (*n*)	*n*	Exponential function, e^n
`log` (*n*)	*n*	Decimal logarithm, $\log_{10} n$
`ln` (*n*)	*n*	Natural logarithm, $\log_e n$
`pow` (*n* , *n*)	*n*	First number raised to the second, n^m
`sin` (*n*)	*n*	Sine of a number, $\sin x$
`cos` (*n*)	*n*	Cosine of a number, $\cos x$
`tan` (*n*)	*n*	Tangent of a number, $\tan x$
`asin` (*n*)	*n*	Arc sine of a number, $\sin^{-1} x$
`acos` (*n*)	*n*	Arc cosine of a number, $\cos^{-1} x$
`atan` (*n*)	*n*	Arctangent of a number, $\tan^{-1} x$
`atan2` (*n* , *n*)	*n*	Quadrant-aligned arctangent
`len` (*s*)	*n*	Length of a string
`bound` (*n* , *n* , *n*)	*n*	The first number bounded by a minimum (second number) and a maximum (third number)

String Operations

Note that string comparisons return strings, and must be enclosed in braces {} if && or —— or other numerical operators are used on the result.

Operator	*Type*	*Operation*
{ *s* }	*s*	String
(*n*)	*s*	In string context, convert number to a string
s ; *s*	*s*	Evaluate both strings, return second
s . *s*	*s*	Concatenate strings
s ? *s* : *s*	*s*	If the numeric value of the first string is nonzero, return the second string; otherwise, return the third string
s == *s*	*s*	Return `"1"` if the two strings match; otherwise, return `"0"`
s != *s*	*s*	Return `"1"` if the two strings do not match; otherwise, return `"0"`
s < *s*	*s*	Return `"1"` if the first string is lexicographically less than the second string; otherwise, return `"0"`
s > *s*	*s*	Return `"1"` if the first string is lexicographically greater than the second string; otherwise, return `"0"`
s <= *s*	*s*	Return `"1"` if the first string is lexicographically less than or equal to the second string; otherwise, return `"0"`
s >= *s*	*s*	Return `"1"` if the first string is lexicographically greater than or equal to
s `in` *s*	*s*	Return `"1"` if the first string is contained in the second string; otherwise, return `"0"`
`chop` (*s*)	*s*	Return the string with the trailing newline, if any, removed
`substr` (*s* , *n* , *n*)	*s*	Return a substring of the first string. The first number is the start index and the second the length. A negative index counts from the end.
`printf` (*args*)	*s*	Format and return a string; *args* is a comma-separated list of expressions. Compound expressions must be enclosed in ( ) or { }.

Variables

Variables are letters `a` through `z` that can hold strings or numbers. When a variable is assigned to, the result of the assignment is returned. All variables are reset to the empty string (or 0) when a database is loaded from disk.

Operator	*Type*	*Operation*
var	*s* ,*n*	Value of a variable
var `=` *s*	*s*	Assign string value to a variable
var `=` *n*	*n*	Assign numeric value to a variable
var `.=` *s*	*s*	Append string to a variable
var `+=` *n*	*n*	Add a number to a variable
var `-=` *n*	*n*	Subtract a number from a variable
var `*=` *n*	*n*	Multiply a variable by a number
var `/=` *n*	*n*	Divide a variable by a number
var `%=` *n*	*n*	Assign modulo with a number to variable
var `&=` *n*	*n*	Perform logical AND with a variable
var `|=` *n*	*n*	Perform logical OR with a variable
var `++`	*n*	Post-increment variable
var `--`	*n*	Post-decrement variable
`++` *var*	*n*	Pre-increment variable
`--` *var*	*n*	Pre-decrement variable

Database Access

Database rows (cards) can be accessed by providing an index in brackets. Without brackets, the current card (`this`) is assumed. Database columns are named. The name must always be prefixed with an underscore (_). In place of the name, the field can be selected with a column number (which must also be prefixed with an underscore), beginning at 0. Only fields that store data in the database can be accessed (types *Input*, *Time*, *Flag*, and *Choice*); this excludes fields of type *Label* and *Print*.

The `avg`, `dev`, `min`, `max`, and `sum` operators differ from all other operators: they don't reference a field in the current or any single card, they operate on a field in all cards by accessing an entire column of the database. These operators are also available as `qavg`, `qdev`, `qmin`, `qmax`, and `qsum`, which apply the calculation only to the result of the last query (i.e., to the cards displayed in the summary). Finally, the `savg`, `sdev`, `smin`, `smax`, and `qsum` variations are applied to all cards in the current section; if there are no sections or if all sections are selected, all cards are considered.

The `switch` statement is legal only in *Action when pressed* expressions for Button-type fields in the form editor. It does nothing except as action for a button in a card. It switches xmbase-grok to a new form as if the Database pulldown had been used (see the Editing Forms chapter for details about the difference between *databases* and *forms*. The first argument is the new form name, the second argument is the query expression or search string that determines which cards are displayed in the summary initially. The possible combinations are:

`search("", "")`
: Does nothing.

`search("", "*")`
: Keep the current form, and put all cards in the summary.

`search("", "{expr}")`
: Keep the current form, and put all cards in the summary for which *expr* returns something other than 0 or the empty string.

`search("", "(expr)")`
: Equivalent to the previous, except that the returned string is converted to a number, which is checked for nonzero values.

`search("", "string")`
: Keep the current form, and put all cards in the summary whose searchable fields contain *string*.

`search("name", "")`
: Switch to form *name*, and display all cards in the summary.

`search("name", "xxx")`
: Switch to form *name*, and then perform a query. `xxx` stands for any of the above query expressions.

Because short-circuiting doesn't work, `switch` can't depend on a conditional, but its two arguments can. `switch` returns the empty string, which means that the button won't execute a command as usual; if this is overridden by appending a semicolon and another string expression, the command is executed after the database switch. To execute a script before switching, prepend a `system` statement and a semicolon to the `switch` statement (the switch is done after the expression is completely evaluated). To switch back to the previous form, use the `prevform` statement.

Operator	*Type*	*Operation*
_field	*s* ,*n*	A field from the database, of current card
_field `[` *n* `]`	*s* ,*n*	A field from the database, from any card
`this`	*n*	The number of the current card, 0 is first
`last`	*n*	The number of the last card, 0 is first
`avg` (*_field*)	*n*	Average of a field in all cards
`qavg` (*_field*)	*n*	Average the current query result
`savg` (*_field*)	*n*	Average the current section
`dev` (*_field*)	*n*	Standard deviation of a field in all cards
`qdev` (*_field*)	*n*	Std. dev. of the current query result
`sdev` (*_field*)	*n*	Std. dev. of the current section
`min` (*_field*)	*n*	Minimum value of a field in all cards
`qmin` (*_field*)	*n*	Minimum of the current query result
`smin` (*_field*)	*n*	Minimum of the current section
`max` (*_field*)	*n*	Maximum value of a field in all cards
`qmax` (*_field*)	*n*	Maximum value of the current query result
`smax` (*_field*)	*n*	Maximum value of the current section
`sum` (*_field*)	*n*	Sum of a field in all cards
`qsum` (*_field*)	*n*	Sum of the current query result
`ssum` (*_field*)	*n*	Sum of the current section
`dbase`	*s*	The name of the accessed database file
`form`	*s*	The name of the accessed form file
`section`	*s*	The name of the current section, or the empty string
`section`	*n*	The number of the section the current card is in, or 0
`section [` *n* `]`	*s*	The name of section *n*
`section [` *n* `]`	*n*	The number of the section card *n* is in, or 0
`prevform`	*s*	The name of the previous accessed form file
`switch (` *s* , *s* `)`	*s*	Database switch and/or query; see above

Operating System Access

Operator	*Type*	*Operation*
`system ( `*s*` )`	*s*	Execute a shell command and return the result as a string
`$ `*envvar*	*s*	Return the value of the environment variable *envvar*
`host`	*s*	The host name of the local host
`user`	*s*	The user's login name
`uid`	*n*	The user's numeric user ID
`gid`	*n*	The user's numeric group ID
`access ( `*s* `,` *n*` )`	*n*	1 if the file name exists (if the number is 0), or if it can be accessed for execution (1), writing (2), and/or reading (4). See access(3).
`beep`	*s*	Ring the terminal bell, return a null string
`error ( `*args*` )`	*s*	Format a string like printf, print it in a window, return a null string

Time Conversion

Dates and times are stored as number of seconds since January 1, 1970. Durations are stored as number of seconds. Note that this means thata time is a significantly larger number than a duration, even if both have the same hh:mm string representation. The representation depends on the date and time format selected in the Preferences menu.

Operator	*Type*	*Operation*
`time`	*s*	The current time as hh:mm or hh:mm[ap] string
`time ( `*n*` )`	*s*	Extract time part of the number, and format as hh:mm or hh:mm[ap] string
`date`	*s*	Today's date as dd.mm.yy or mm/dd/yy string
`date ( `*n*` )`	*s*	Extract date part of the number, and format as dd.mm.yy or mm/dd/yy string
`duration ( `*n*` )`	*s*	Convert a number of seconds to a hh:mm string
`date`	*n*	Current time in seconds since January 1, 1970
`year ( `*n*` )`	*n*	Extract the (four-digit) year from a time
`month ( `*n*` )`	*n*	Extract the month 1..12 from a time
`day ( `*n*` )`	*n*	Extract the day 1..31 from a time
`hour ( `*n*` )`	*n*	Extract the hour 0..23 from a time
`minute ( `*n*` )`	*n*	Extract the minute 0..59 from a time
`second ( `*n*` )`	*n*	Extract the second 0..59 from a time
`julian ( `*n*` )`	*n*	Extract the julian date 0..365 from a time
`leap ( `*n*` )`	*n*	1 if the time is in a leap year, or 0 otherwise

Files and Programs

This is a complete list of all files and directories required for xmbase-grok:

file	*location*	*contents*
`grok`	GBIN	main executable
`grok.hlp`	GLIB	ascii help texts
`Manual.ps`	GLIB	this PostScript manual
`grokdir`	GLIB	demo application directory
`.grok`	~	default form and database directory
`.grokrc`	`~/.grok`	configuration data for xmbase-grok
`grok.xpm`	LIBDIR	X pixmap icon, optional
`Grok.icon`	`~/.icons`	full-color icon, SGI systems only

"GBIN" stands for the directory specified in the `GBIN` macro in the Imakefile or Makefile.alt, by default /usr/local/bin. "GLIB" is also in the Imakefile or Makefile.alt, the default is /usr/local/lib. "LIBDIR" is provided by imake and depends on the system.

Every xmbase-grok application consists of one "form" file and one or more "database" files. The form file describes the card layout and the data interpretation; they have the extension `.gf`. The name of the form file without the extension is put into the Database pulldown in the main window. The default location for form files is the `~/.grok` directory.

The form references data files. Normally, they have the same name as the form file but with the extension `.db`, and are also stored in the `~/.grok` directory. If the resulting path references a directory instead of a file, the contents of this directory is searched. Every file in this directory also ends in `.db` and becomes a section that is put into the Section pulldown. If the directory contains subdirectories, they are searched for more `.db` files recursively.

xmbase-grok searches for form files in four locations, in the following order:

1. the current directory,
2. ./grokdir,
3. ~/.grok,
4. GLIB/grokdir

All the form files ending with `.gf` found in these directories are put into the Database pulldown. Form files from different directories are separated by an etched line. xmbase-grok attempts to recognize and eliminate duplicate form files found over two different paths, but puts duplicate form files that it thinks are in different directories into the pulldown multiple times. This can be disabled with the preferences menu.

Contents