Commands

Note: use the << and >> browse buttons to view the list of commands, or press the Help button in the Insert Command dialog in *Idealist*.

Commands can be given to *Idealist* in one of several ways:

1. Via the menu (the usual and obvious method);
2. By placing commands inside buttons in the button bar;
3. By placing commands inside buttons embedded in the text of a record;
4. By sending commands from another application using DDE;
5. By placing commands in a text file called IDEALIST.CMD;
6. By placing commands in a text file called *database*.CMD;
7. By placing the commands in a text file, and using the FileCommand() command;
8. By typing Alt+F10 and entering the commands directly.

Each command has a name (for example "FileOpen") and an optional list of parameters that may follow the command. For example, passing the command "FileOpen" to *Idealist* is identical to selecting File, Open from the menu, and will produce a dialog box from which you can select a database to open. Alternatively, the command can be given the name of a database directly, for example FileOpen("sample"). This bypasses the dialog box and searches for a database called "sample" in the current directory. Parameters are given after the name of the command, enclosed in parentheses and separated by commas or semi-colons (or whatever list separator is specified in the International section of Control Panel).

String parameters

Where a parameter is a string, as in the example above, it should be enclosed in quotes (double, single or those in the extended character set). Escape sequences are permitted in character strings, so be careful when typing in filenames that contain backslashes!

Getting a parameter from the Clipboard

A parameter can be fetched from the Clipboard by using "{@Clipboard}" as one or more of the arguments. For example:

```
Find("{@Clipboard}")
```

...would search the current database using the text on the Clipboard as a search expression.

Using the current selection as a parameter

The current selection (from the current field) can be used as the parameter to a command by using "{@Selection}" as one or more of the arguments. For example:

```
Widen("{@Selection}")
```

...would widen the hit list using the current selection as a search expression.

Using the contents of a field as a parameter

The contents of another field in the current record can be used as the parameter to a command, by enclosing the name of the field in braces. For example, the following command moves the insertion point into the second Notes field and then inserts the contents of the first Notes field:

```
FieldGoto("Notes[2]")
Insert("{Notes[1]}")
```

Other special parameters

The following can be used in place of a parameter:

```
{@Clipboard}
{@Database}
{@Date}
{@FieldName}
{@FieldType}
{@LongDate}
{@Name}
{@Position}
{@Selection}
{@Time}
{@Type}
```

For example, the following moves the insertion point to the first Notes field and then inserts the current date, in the long date format:

```
FieldGoto("Notes[1]")
Insert("{@LongDate}")
```

Command names

Several commands have synonyms, for example, FilePrint and Print are the same command.

Multiple commands

Multiple commands in a button or DDE command string should be separated by tabs, blanks or new lines, for example:

```
FileOpen("c:\\idea\\people")
ViewStack Find("!Home")
```

Autodial(*Number,Prefix,Flags,Type,Port*)

Invokes the automatic telephone dialler; see the section on automatic telephone dialing for more details. AutoDial is assigned to CTRL+F11.

If AutoDial() is used without any parameters, the current word/number in the current field is selected and the autodial dialog is displayed.

If AutoDial() is used with one or more parameters, the number/options are set from the parameters and autodialing starts immediately, without displaying the dialog.

For example:

```
AutoDial("206206","9",1,"T","1")
AutoDial("791738","9",1)
AutoDial("202376")
AutoDial()
```

Flags are:

```
1        Use prefix
```

Type is:

```
"T"      Tone dial
"P"      Pulse dial
```

Port is:

```
"1"      Comm port 1
"2"      Comm port 2
```

AutoOpen

Displays the dialog box that lets you edit either IDEALIST.CMD (if no database is open, or the open database is iconized) or *database*.CMD.
The commands in IDEALIST.CMD are run when *Idealist* starts.
The commands in *database*.CMD are run when *database* is opened.

Useful AutoOpen .CMD file commands include:

- Find()
- GetUserName()
- MoveWindow()
- ShowOver()

Backtrack(*Shift*)
A synonym for ViewBacktrack().

Cd(*path*)

Changes the current directory to *path* (which may be relative or fully qualified).

This command can be useful after the user has used one of the file open dialog boxes, and you need to return the current directory to a known location.

This is similar to the DOS 'cd' command, except that if the new directory name includes a drive letter, the default drive is also changed.

CdApplication()

Changes the current drive and directory to the *Idealist* application directory (listed in the Help, About, More dialog).

This command can be useful after the user has used one of the file open dialog boxes, and you need to return the current directory to a known location.

CdDatabase()

Changes the current drive and directory to that holding the current database.

This command can be useful after the user has used one of the file open dialog boxes, and you need to return the current directory to a known location. For example, the following could be used to backup the current database to a Natural format file in the same directory as the database:

```
CdDatabase() Export(3,0,0,"","notes.txt",0,3)
```

CharLeft(*Chars, Select*)

Moves the text cursor to the left.

This command can take two parameters; the first is the number of characters to move the text cursor by (defaulting to 1), and the second indicates whether or not the selection should be extended while the cursor is moved (defaulting to 0, which means not). For example, the following moves the text cursor left by five characters:

```
CharLeft(5)
```

The following does the same, but extends the selection over the characters as it does so:

```
CharLeft(5, 1)
```

CharRight(*Chars, Select*)

Moves the text cursor to the right.

This command can take two parameters; the first is the number of characters to move the text cursor by (defaulting to 1), and the second indicates whether or not the selection should be extended while the cursor is moved (defaulting to 0, which means not). For example, the following moves the text cursor right by five characters:

```
CharRight(5)
```

The following does the same, but extends the selection over the characters as it does so:

```
CharRight(5, 1)
```

ClearFlag(*Flag*, *Mask*)

See ToggleFlag() for details of *Flag* and *Mask*.

For example, the following command can be used to turn the status bar off:

```
ClearFlag("View", 8)
```

Comfort()

Saves all unsaved information in *Idealist.* This includes unsaved edits to the current record, altered record and field definitions, altered synonyms, altered glossary entries, altered button bar commands and all options.

This command takes no parameters.

See also Save()

CrossRef(*Search*)
CrossReference(*Search*)
A synonym for ViewCrossRef().

DdeExecute(*Service*, *Topic*, *Commands*)

Establishes a DDE link with Service/Topic, sends Commands to the link and then terminates the link. For example:

```
DdeExecute("Progman", "Progman", "[AddItem(Hello.exe)]")
```

Idealist cannot send DDE commands to itself.

Dedupe()
Produces the record deduplication dialog.

DeleteMenu(*Command*)

Removes *command* from the *Idealist* menu. *Command* is the name of the command, not the text that appears in the menu. For example:

```
DeleteMenu("SearchExclude")
DeleteMenu("EditInsertRecno")
DeleteMenu("AutoOpen")
```

Drop()
Removes the current record from the hit list. This does not delete the record.

This command takes no parameters.

EditBold()

Changes the currently-selected text in a rich text field to bold. This has no effect on the text in other field types.

This command takes no parameters.

EditChangeCase()

Alternates the case of the current selection between all lower case, all upper case, and initial caps based on the first two characters of the selection.

This command takes no parameters.

EditClear(*Count*)

Deletes the selection without changing the contents of the Clipboard. If the selection is an insertion point, it deletes the character to the right of the insertion point. If *Count* is supplied, it deletes the specified number of characters from the right of the insertion point. If *Count* is a negative number, it deletes to the left of the insertion point.

EditCopy()

Copies the current selection onto the Clipboard.

EditCut()

Copies the selection onto the Clipboard and then deletes it from the record.

EditDeleteLine()
Deletes the current line; the same as pressing CTRL+Y.

EditFind(*String*, *Options*)

Searches for *String* within the text of the current record.

Options is formed by adding together the following possible options:

```
1    Search is case sensitive
2    Start from beginning of record
4    Stop searching at the end of the current field
8    (unused)
16   Search string is a word
32   (unused)
64   (unused)
128  (internal use)
256  Stop at the end of the hit list
512  Start at the beginning of the current field
```

For example, the following searches forward from the beginning of the current record for the word "Frank":

```
EditFind("frank", 18)
```

If either parameter is missing, both are prompted for by dialog box.

EditFindNextHit()

Moves the text cursor to the end of the next hit word in the current record.

This command takes no parameters.

EditGlossary(*Name*, *Text*)

If both *Name* and *Text* are given, this command modifies the glossary list. *Name* must be a legal *Idealist* name, *i.e.* consist of up to 31 alphanumeric characters. *Text* can be any combination of characters.

If *Name* does not exist in the glossary list, a new entry is defined.
If *Name* already exists in the glossary list, its text is replaced with Text. If Text is an empty string, then *Name* is deleted. For example:

```
EditGlossary("i", "Idealist")
EditGlossary("BSL", "Blackwell Science Ltd")
EditGlossary("addr", "4 The Square\r\nRadlett\r\nHerts")
EditGlossary("BSP", "")
```

If no parameters are supplied, this command displays the glossary dialog box.

EditIncrement(*n*)

Takes the current selection, or the word at the cursor, and attempts to increment (*i.e.* add one to) it. The following gives examples of the effect of this command:

```
0          ...becomes...    1
0.1        ...becomes...    1.1
2143       ...becomes...    2144
31/10/93   ...becomes...    1/11/93
Friday     ...becomes...    Saturday
June       ...becomes...    July
```

If a number is given as a parameter, then that number is added to the current word, instead of 1. The number may be negative, in which case the word is decremented.

EditInsert()

Inserts an object into the text of the current record at the text cursor. The type of object depends on the type of field.

See also

- EditInsertButton
- EditInsertCommand
- EditInsertDate
- EditInsertFunction
- EditInsertGraphic
- EditInsertPickList
- EditInsertRecno
- EditInsertTime
- EditInsertVocab

EditInsertButton()

Inserts a command button into the text of the current record at the text cursor.

For this command to work, the following conditions must be met:

1. The current database must be in Source mode;
2. The current field must not be read-only;
3. The field must be a regular text, wrapped text or line of text field.
4. The field must not be a rich text field.

EditInsertCommand()

Produces a dialog box containing a list of all the commands that can be inserted into a button. Selecting a command pastes that function into the text of the current field.

EditInsertDate(*Shift*)

Replaces the current selection with the date. The current field need not be a date field.
If *Shift* is missing or zero, the date is formatted in the Windows short date format. If *Shift* is non-zero, and the current field is not a date field, the date is formatted in the Windows long date format.

EditInsertFunction()

Produces a dialog box containing a list of all the functions that can be inserted into a Calculated field. Selecting a function pastes that function, and placeholders for its arguments, into the text of the current field.

EditInsertGraphic(*Filename*)

Inserts the filename of a graphic file into a Graphic field. If *Filename* is not given it will be prompted for.

For this command to work, the following conditions must be met:

1. The current database must be in Source mode;
2. The current field must not be read-only;

EditInsertPickList()

Produces the Pick Item dialog box, enabling you to insert one item from the current field's pick list.

For this command to work, the following conditions must be met:

1. The current field must not be read-only;
2. The field must be defined to have a pick list.

EditInsertRecno()

Replaces the current selection with the internal record number of the current record.

EditInsertTime()

Replaces the current selection with the current system time, formatted according to the short date format in the International section of Control Panel.

EditInsertVocab()

Produces the pick vocabulary item(s) dialog box, enabling you to insert one or more items from the current field's vocabulary into the field.

For this command to work, the following conditions must be met:

1. The current field must not be read-only;
2. The field must be defined to have a vocabulary.

EditItalic()

Changes the currently-selected text in a rich text field to italic. This has no effect on the text in other field types.

This command takes no parameters.

EditPaste()

Replaces the current selection with text from the Clipboard.

EditRepeatFind()

Repeats the last edit find/replace command.

EditReplace(String1, String2, Options)

Searches for *String1* within the text of the current record and, if found, replaces it with *String2.*

Options is formed by adding together the following possible options:

```
1    Search is case sensitive
2    Start from beginning of record
4    Stop searching at the end of the current field
8    (unused)
16   Search string is a word
32   (unused)
64   (unused)
128  (internal use)
256  Stop at the end of the hit list
512  Start at the beginning of the current field
```

For example, the following searches forward from the beginning of the current record for the word "Frank", and replaces it with an empty string:

```
EditReplace("frank", "", 18)
```

If any of the parameters are missing, all are prompted for by a dialog box.

EditSelectAll()

Selects all the text in the current field.

EditSpell(*Shift*)

Checks the spelling of one of three quantities of text within the current record:

1. If *Shift* is zero or missing, and there is text selected in the current field, then the selected text is checked.
2. If *Shift* is zero or missing, and there is no text selected in the current field then all the text in the current field is checked.
3. If *Shift* is non-zero, then all the text in all the fields in the current record is checked.

Text that is read-only will not be checked.

Note that *Idealist* sends indexable words to the spelling checker (*i.e.* sequences of indexable characters), rather than just words (*i.e.* sequences of alphanumeric characters). Therefore, the words that will be sent to the spelling checker depend on the indexable characters you have defined for the database.

EditSub()

Changes the currently-selected text in a rich text field to sub-script. This has no effect on the text in other field types.

This command takes no parameters.

EditSuper()

Changes the currently-selected text in a rich text field to super-script. This has no effect on the text in other field types.

This command takes no parameters.

EditUnderline()

Underlines the currently-selected text in a rich text field. This has no effect on the text in other field types.

This command takes no parameters.

EditUndo()

Undoes changes to the current record by reloading it from the database.

EndOfField(*Select*)
Moves the insertion point to the end of the field. If *Select* is a non-zero value, the selection is extended.

EndOfLine(*Select*)
Moves the insertion point to the end of the current line. If *Select* is a non-zero value, the selection is extended.

EndOfRecord()

Moves the insertion point to the end of the current record.

Exclude(*Search*)
A synonym for SearchExclude.

ExcludeAll(*Search*)
A synonym for SearchExcludeAll.

ExcludeRecno(*List*)

Excludes records with the listed record handles from the hit list. For example:

```
ExcludeRecno("12 25 2143 99")
```

Export()

A synonym for FileExport.

FieldAppend(*Field*)
Appends a field of type *Field* to the current record. If *Field* is not given as a parameter, it will be prompted for.

FieldChangeType(*Field*)

Destroys the current field, and replaces it with one of type *Field.* The text in the field is preserved. If *Field* is not given as a parameter, it will be prompted for.

This command is also useful if you change the type of a field FROM its existing type TO its existing type (for example, change the type of a Notes field to a Notes field). This causes the field in the database to conform to the data type listed for it in the Field, Define dialog.

See the FieldChangeTypeAll() command for a more powerful variant of this command.

FieldChangeTypeAll()

Changes the type of each field in the current record to match that currently given in the field definition. Why would anyone want to do that?

For an example, suppose you have set a database up with a PartNumber field defined to be an Integer field (part numbers are numbers, right?). You type in a few records. Then you come to type in a part number like "123-X7". Oops. Part numbers can contain alphabetic characters, too. You pull up the Field, Define dialog, select the PartNumber field, and change the data type from "Integer" to "Line of Text". You type in the remaining records.

Everything ok now? No quite. The part number fields you typed in first are sitting in the database, indexed and formatted as integers, blissfully unaware that you think they are now bits of text. In order to process them as bits of text, you will have to change them to bits of text. You can do this by finding all the records you typed in first, then using the **FieldChangeTypeAll()** command on them.

This command takes no parameters.

FieldDefine()
Produces the Define Field Types dialog box.

FieldDelete(*Field*)

Deletes *Field* from the current record, if *Field* is not read-only. For example:

```
FieldDelete("Text[3]")
```

If *Field* is not given, this command deletes the current field.

FieldDown(*Num*)
Moves the insertion point down *Num* fields within the current record.

FieldGoto(*Field*)

Moves the selection to *Field* within the current record. For example:

```
FieldGoto("Text[3]")
```

If *Field* is not given as a parameter, it will be prompted for.

FieldInsert(*Field*)

Inserts a field of type *Field* into the current record 'above' the current field. If *Field* is not given, it will be prompted for.

FieldJoin()

Concatenates the current field and the next field in the current record. This command takes no parameters.

FieldSplit()

Creates a new field, and inserts it into the current record after the current field. The text before the selection stays in the original field; the text after the selection is moved to the new field.

This command takes no parameters.

FieldUp(*Num*)
Moves the insertion point up *Num* fields within the current record.

File1()

Attempts to open the database listed under item 1 in the file menu.

File2()
Attempts to open the database listed under item 2 in the file menu.

File3()
Attempts to open the database listed under item 3 in the file menu.

File4()

Attempts to open the database listed under item 4 in the file menu.

FileClose(*Shift*)

If *Shift* is missing or zero, this command closes the current database. If *Shift* is non-zero, all open databases are closed.

FileCommand(*Filename*)
Causes the list of commands in the text file *Filename* to be executed.

FileDelete(*Database*)

Attempts to delete *Database* from the disk. If *Database* is not given as a parameter, it will be prompted for. If *Database* is open, it will not be deleted.

For example, if you have a database called SAMPLE, then this command will delete the following files:

SAMPLE.TEX
SAMPLE.ROT
SAMPLE.OCC
SAMPLE.TRM
SAMPLE.DIR
SAMPLE.NET (if present)

The FileDelete command will **not** delete the following files:

SAMPLE.DEF	(field and record definitions)
SAMPLE.CMD	(autoopen commands)
SAMPLE.BAR	(button bar)

FileExport(*What*, *From*, *To*, *Format*, *Output*, *Flags, Method*)
Export(*What*, *From*, *To*, *Format*, *Output*, *Flags, Method*)

Copies records from the current database to a text file.

What must be one of:

```
1    Current record
2    Hit list
3    All
4    Range
```

If *What* is 4, then *From* and *To* must be set to the first and last records in the hit list to be exported.

Format should be the name of an export format file; if this is given as an empty string ("") then the records will be exported in Natural format.

Output is the name of the text file the records are to be exported to. If this is given as an empty string (""), then the records will be exported as text to the clipboard.

Flags should be formed by adding the following:

```
2    Convert ANSI to OEM (ASCII)
4    Don't collate
8    Suppress blank lines when exporting formatted
128  Convert ANSI to Macintosh
256  reserved for internal use
```

Method is optional. If it is not given, the exporting method will be Formatted if a format file is given, or Natural if no format file is given. If *Method* is specified, it should be one of the following:

```
0    Formatted
1    Comma separated
2    Tab separated
3    Natural
```

For example:

```
FileExport(1, 0, 0, "", "TEST.TXT", 0)
```

exports the current record to TEST.TXT in Natural format.

Beware! Backslashes in filenames must be given twice, *e.g.* C:\\WIN\\IDEALIST\\TEXT.TXT.

FileImport(*Method*, *Source*, *Format*, *RecordType*, *IgnoreCharacters, Flags*)
Import(*Method*, *Source*, *Format*, *RecordType*, *IgnoreCharacters, Flags*)

Copies records from an external textual source into *Idealist.*

Method must be one of:

```
1    Natural
2    Idealist database
3    Inflected (tagged) text file
4    Non-inflected (separated) text file
5    DBF file
6    Files as records
7    Comma separated fields
8    Tab separated fields
9    Simple
```

Source must be the filename of the file to be imported from. It can have been compressed using the Microsoft COMPRESS utility. If this is given as an empty string (""), then the records will be imported from the clipboard.

If *Method* is 3 or 4, then *Format* must be the name of an import format file.

If *Method* is 5, 6, 7, 8 or 9, then *RecordType* must be the name of the record type into which records will be imported.

If *Method* is 3 or 4, then *IgnoreCharacters* must be a string of up to 10 characters which will be stripped from the input file when it is imported.

Flags should be formed by adding the following:

```
1    Strip leading blanks
2    Convert characters from DOS to Windows
4    Minimize while importing
8    Ignore blank fields
16   Strip trailing blanks
32   Ignore single line breaks
64   Ignore auto create fields
128  Convert characters from Macintosh to Windows
256  reserved for internal use
```

For example:

```
FileImport(3, "MEDLINE.TXT", "MEDLINE.IMP", "", "|", 0)
```

Imports inflected records from MEDLINE.TXT using MEDLINE.IMP, ignoring any '|' characters in the input.

Beware! Backslashes in filenames must be given twice, *e.g.* C:\\WIN\\IDEALIST\\MEDLINE.TXT.

FileInfo()
Produces the File, Info dialog.

FileNew(*Database*)
Creates a new database, if one of that name does not already exist.

FileOpen(*Database, Flags*)

Opens a database. If *Database* is already open, then it is made the current database.

If *Flags* is missing or zero, then the database is opened in "single user" mode, giving the current user exclusive write access to the database.
If *Flags* is set to 1, then the database is opened in read-only mode.
If *Flags* is set to 2, then the database is opened in multi-user (shared) mode.

FilePrint(*What*, *From*, *To*, *Format*)
Print(*What*, *From*, *To*, *Format*)

Copies record(s) from the current database to an attached printer, optionally formatting them.

What must be one of:

```
1    Current record
2    Hit list
3    All
4    Range
```

If *What* is 4, then *From* and *To* must be set to the first and last records in the hit list to be printed.

Format should be the name of a print format file; if this is given as an empty string ("") then the records will be printed in standard format.

FileReindex()

Rebuilds the index of the current database.

This does **not** have to be done every time records are added to the database, since indexing is automatic. It only needs to be done if the index has become corrupt or you wish to change the stopword list.

This command is not available when the database is open in multi-user mode.

FileSaveCopyAs(*Filename*)

Saves a copy of the current database in *Filename*.*, and leaves you editing the original database. For example, if you are working on a database called SAMPLE, and use this command to save a copy of the SAMPLE database under the name BACKUP, then the following files will be produced:

```
BACKUP.TEX
BACKUP.ROT
BACKUP.OCC
BACKUP.TRM
BACKUP.DIR
BACKUP.BAR (if SAMPLE.BAR exists)
BACKUP.DEF (if SAMPLE.DEF exists)
```

If you do not supply *Filename*, a name will be prompted for.

This command is not available when the database is open in multi-user mode.

FileSecurity()
Produces the security dialog box.

Find(*Search*)

Searches for records in the current database that contain the search expression *Search,* and creates a new hit list of those records.

For example:

```
Find("(Gudmundsdottir | Björk, Gudmundsdottir) & Debut")
```

If more than one parameter is given, the parameters are concatenated before being fed into the search engine. For example:

```
Find("ISBN==", "{@Selection}")
Find("(Text)=", "{@Clipboard}", " [or] ", "{@Clipboard}")
```

This command, and all the other search commands, can take up to ten parameters. Each parameter is individually evaluated and then joined to the other parameters. For example, the following can be used to search for the contents of the clipboard in any text field: `Find("(Text)=", "{@Clipboard}").`

FindAll(*Search*)

Searches for records in all open databases that contain the search expression *Search,* and creates new hit lists of those records.

For example:

```
FindAll("Gudmundsdottir [or] Björk, Gudmundsdottir")
```

If more than one parameter is given, the parameters are concatenated before being fed into the search engine. For example:

```
FindAll("ISBN==", "{@Selection}")
FindAll("(Text)=", "{@Clipboard}", " [or] ", "{@Clipboard}")
```

FindAllRecords()

A synonym for SearchFindAllRecords.

FindRecno(*List*)

Produces a hit list containing records that have in internal record numbers in *List*. For example:

```
FindRecno("2143 0 777")
```

FirstField()

Makes the first field in a record the current field. The same as pressing CTRL+PgUp.

This command takes no parameters.

GetGlossary(*Item*)
Replaces the current selection with glossary item *Item.* If *Item* is not given, the current selection is used.

GetUserName()

Prompts for a user name, which replaces the user name *Idealist* obtained when it started up.

HardSpace()

Inserts a non-breaking space character (ANSI 160) into the current field.

HelpAbout()
Produces information about *Idealist*.

HelpTopics()
Produces the help table of contents (loaded from a small text file called IDEALIST.CNT).

HideImage()

Closes the image viewer window (opened by using ViewImage()).

HideOver()

Closes the overview window, opened with the ShowOver() command.

HitListCommand(*Command*)

Executes the command string *Command* on all the records in the hit list from the current record to the end of the hit list. For example, to change the word "angel" into the word "bloke" in all the records in the hit list, go to the start of the hit list and put the following command into a button bar button:

```
HitListCommand("EditReplace(\"angel\", \"bloke\", 10)")
```

Three caveats:
1. Do not use the HitListCommand() command in the command string;
2. Do not use the ViewNext or Next commands in your command string, since moving to the next record is done automatically;
2. When using quote marks inside the command string, prefix them with backslashes (as in the example above) so that they don't get confused with the quotes surrounding the string itself.

Use this command with extreme caution!

HitListDelete()

Deletes all the records in the current hit list from the database.

This command takes no parameters.

HitListOptions()
Produces the Hit List Options dialog.

HitListPop(*Shift*)

Discards the current hit list and replaces it with one taken from the top of the hit list 'stack'.

If *Shift* is non-zero, the current hit list is replaced by the hit list at the bottom of the hit list stack.

HitListPush()

Pushes a copy of the current hit list onto the hit list 'stack'. This enables you to return to this hit list by using the HitListPop command.

IfFieldEmpty(Field, TrueCommands, FalseCommands)

Advanced users only!

If *Field* is empty, then *TrueCommands* are carried out.
If *Field* is not empty, then *FalseCommands* are carried out.
If *Field* is not given (*i.e.* is an empty string), then the current field is tested.
If there is no record currently being displayed, or if the current record does not contain *Field*, then no commands are carried out.

For example:

```
IfFieldEmpty("Date[3]", "Insert(\"{Today}\")", "")
IfFieldEmpty("", "Insert(\"{Today\")", "")
```

See also: IfFieldPresent(), IfStatus().

IfFieldPresent(*Field*, *TrueCommands*, *FalseCommands*)

Advanced users only!

If the current record contains *Field*, then *TrueCommands* are carried out.
If the current record does not contain *Field*, then *FalseCommands* are carried out.
If there is no current record, then no commands are carried out.

For example:

```
IfFieldPresent("Text[3]", "", "FieldAppend(\"Text\")")
```

See also IfFieldEmpty(), IfStatus().

IfStatus(*Status*, *TrueCommands*, *FalseCommands*)

Advanced users only!

When *Idealist* is about to display its menu, it gets the status of itself and then uses that status to grey out menu items that are not currently available. For example, if there is no database currently open, then File, Close is greyed out.

The **IfStatus()** command allows one set of commands (*TrueCommands*) to be executed if *Idealist* is in a certain state, and an alternate set of commands (*FalseCommands*) if it is not in that state.

The *Status* argument to the **IfStatus()** command consists of a number, that is composed of a number of states ANDed together. In hexadecimal, the states are (the following is pasted directly out of *Idealist*'s C source code):

```
STATUS_DBOPEN        0x00000001 /* there is a database open */
STATUS_DBWRITE       0x00000002 /* database has write permission */
STATUS_HITLST        0x00000004 /* there is a hit list */
STATUS_RECORD        0x00000008 /* there is a record visible */
STATUC_UNUSED        0x00000010
STATUS_SEL           0x00000020 /* selection marked in current edit */
STATUS_CLIPBOARD     0x00000040 /* there is text in clipboard */
STATUS_HITSTACK      0x00000080 /* hit stack level > 0 */
STATUS_LAYOUT        0x00000100 /* view layout loaded */
STATUS_GRAPHICFIELD  0x00000200 /* current field is graphic */
STATUS_DIRTY         0x00000400 /* current record is dirty */
STATUS_RICH          0x00000800 /* current field is rich text */
STATUS_SOURCE        0x00001000 /* we are looking at source */
STATUS_MULTIFIELD    0x00002000 /* >1 field in the record */
STATUS_VOCABFIELD    0x00004000 /* current field has vocab */
STATUS_FIELDWRITE    0x00008000 /* current field has write permission */
STATUS_RECORDMARKED  0x00010000 /* current record is marked */
STATUS_UNUSED        0x00020000
STATUS_FOOBAR        0x00040000 /* The button bar is visible */
STATUS_MAXCHILD      0x00080000 /* There is a maximized child */
STATUS_AMANDA        0x00100000 /* current field is regular text */
STATUS_SPELL         0x00200000 /* Spelling checker available */
STATUS_CALCFIELD     0x00400000 /* current field is calculated */
STATUS_PICKLISTFIELD 0x00800000 /* Current field has a pick list */
STATUS_TEXTFIELD     0x01000000 /* Current field is just text */
STATUS_LAYOUTEDIT    0x02000000 /* layout can be edited */
STATUS_EMPTYFIELD    0x04000000 /* current field is empty */
STATUS_EXCLWRITE     0x08000000 /* database is open with exclusive write permission */
STATUS_RECORDWRITE   0x10000000 /* current record is editable */
STATUS_OVERVIEW      0x20000000 /* overview window is open */
```

The *Status* argument is best expressed in hexadecimal. This can be done C-style, by prefixing the number with '0x' or '0X'.

For example, the following will backtrack if there is a stacked hit list, otherwise it will do nothing:

```
IfStatus(0x80, "Backtrack", "")
```

The following will invoke the spelling checker if (a) the spelling checker is available and (b) we have write permission to the database, otherwise an overview of the hit list will be displayed:

```
IfStatus(0x02000002, "EditSpell","Overview(\"Text\")")
```

See also: IfFieldPresent(), IfFieldEmpty().

Import()

A synonym for FileImport.

Insert(*String*)

Replaces the current selection with *String,* just as if it had been typed from the keyboard. *String* may contain escape sequences. For example:

```
Insert("From:\tFielding, John\rTo:\tGold, Stanley")
```

LastField()

Makes the last field in a record the current field. The same as pressing CTRL+PGDN.

LayoutClose()

Closes the current layout, if one is open.

LayoutCreateType()
Creates a new layout type for the current record type.

LayoutDeleteItem()

Deletes the current layout item, if the current database is in Source mode.

LayoutDeleteType()

Removes the layout type for the current record type from the current layout file.

LayoutInsertItem(*Type*)

Inserts an item of *Type* into the current layout. Type must be one of:

```
70    Field item
84    Text item
66    Graphic item
67    Command button
128   Hollow frame
129   Filled rectangle
```

LayoutMoveItem()
Changes the cursor to a four pointed arrow, allowing the current layout item to be moved.

LayoutNew()
Creates a new layout, containing layout types for all the currently defined record types.

LayoutOpen(*Filename*)
Opens an existing layout, stored in a text file with the extension .LAY. If *Filename* is missing, it will be prompted for.

LayoutProps()

If the current database is in Source mode, this produces the properties dialog for the current layout item.

LayoutSave()

Saves any changes made to the currently open layout.

LayoutSaveAs(*Filename*)
Saves the current layout to *Filename*. If *Filename* is missing, it will be prompted for.

LayoutSnap(*Number*)

Sets the layout grid to *Number* pixels. If *Number* is not given, the layout grid is toggled on and off.

The size of the layout snap grid can also be set by holding down SHIFT while pressing the snap button in the layout floating button bar.

LineDown(*Repeat*, *Select*)

Moves the text cursor down by *Repeat* lines. If *Repeat* is omitted, 1 is assumed. If *Select* is non-zero, the selection is extended.

LineUp(*Repeat*, *Select*)
Moves the text cursor up by *Repeat* lines. If *Repeat* is omitted, 1 is assumed. If *Select* is non-zero, the selection is extended.

LogClear()
Clears all text from the log window.

LogOff()
Turns off logging, but does not close the log window.

LogOn()
Turns on logging, and opens a log window.

Maximize(*Flag*)

If *Flag* is non-zero, the main *Idealist* window is maximized.
If *Flag* is zero or missing, the current database window is maximized.

See also:
Minimize()

MCI(*Commands*)

Commands is a string of MCI commands, for example:

```
MCI("open c:\\win\\waves\\turner.wav alias turner
play turner from 0 wait
close turner")
```

The commands within the MCI command string must be separated by line breaks (as above) or by semi-colons.

Refer to the Microsoft Multimedia Programmer's Reference for details of MCI command strings.

MessageBox(*Text*, *Title*, *Style*)

This command displays a message box on screen.

Any pseudo-fields in *Text* are expanded. The available pseudo-fields are:

```
{@CheckSum}    A CRC checksum for the current record
{@Clipboard}   The text contents of the clipboard
{@Database}    The name of the current database
{@Date}        The date in short format
{@FieldName}   The name of the current field
{@Fields}      A list of fields in the current record
{@FieldType}   The type of the current field
{@Hits}        The number of records in the hit list
{@LastSearch}  The last search command
{@LongDate}    The date in long format
{@Name}        The name of the current record
{@Position}    Eg "1st of 720"
{@Recno}       The current record number
{@Selection}   The currently selected text
{@Time}        The current time
{@Type}        The type of the current record
```

Style may be given in decimal or hexadecimal, and is composed by adding one or more of the following together:

```
Buttons:
OK                          0x0000
OK Cancel                   0x0001
Abort Retry Ignore          0x0002
Yes No Cancel               0x0003
Yes No                      0x0004
Retry Cancel                0x0005

Icons:
Hand/Stop                   0x0010
Question                    0x0020
Exclamation                 0x0030
Asterisk/Information        0x0040

Default Button:
First                       0x0000
Second                      0x0100
Third                       0x0200

Style:
Application Modal           0x0000
System Modal                0x1000
Task Modal                  0x2000
```

Windows API programmers will recognise this command to be a direct entry to the Windows MessageBox(*hwnd*, *text*, *title*, *style*) function.

Minimize(*Flag*)

If *Flag* is non-zero, the main *Idealist* window is minimized.
If *Flag* is zero or missing, the current database window is minimized.

See also:
Maximize()

ModifyButton(*Button*)
A synonym for ViewModifyButton.

ModifyGlossary(*Name*)

ModifySynonyms()

Produces the Modify Synonyms dialog box. This will be either the Regular synonyms dialog, or the Wider synonyms dialog, depending on the current search options.

MoveWindow(*Database*, *x*, *y*, *Width*, *Height*)

Moves the database window to position *x*,*y* and resizes it so that it is *width* pixels wide and *height* pixels high. For example:

```
MoveWindow("c:\\idealist\\sample.tex", 0,0, 300, 200)
```

If *Database* is missing, then the main *Idealist* window is repositioned and resized. For example:

```
MoveWindow(0,0,640,480)
```

Narrow(*Search*)
A synonym for SearchNarrow.

NarrowAll(*Search*)
A synonym for SearchNarrowAll.

NarrowRecno(*List*)
Narrows the hit list so that it only contains records whose internal record numbers are in *List*.

Next(*Shift*)
Makes the next record in the hit list the current record.

If *Shift* is non-zero, then the last record in the hit list becomes the current record.

NextField()
Makes the next field the current field. The same as pressing CTRL+DOWN.

Overview()
Toggles the overview window on and off.

PageDown()
The same as pressing PgDn on the keyboard.

PageUp()
The same as pressing PgUp on the keyboard.

Prev(*Shift*)
Previous(*Shift*)
Makes the previous record in the hit list the current record.

If *Shift* is non-zero, then the first record in the hit list becomes the current record.

Print()

A synonym for FilePrint.

PrinterSetup(*device*, *driver*, *port*)

If used with no arguments, this command produces the standard Printer Setup dialog.

Otherwise, this command sets the current printer to that specified by *device*, *driver* and *port*.
For example, to use the PrinterSetup() command to make the fax machine my current printer, I would use the command:

```
PrinterSetup("Microsoft At Work Fax","EFAXDRV","FAX:")
```

Quiet(*Number*)

This command is used to alter an internal 'quiet level', which determines whether or not *Idealist* message boxes are displayed. The default behaviour ('quiet level' equal to zero) is that all messages boxes are displayed. When the 'quiet level' is raised above zero, message boxes are not displayed and a default value returned (this value can be set by the Reply command).

When the Quiet command is used with *Number* greater than zero, the internal 'quiet level' is increased by one.
When the Quiet command is used with *Number* less than zero, the internal 'quiet level' is decreased by one.
When the Quiet command is used with *Number* equal to zero, the internal 'quiet level' is reset to zero.

Unless explicitly set by this command, the quiet level stays at 0 under normal circumstances, and rises to 1 when processing a DDE command.

RecordChangeType(*Type*)

Changes the type of the current record to *Type*, then rearranges the fields in the current record so that they conform to the fields defined to be part of record *Type*. If *Type* is not given, it will be prompted for. Any fields that are in the record definition, but not in the record, are added to the record. Any fields that are in the record, but not in the record definition, are moved to the end of the record.

This command is very useful after a record definition has been changed; it can be used to modify existing records so that they conform to the new record definition.

See also: SetRecordType()

RecordCreate(*Type*)

Creates a record of type *Type.* For example:

```
RecordCreate("Default")
```

If *Type* is not given as a parameter, it will be prompted for.

RecordDefine()

Produces the Define Record Types dialog box.

RecordDelete()

Deletes the current record from the database. No confirmation is asked for. This command takes no parameters.

RecordDrop()
Drop()
Removes the current record from the hit list. This does not delete the record.

This command takes no parameters.

RecordDropMarked()

Removes all marked records from the hit list. This does not delete the records.

This command takes no parameters.

RecordDropUnmarked()

Removes all unmarked records from the hit list. This does not delete the records.

This command takes no parameters.

RecordMark(*Mark*)

If *Mark* is non-zero, sets the mark on the current record.
If *Mark* is zero, clears the mark on the current record.
If *Mark* is omitted, toggles the mark on the current record.

RecordName(*Name*)
Sets the name of the current record to *Name.* If *Name* is not given it will be prompted for.

RecordUndelete()

Produces the Undelete Record dialog box.

RecordUnmarkAll()

Unmarks all the marked records in the current database.

Repeat(*Count*)
Repeats the last command *Count* times. If omitted, *Count* defaults to 1. The Repeat() command cannot be repeated.

Reply(*Button*)

Idealist message boxes do not appear when *Idealist* is processing a string of commands and the Quiet level is non-zero. Instead, the message box returns OK immediately, before it has a chance to appear on screen.

The Reply() command lets you change the default response of message boxes when processing command strings. *Button* must be one of:

```
16        OK
32        Yes
64        Yes To All
128       Overwrite
256       Append
512       Retry
1024      No
2048      No To All
4096      Cancel
8192      Options
```

The exact consequences of overriding the default response should be studied with care before exposing the effects to users.

Remember to set the default back to OK! For example, the following technique is strongly recommended:

```
Quiet(1)
Reply(128)
FileExport(1, 0,0, "", "myfile.txt", 0)
Reply(16)
Quiet(0)
```

ReportHits(*Flag*)
See SearchListAllHits.

Restore(*Flag*)

If *Flag* is non-zero, the main *Idealist* window is restored.
If *Flag* is zero or missing, the current database window is restored.

Run(*Command*)

Executes *Command* as if it were typed into File/Program Manager's File, Run dialog box.

For example:

```
Run("WRITE.EXE")
Run("PBRUSH TIGERCUB.BMP")
Run("WILSON.DOC")
Run("DOSPRMPT.PIF /E:640 /C WP.BAT")
Run("\\Windows\\PifEdit DOSPrmpt.pif")
```

If *Command* is missing, nothing happens.

Save()
Explicitly saves and indexes the current record.

See also Comfort()

SearchDirect(*Command*)
A direct entry in the *Idealist* search engine.

SearchExclude(*Search*)
Exclude(*Search*)

Drops those records matched by the search expression *Search* from the hit list. For example:

```
SearchExclude("Terry Björk & Gloria Hunniford")
```

This command, and all the other search commands, can take up to ten parameters. Each parameter is individually evaluated and then joined to the other parameters. For example, the following can be used to search for the contents of the clipboard in any text field: `Find("(Text)=", "{@Clipboard}")`.

SearchExcludeAll(*Search*)
ExcludeAll(*Search*)

Drops those records matched by the search expression *Search* from the hit list in all open databases. For example:

```
SearchExcludeAll("Terry Björk & Gloria Hunniford")
```

This command, and all the other search commands, can take up to ten parameters. Each parameter is individually evaluated and then joined to the other parameters. For example, the following can be used to search for the contents of the clipboard in any text field: `Find("(Text)=", "{@Clipboard}").`

SearchFind(*Search*)
Find(*Search*)

Searches for records in the current database that contain the search expression *Search,* and creates a new hit list of those records.

For example:

```
SearchFind("Terry Björk [or] Björk, Terry")
```

If more than one parameter is given, the parameters are concatenated before being fed into the search engine. For example:

```
SearchFind("ISBN==", "{@Selection}")
SearchFind("ISBN==", "{@Clipboard}")
SearchFind("(Text)=", "{@Clipboard}", " [or] ", "@{Clipboard}")
```

This command, and all the other search commands, can take up to ten parameters. Each parameter is individually evaluated and then joined to the other parameters. For example, the following can be used to search for the contents of the clipboard in any text field: `Find("(Text)=", "{@Clipboard}").`

SearchFindAll(*Search*)
FindAll(*Search*)

Searches for records in all open databases that contain the search expression *Search,* and creates new hit lists of those records.

For example:

```
SearchFindAll("Terry Björk [or] Björk, Terry")
```

If more than one parameter is given, the parameters are concatenated before being fed into the search engine. For example:

```
SearchFindAll("ISBN==", "{@Selection}")
SearchFindAll("ISBN==", "{@Clipboard}")
SearchFindAll("(Text)=", "{@Clipboard}", " [or] ", "{@Clipboard}")
```

This command, and all the other search commands, can take up to ten parameters. Each parameter is individually evaluated and then joined to the other parameters. For example, the following can be used to search for the contents of the clipboard in any text field: `Find("(Text)=", "{@Clipboard}").`

SearchFindAllRecords()
FindAllRecords()
Rapidly produces a hit list containing all the records in the current database.

SearchListAllHits()

Displays a list box containing the currently hit words for the current database.

SearchNarrow(*Search*)
Narrow(*Search*)
Searches within the current hit list for records containing search expression *Search.*

For example:

```
SearchNarrow("Terry Björk [or] Pink Elephant")
```

This command, and all the other search commands, can take up to ten parameters. Each parameter is individually evaluated and then joined to the other parameters. For example, the following can be used to search for the contents of the clipboard in any text field: `Find("(Text)=", "{@Clipboard}")`.

SearchNarrowAll(*Search*)
NarrowAll(*Search*)
Searches within the current hit list of all open databases for records containing search expression *Search.*

For example:

```
SearchNarrowAll("Terry Björk [or] Pink Elephant")
```

This command, and all the other search commands, can take up to ten parameters. Each parameter is individually evaluated and then joined to the other parameters. For example, the following can be used to search for the contents of the clipboard in any text field: `Find("(Text)=", "{@Clipboard}")`.

SearchOptions()

Produces the Search Options dialog box.

This command takes no parameters.

SearchWiden(*Search*)
Widen(*Search*)

Extends the current hit list to include those records matched by search expression *Search.* For example:

```
SearchWiden("Orange [near] Soda")
```

This command, and all the other search commands, can take up to ten parameters. Each parameter is individually evaluated and then joined to the other parameters. For example, the following can be used to search for the contents of the clipboard in any text field: `Find("(Text)=", "{@Clipboard}").`

SearchWidenAll(*Search*)
WidenAll(*Search*)

Extends the current hit list in all open databases to include those records matched by search expression *Search.* For example:

```
SearchWidenAll("Orange [near] Soda")
```

This command, and all the other search commands, can take up to ten parameters. Each parameter is individually evaluated and then joined to the other parameters. For example, the following can be used to search for the contents of the clipboard in any text field: `Find("(Text)=", "{@Clipboard}").`

Security()
Produces the File, Security dialog.

SetFlag(*Flag*, *Mask*)

See ToggleFlag() for details of *Flag* and *Mask*.

For example, the following command can be used to turn the status bar on:

```
SetFlag("view", 0x0008)
```

SetImageZoom(*n*,*n*,*n*,*n*,*n*,*n*)

Sets the zoom levels for the ViewImage pop-up window. Each zoom level is given as a percentage of the image's real pixel size. The default zoom levels are:

```
SetImageZoom(25,50,75,100,200)
```

SetRecordType(*Type*)
Sets the type of the current record to *Type*. If *Type* is not given it will be prompted for.

See also: RecordChangeType()

SetTitle(*Title*)

Sets the title of the main *Idealist* window to *Title*. For example:

```
SetTitle("Pearl Data Service")
```

ShowOver()
Displays the overview window.

Sort(*FieldList, Options, WordList*)
A synonym for ViewSort.

Spell()

The same command as EditSpell(). Invokes the spelling checker.

SpellDatabase()

Produces a hit list containing every record in the database that contains an indexed word that isn't in the spell checker dictionary.

SpellOptions()
Invokes the spelling checker options dialog.

Stack()
A synonym for ViewStack.

StartOfField(*Select*)
Moves the insertion point to the start of the current field. If *Select* is a non-zero value, the selection is extended.

StartOfLine(*Select*)

Moves the insertion point to the start of the current line. If *Select* is a non-zero value, the selection is extended.

StartOfRecord()
Moves the insertion point to the start of the current record.

Sulk(*Filename, Buttons, Flags*)

Loads and displays a Search-U-Like dialog from the description in the text file *Filename*. If *Filename* is not given it will be prompted for.

Buttons, if given, is a number that describes which buttons should be displayed in the dialog. *Buttons* is made by adding together one or more of the following:

```
find       0x00000001
widen      0x00000002
narrow     0x00000004
exclude....0x00000008
index......0x00000010
save.......0x00000020
recall.....0x00000040
recalllast.0x00000080
cancel.....0x00000100
help.......0x00000200
```

By default, all buttons are displayed (thus, the default value for *Buttons* is 0x000003FF).
To turn off the widen, narrow and exclude buttons, use 0x000003F1.

Flags is a number made by adding together one or more of the following:

1 OR fields together (the default is AND the fields together)
2 Use private help file. The name of the help file is taken from *Filename* + the extension ".HLP". The contents page of this help file, if found, is displayed.

A Search-U-Like dialog is a customised search dialog. It contains a series of edit controls that can be filled in by the user, each edit control corresponding to one particular field.

The title of the dialog, and the type and placement of the edit controls, are loaded from a small text file.

```
"Full Search Dialog", 30,30,320,182
"Text"              (Text)   6,  8, 60,8,  70,  6, 100,12
"&Author(s):"       ATH      6, 24, 60,8,  70, 22, 150,12
"Article &Title:"   ATI      6, 40, 60,8,  70, 38, 150,12
"&Journal:"         JTI      6, 56, 60,8,  70, 54, 150,12
"ISS&N:"            ISN      6, 72, 60,8,  70, 70, 100,12
"&Year:"            YPB      6, 88, 60,8,  70, 86,  50,12
"&Volume:"          VON      6,104, 60,8,  70,102,  50,12
"Iss&ue:"           IUN      6,120, 60,8,  70,118,  50,12
"Pa&ges:"           PGN      6,136, 60,8,  70,134,  50,12
"&Page Range:"      PGR      6,152, 60,8,  70,150,  50,12
"Supp&lement:"      SPN      6,168, 60,8,  70,166,  50,12
```

Any line starting with a semi-colon or hash character is ignored, and can be used to store comments.

The first line contains the title of the dialog, followed by four numbers; the x- and y- coordinates of the upper left corner of the dialog (this coordinate is relative to the upper left corner of the main *Idealist* window) and the width and height of the dialog.

Thereafter, each line describes one 'field' within the dialog. The lines contain:

1. The title, or prompt, of the field;
2. Either the name of the field (as defined in the Field, Define dialog) or a type of index, enclosed in

parentheses (for example (text), which would search all text fields);
3 & 4. The coordinates of the title, within the dialog;
5 & 6. The width and height of the rectangle the title appears within;
7 & 8. The coordinates of the edit control;
9 & 10. The width and height of the edit control.

All measurements are in Windows dialog units. One unit in the x-axis is equal to a quarter of the width of an average system font character. One unit in the y-axis is equal to one eigth of the height of an average system font character.

When displayed, the dialog will contain seven buttons, namely Find, Widen, Narrow, Exclude, Index (displays a pop-up index browser for the current field), Cancel and Help.

Text entered into the edit controls is turned into a search command and then sent to the search engine. If text is entered for more than one field, the sub-commands are 'ANDed' together. For example, the following filled in dialog:

```
Text:     [Terry Wogan        ]
Date:     [>29/04/62          ]
Notes:    [Ire*               ]
```

produces the search command:

```
Text=Terry Wogan & Date>29/04/62 & Notes=Ire*
```

ToggleFlag(*Flag*, *Mask*)

Flag can be one of:

```
Flag         Equivalent registry entry

"View"       ViewFlags=
"EditFind"   EditFindFlags=
"Export"     ExportFlags=
"File"       FileFlags=
"Import"     ImportFlags=
"Log"        LogFlags=
"Print"      PrintFlags=
"Record"     RecordFlags=
"Search"     SearchFlags=
"Sort"       SortFlags=
"Spell"      SpellFlags=
```

The value of *Mask* can be found in the section describing the specific flag in the Registry topic in this help file. The value of *Mask* can be given as a decimal or hex number (prefixed with a C-style '0x'), and several *Mask*s can be ORed together.

For example, the following command can be used to toggle the main menu on and off:

```
ToggleFlag("view", 0x1000)
```

UserCommand()

Displays a small dialog that lets you pick one or more commands and then run them. The same as pressing ALT+F10.

ViewBacktrack(*Shift*)
Backtrack(*Shift*)
Returns from a cross-reference, or restores the most recently stacked hit list.

If *Shift* is non-zero, the original hit list is restored and **all** stacked hit lists are discarded.

ViewCrossRef(*Search*)
ViewCrossReference(*Search*)
CrossRef(*Search*)
CrossReference(*Search*)

Stacks the current hit list, and creates another one consisting of all those records that contain the search expression given as the parameter or, if none is given, the search expression formed by extracting the current selection from the current record or, if there is no selection, the current word from the current record. For example:

```
ViewCrossRef("Terry Björk")
ViewCrossReference
```

ViewDelete()

Deletes all the records in the current hit list from the database.

This command takes no parameters.

ViewFieldComment()

Displays commentary information for the current field. The keyboard shortcut for this command is CTRL+F1.

Field comments can be specified for each field type by pressing the "Comment" button in the Define Field Types dialog.

ViewGoto(*Number*)

Makes the record at location *Number* in the hit list the current record. If *Number* is not given as a parameter, it is prompted for.

For example, the following would make the 500th record in the hit list the current record:

```
ViewGoto(500)
```

ViewImage(*Field*)

Searches *Field* for the filename of a bitmapped graphic image and displays that image in a pop-up window. If *Field* is not given, then the contents of the current field is used. For example:

```
ViewImage()
ViewImage("Text")
ViewImage("Picture[3]")
```

The window remains active until it is closed. This means that you can leave the window open, go to another record containing *Field* and the image in the new field will be displayed.

The window can be closed either from its system menu, or by using the HideImage() command.

The pop-up window is designed to display multi-page fax images. The window has its own button bar which allows the image to be zoomed, rotated, inverted and printed. There are also previous/next page buttons for use when the image contains more than one page (for example, a multi-page fax image stored in a TIFF or DCX file).

The filename can be followed by a number of parameters that help set the initial state of the pop-up window. For example:

```
c:\images\fax32.dcx invert
d:\fax78.dcx page=7
d:\fax79.tif rotate=180 invert zoom=5
d:\fax80.dcx x=200 y=290
```

Invert inverts the color of the image, for example from black-on-white to white-on-black.
Page sets the initial page from the image to be displayed.
Rotate sets the initial angle of rotation of the image. It can be one of 0, 90, 180 or 270.
x and **y** set the size, in pixels, of the image.
Zoom sets the initial zoom level from 1 to 5, inclusive. The default zoom levels are 1=25%, 2=50%, 3=75%, 4=100%, 5=200%. These levels can be changed using the SetImageZoom() command.

ViewMode(*Mode*)

If *Mode* is non-zero, the view mode is set to Source.
If *Mode* is zero, the view mode is set to Result.
If *Mode* is omitted, the view mode is toggled between Source and Result.

ViewModifyButton(*Button*)
ModifyButton(*Button*)

Produces the Modify Button Bar dialog for button number *Button* in the button bar. Buttons are numbered from the left, starting with button 0. If *Button* is omitted, the button to modify is chosen by clicking on it.

ViewNext(*Shift*)
Next(*Shift*)
Makes the next record in the hit list the current record.

If *Shift* is non-zero, then the last record in the hit list becomes the current record.

ViewOptions()

Displays the View, Options dialog. This command takes no parameters.

ViewOverview()
Overview()
Toggles display of the hit list overview on and off.

ViewPrev(*Shift*)
ViewPrevious(*Shift*)
Prev(*Shift*)
Previous(*Shift*)
Makes the previous record in the hit list the current record.

If *Shift* is non-zero, then the first record in the hit list becomes the current record.

ViewSort(*FieldList, Options, WordList*)
Sort(*Field, Options, WordList*)

Rearranges the records in the current hit list so that they appear in alphanumeric order according to the contents of *Field. Field* can either be the name of a field, with an optional field instance in [brackets]) or one of:

```
(First)   The first field in each record.
(Type)    The record type, e.g. Default.
(Name)    The record name, e.g. Home.
```

Options can be a combination of the following:

```
1    Sort in descending order
2    Ignore blanks
4    Ignore punctuation
8    Ignore certain words if they occur at the start of the field
16   Sort is case sensitive
32   Sort on last word in field
```

WordList can be a short list of words, separated by blanks or punctuation. The list defaults to "a an the".

If you want to sort a hit list on more than one field, you can do so by repeatedly sorting the hit list from the least to the most important field. For example, if you want to sort a hit list by Name, Date and Title, you would first sort by Title, then by Date and finally by Name. *Idealist* will recognise that you are doing a multi-level sort and behave accordingly.

Examples:

```
ViewSort("Text", 0, "")
ViewSort("Text[2]", 0, "")
ViewSort("(First)", 6, "")
ViewSort(Text, 10, "a the")
```

ViewStack()

Stack()

Stacks the current hit list. This command takes no parameters.

Widen(*Search*)
See SearchWiden.

WidenAll(*Search*)
See SearchWidenAll.

WidenRecno(*List*)

Widens the hit list so that it contains all the records in the database with the internal record numbers in *List*.

WindowCascade()

Arranges the open database windows.

This command takes no parameters.

WindowCloseAll()

Closes all open databases.

This command takes no parameters.

WindowTile(*Shift*)

Arranges the open database windows.

If *Shift* is missing or zero, the windows are arranged vertically (they become tall). If *Shift* is non-zero, the windows are arranged horizontally (they become wide).

WindowTileHorz()

Arranges the open database windows horizontally (so they become wide).

WindowTileVert()

Arranges the open database windows vertically (so they become tall).

WordCount(*n*)

If *n* is missing or zero, this command displays the number of characters, words and lines in the current field.

If *n* is non-zero, this command displays the number of characters, words lines and fields in the current record.

WordLeft(*Repeat, Select*)
Moves the text cursor left by *Repeat* words. If *Select* is non-zero, the selection is extended.

WordRight(*Repeat*, *Select*)
Moves the text cursor right by *Repeat* words. If *Select* is non-zero, the selection is extended.

What is *Idealist* and how does it work?

Idealist is a text database manager. "Database", because it stores information in fields and records. "Text", because it does not store data in a table (like flat-file or relational database managers), but instead keeps it as a long string of text. This fundamental difference enables *Idealist* to do things that other databases cannot.

Firstly, fields can vary in size so that they are individually as long or short as the information they contain and are not a predefined, fixed length.

Secondly, it lets you store a variety of different types of record together in the same file.

Whilst this makes an *Idealist* database flexible and very easy to understand, it does not explain how you can find anything anywhere in the database instantly.

There is no magic to it: *Idealist* records the location of literally everything as it goes into the database.

This "full-text" index then lets you retrieve records by whatever might be in any of the fields, and the time taken to complete a search for any item in any or every field - regardless of the size of the database - remains constant: less than a fraction of a second.

Next

Variable length fields and records

Fields are, by default, individually variable in size up to a maximum of about 40 pages of closely typed text each. You can also predefine them to use a restricted vocabulary or pick list, or to accept only certain classes of character at predetermined positions; and of course fields can be defined for handling non-textual information.

Variable-length fields mean that you do not store empty space, so although the index to an *Idealist* database is typically larger than that of an equivalent tabular database, the data file itself will be more compact.

Next

Multiple record types in a single file

In one database file you can employ an unlimited variety of different types of record, using a different set of fields in each type of record (to a maximum of 160 fields per record). This flexibility in the structure of the database enables you to store all your information in one place. Inside records, fields can be repeated and actually added or deleted "on-the-fly" - *Idealist* will show you the real meaning of flexibility in database structure!

Next

Dynamic full-text index

As records are added or edited the words, numbers, dates, times *&c* that they contain are immediately and automatically added to a list of all the words in the database and their locations recorded. This is not like a key or alphanumeric sort on one or two fields of a tabular database. It is more like the index in a book, except that this index contains a structured list of all the words in the book. Because the index is updated immediately, as soon as information has gone into the database it is available for searching.

The index can be controlled at three levels: you can add unusual characters that you want to the index, *e.g.* , the trademark symbol; you can exclude redundant words in a "stopword" list (the most usual are already provided); and you can turn indexing off in particular fields.

In *Idealist* the index stores the location of words by record and by field. This means that not only simple searches like 'government & London', are done in a fraction of a second, but also field searches, like 'city=London & sector=government'. By storing information in a structured form, and by using this field structure in its index, *Idealist* is able to retrieve information with far greater accuracy than full-text indexing software that does not offer a database structure.

Next

Searching

Press one function key or select an item with the mouse and up comes the search dialog box: either browse the index (divided into text, numbers, dates, times, record types and record names) or type your search straight in. *Idealist* shows you the first record found; the status bar tells you how many records were found, and the word or words you were looking for are highlighted. You can refine the search by narrowing the "hit list" of found records by specifying further words that must also occur in the records, or you can widen the list or exclude records from it using other words. You can also sort and overview your hit list, and combine search expressions with AND, OR and NOT.

Next

Networking

Enabling a group of individuals with their different objectives, different information and different abilities to share their information requires precisely the sort of databases that *Idealist* offers: flexible in content, completely searchable and easy to comprehend.

The classic text databases are in fact the huge databases found online. *Idealist* gives your organization its own version of this: a number of people keeping records together. where they can be rapidly and easily found by their colleagues, and with the option of making the information available to an even wider group. Such a workgroup has little excuse for the left hand not knowing what the right hand is doing. On a network, *Idealist* is available for server, peer-to-peer, and client/server configurations. Network features are not limited to specific network systems, and any that runs properly with Windows will let *Idealist* offer groups of people simultaneous access to a shared and constantly changing information resource.

Idealist is not only quick to find information, but also extraordinarily economical of network traffic: it takes just one look in the index for each search element and then a single access to the data file to retrieve each record as you move to it in the hit list.

Autodial dialog

If you have a Hayes or Hayes-compatible modem attached between your computer and a telephone line, *Idealist* can automatically dial any telephone number stored in a record.

Number

The number to be dialled, copied from the current cursor position in the current record. The number can contain blanks and hyphens, which are ignored, or # and *, which are not.

Prefix

This usually contains the dialling prefix needed to obtain an outside line on your telephone system. The number 9 is a common prefix.

Use Prefix

Toggle use of the above dialling prefix.

Dial Type

Select the dialling method used by your telephone system.

Modem Port

Select the serial port your modem is attached to. The port should be configured using Control Panel.

Choose record type dialog

Presents a list of the record types currently defined. Select one from the list, and press OK.

Search hits dialog

Lists all the words found by the current search.

Modify synonyms dialog

The dialog box that appears when Regular Synonyms are selected in Search, Options.

Trigger

The search words that act as a trigger for each list of synonyms. The trigger word should be the word you are most likely to search for.

Expansion

The list of search words that are searched for (in addition to the trigger word) whenever the trigger word is searched for. Phrases must be placed in double quotes, *e.g.* "British Broadcasting Corporation".

Add

Adds a new trigger word.

Delete

Deletes the currently selected trigger and its list of synonyms.

Modify wide synonyms dialog

The dialog box that appears when Wider Synonyms are selected in Search, Options.

Any time that any textual word or phrase in the synonym list is searched for, all the other words and phrases on the same line are also searched for. Phrases must be placed in double quotes, *e.g.* "British Broadcasting Corporation".

Add synonyms by clicking on the required line and typing the extra synonym(s). To add a new list of synonyms, move to the end of the last line, press ENTER and type the new line.

Search options dialog

Put cursor on first hit word

Causes the text cursor to be moved to the first hit word in a record, whenever that record is displayed.

Command line search dialog

By default, *Idealist* presents a search dialog which enables the casual user to perform simple searches. However, for users with some knowledge of the *Idealist* search command syntax, the Command Line Search Dialog may be preferable. This enables you to enter search commands using the *Idealist* search command syntax. The reward is access to more powerful searches than can be performed using the alternate dialog.

Automatic Right Truncation

Causes the search engine to treat single words (*e.g.* 'red') as if they were right-truncated (*e.g.* 'red*'). This has no effect on phrase searching.

Synonyms

Off Synonyms are not substituted for words being searched for.
Regular Given a synonym defined as 'color red green blue', a search for 'color' will also trigger searches for 'red', 'green' and 'blue'. A search for 'blue' will only search for 'blue'.
Wider Given a synonym defined as 'color red green blue', a search for any one of the synonyms will also trigger searches for all the other synonyms.

Export records to file dialog

See the section on exporting for general help.

What

Current Record Only the current record from the current database is exported.
Hit List All the records in the current hit list are exported, in the order in which they appear in the hit list.
All Records in the database are exported, in whatever order they are physically stored in.
Hit List Range The records between the given positions (inclusive) in the hit list are exported, in the order in which they appear in the hit list.

Method

Formatted The fields from each record are arranged according the information in an export format file.
Comma Separated The fields from each record are exported, separated by commas (ANSI 44). The records are separated by line breaks. Try not to export fields containing line breaks!
Tab Separated The fields from each record are exported, separated by tabs (ANSI 9). The records are separated by line breaks. Try not to export fields containing line breaks!
Idealist Natural The records are formatted in the *Idealist* Natural format (portable, flexible and robust - used internally by Blackwell when creating backups).

To File

Specifies the file to which the records are exported. If no file is given, *Idealist* attempts to export the records to the Clipboard.

Format

File The export format file to be used with the Formatted method.
Edit Invokes the export format editor to modify the specified export format file.

Options

Copies The number of copies of each record to be exported. This defaults to one.
Collate Copies If off, copies are exported in the order 111222333. If on, copies are exported in the order 123123123.
Suppress blank lines When exporting using an export format file, any empty lines or lines containing only spaces and tabs will not be exported.
Character Conversion Converts the characters in the exported text from one character set to another. Text in an *Idealist* database is stored in the Windows (ANSI) character set.

Choose field dialog

Presents a list of the field types currently defined, from which you may choose one.

Choose field dialog

This dialog lets you choose the arrangement of fields in the overview window.

Fields lists the defined fields available for display, including (First) (the first field in each record), (Type) (the type of the record) and (Name) (the name of the record).

Insert inserts the field you select from the list box into the overview at the position of the field header button you pressed to produce this dialog.

Append adds the field you select from the list box to the overview.

Replace replaces the field header button you pressed to produce this dialog with the field you select from the list box.

Delete deletes the field header button you pressed to produce this dialog.

The overview window can be turned on and off by choosing Overview from the View menu.

Glossary dialog

Use this to create a glossary entry out of selected text, or to insert a specified glossary entry into the current field. The command is unavailable unless you select text in your document or have existing glossary entries.

Glossary name

Type a name of a new glossary entry with up to 31 alphanumeric characters, or select an existing glossary entry you want to insert, delete, or redefine.

Selection/glossary name

Displays either the selection in the document or the text of a selected glossary entry.

Insert

Inserts the contents of the selected glossary entry at the insertion point.

Define

Defines or redefines a glossary entry. This button is unavailable if you choose Edit Glossary without first selecting text.

Delete

Deletes the selected glossary entry.

Print dialog

What

Current Record Only the current record from the current database is printed.
Hit List All the records in the current hit list are printed, in the order in which they appear in the hit list.
All Records in the database are printed, in whatever order they are physically stored in.
Hit List Range The records between the given positions (inclusive) in the hit list are printed, in the order in which they appear in the hit list.

Setup

Press this button to specify which printer will be used, and the setup options for that printer.

Format

File The print format file to be used. If none is specified, the records are printed in a way that is similar to their appearance on screen.
Edit Invokes the print format editor to modify the specified print format file.

Sort dialog

Rearranges the order of the records in the current hit list. Note that this does not change the physical order of the records in the database.

If you want to sort a hit list on more than one field, you can do so by repeatedly sorting the hit list from the least to the most important field. For example, if you want to sort a hit list by Name, Date and Title, you would first sort by Title, then by Date and finally by Name. *Idealist* will recognise that you are doing a multi-level sort and behave accordingly.

Field

Choose the field to use as the sort key.

Order

Ascending Records are sorted in the order 123ABC.
Descending Records are sorted in the order CBA321.

Last Word Only

Only the last word in each field is used as the sort key. This is very useful for fields containing names in the form "Mr John Smith", when you want to sort the field on the surname. Note that enabling this option overrides the following "ignore" options.

Ignore

Blanks Any whitespace (blanks, tabs, line breaks) in the sort key are ignored.
Punctuation Any punctuation in the sort key is ignored. Punctuation characters are any characters except letters, digits and whitespace.
First Word(s) The specified list of words are ignored when sorting, if any of those words occur at the start of the sort key.

Undelete record dialog

Records that have been deleted or edited may be undeleted.

Record

A list of the first line from the first field in each undeleted record that is available for undeleting.

Pick vocabulary item(s) dialog

This dialog presents a list of valid words and phrases which may be entered into the current field. Note that the list box allows multiple selection of items.

The words and phrases are stored in a text file called *fieldname*.VCB, which can be edited with Notepad.

Modify button dialog

You can use this dialog to edit the contents of the button bar on either the main *Idealist* window, or a button bar on a database window.

Button

The design of button that is displayed on the button bar.

Commands

A list of *Idealist* commands that are carried out when the button is pressed.

Description

A description of the button's function that appears as a tooltip and in the status bar when the cursor lingers over the button.

Insert Command

Produces a list of available commands.

Delete

Deletes the selected button from the button bar.

Export format editor, format margins dialog

Specifies the margin sizes in Export Format files.

Left

The left margin in characters. Blanks will be inserted into the output to create this margin.

Text Width

The maximum width of the text in characters. Line breaks are inserted between words in the output stream to prevent any line from being greater than this width.

Print format editor, format page dialog

Rows

The number of rows of records to place on the page

Columns

The number of columns of records to place on the page

LeftHeader
Header
RightHeader

Parts of the header for each page, placed in the left, center and right of the top margin. The following may be embedded in the header:

```
{@Database}    The name of the current database
{@Date}        The current date
{@Page}        The page number
{@Time}        The current time
```

LeftFooter
Footer
RightFooter

Parts of the footer for each page, placed in the left, center and right of the bottom margin. The following may be embedded in the footer:

```
{@Database}    The name of the current database
{@Date}        The current date
{@Page}        The page number
{@Time}        The current time
```

Margins

Left Margin
Top Margin
Bottom Margin

Font

The default font to be used when printing text; if no font is specified, then the fonts defined in Field, Define Types are used.

Printable Area

Most printers cannot print right to the edge of a piece of paper. So, for example, although an A4 sheet measures 297 by 210mm, the printer can only print to an area 285 by 197mm.

Print format editor, format record dialog

Width

The maximum width into which text from this record will be printed. Text outside this width is clipped.

Height

The maximum height into which text from this record will be printed. Text outside this height is clipped.

Right Gap

The distance between the right of this record and the left of the next record in this row.

Bottom Gap

The distance between the bottom of this record and the top of the next record in this column.

Border

Selecting this option draws a solid black border around the record.

Print format editor, format element dialog

Element

Either literal text (*e.g.* "Hello, Windows") or a field expression (*e.g.* {Text[3]}) or a pseudo-field (*e.g.* {@Database}).

Position

X

The distance from the left of the record box to the element.

Y

The distance from the top of the record box to the element.

Size

Width

The width of the element. The text of the element is clipped to fit within this width.

Height

The height of the element. The text of the element is clipped to fit within this height.

Alignment

Specifies whether the text of the element is left, center or right-aligned within it's box.

Border

Selecting this option draws a solid black border around the element.

Font...

Changes the font, font size and style of the text of the element.

Search-O-Matic dialog

A dialog that is used to assist in building simple search commands. An alternative command-line search dialog is available that permits a wider choice of searches, such as proximity and range searches.

The topmost drop-down list contains all the defined fields, as well as separate entries for the data types of fields.

The contents of the middle drop-down list varies according to the data type of the field selected in the topmost drop-down list.

The edit box allows entry of a search word, partial search word or phrase - which depends on the type of search you wish to perform.

The index-browser list to the right displays the section of the index containing possible words from the currently-selected field. Some words may be grayed out, which means that they are in the index, but not in the specific field being searched.

Find All Records creates a hit list containing all the records in the current database, arranged in the physical order in which they appear in the database.

Layout button item properties dialog

Position

Specifies the x and y co-ordinates of the item, measured in pixels.

Size

Specifies the width and height of the item, measured in pixels.

Z-Order

Not relevant to this type of item.

Title

Specifies a brief title which appears on the button.

Commands

Specifies a list of *Idealist* commands that are carried out when the button is pressed.

Font

Specifies the font, font size and style used to display the button.

Border

Selecting this option draws a shaded border around the item.

Layout field item properties dialog

Position

Specifies the x and y co-ordinates of the item, measured in pixels.

Size

Specifies the width and height of the item, measured in pixels.

Bring To Front

Makes this field item the 'first' in the record.

Send To Back

Makes this field item the 'last' in the record.

Field

Specifies which field from the underlying record this item displays.

Border

Selecting this option draws a shaded border around the field item.

Layout graphic item properties dialog

Position

Specifies the x and y co-ordinates of the item, measured in pixels.

Size

Specifies the width and height of the item, measured in pixels. If you enter 0 for both the width and the height, the actual size of the bitmap will be used.

Bring To Front

Draws this graphic over the top of other text and graphic items.

Send To Back

Draws this graphic underneath other text and graphic items.

Graphic

Specifies the filename of the graphic image this item displays.

Border

Selecting this option draws a shaded border around the graphic item.

Layout shape item properties dialog

Position

Specifies the x and y co-ordinates of the item, measured in pixels.

Size

Specifies the width and height of the item, measured in pixels.

Bring To Front

Causes this shape to be drawn over the top of other shapes and text items.

Send To Back

Causes this shape to be drawn underneath other shapes and text items.

Color

Specifies the color used to display the shape.

Border

Selecting this option draws a shaded border around the item.

Layout text item properties dialog

Position
Specifies the x and y co-ordinates of the item, measured in pixels.

Size
Specifies the width and height of the item, measured in pixels.

Bring To Front
Makes this text item lie above others in the record, so that this text will be drawn over the top of overlapping text and graphic items.

Send To Back
Makes this text item lie below others in the record, so that this text will be drawn underneath overlapping text and graphic items.

Text
Specifies the string of characters to be displayed

Alignment
Specifies the alignment of the string of characters with the item's box.

Border
Selecting this option draws a shaded border around the text item.

Font
Specifies the font, font size and style used to display the text.

Choose name format dialog

Select the format to use when reformatting lists of names in fields with the Name data type. The name formats are stored in text file called NAME.LST. See the Name Formatting topic for more details.

Spelling dialog

The main spell check dialog box.

Ignore

Skips over the current, unknown, word (for this occurrence only).

Ignore All

Ignores any further occurrences of the current, unknown, word. The list of these ignored words is reset every time *Idealist* is started.

Change

Replaces the current, unknown, word with the suggested alternative

Add

Adds the current, unknown, word to the personal dictionary, stored in the text file PERSONAL.DIC. This can be viewed and edited using Windows Notepad. Words in this dictionary prefixed with '-' are negative words; their presence in the text will be reported as an error.

Suggest

Suggests alternative spellings for the current, unknown, word.

Options

Configure the spelling checker. The exact options will depend on your current language, current dictionary and version of the spelling checker.

Spelling options dialog

Deduplicate dialog

Idealist can assist in the detection and deletion of duplicate records in a hit list.

Imagine that you have a hit list of eight records, each of which contains only a single letter. Arranged side-by-side, the records look like this:

```
ABCADAEB
```

In this case, the redundant duplicates are those records that are repeats of previous records. *Idealist* can either automatically delete the redundant duplicates (those marked with an asterisk):

```
ABCADAEB
** *
```

..leaving...

```
ABCDE
```

...or it can mark all the duplicate records (not just the redundant duplicates), for you to identify and delete manually:

```
ABCADAEB
** * * *
```

The deduplicate dialog represents the first step in a three-stage process:

1. Select the criteria *Idealist* should use when judging records to be duplicates. The default is that all the fields in each record are examined. This can be refined by selecting one or more fields to be compared from each record.

2. *Idealist* examines all the records in the hit list. This can take some time, as each record must be found, loaded and processed.

3. *Idealist* reports the number of records that have duplicates. At this stage, *Idealist* can either:
(a) automatically delete the redundant duplicates, or (b) leave the suspect records marked for you to inspect and delete manually (Hint: the first thing you could do is drop all the unmarked records from the hit list).

Hit list options dialog

Wrap at Either End When on, moving past the end of the hit list wraps you back to the start of the hit list. When off, you will be warned when you try to move past the end of the hit list.

Create a Hit List of All Records When Opening When *Idealist* opens a database, it will create a hit list for that database, containing all the records in the database. Deselecting this option prevents this, leaving the database with no hit list.

Index browser dialog

The index-browser displays the section of the index containing possible words from the currently selected field. Some words may be grayed out, which means that they are in the index for the field's data type, but not in the specific field being searched.

Pick field contents from a list dialog

This dialog lists valid words and phrases which may be entered into the current field. Only one item from the list can be stored in the field.

The words and phrases are stored in a text file called *fieldname*.PCK, which can be edited with Notepad.

Import records from file dialog

See the section on importing for general help.

Method

Idealist Natural The input file is a text file containing *Idealist* fields and records arranged in a particular format, referred to as Natural.
Idealist Database The input file is an *Idealist* for Windows, *Idealist* for DOS or *Idealist* for Macintosh database.
Inflected The input file is assumed to contain a regular arrangement of fields and records tagged by text patterns specified in an import format file.
Non-inflected The input file is assumed to contain a regular arrangement of fields and records delimited by text patterns specified in an import format file.
DBF File Records are imported from a .DBF (dBASE) file into the specified record type. Each field in the DBF file is imported into a field of the same name in the *Idealist* record. For further details see Importing.
Files As Records One or more files are read, each file producing one imported record. The contents of the file are read into the first field in each record. A record type needs to be specified.
Comma Separated The input file is assumed to contain one record per line, with fields separated by comma (ANSI 44) characters. A record type needs to be specified.
Tab Separated The input file is assumed to contain one record per line, with fields separated by tab (ANSI 9) characters. A record type needs to be specified.
Simple The input file is split into single records of the selected type. All the text is directed into the first field in the record. Every blank line in the input file causes a new record to be created. This method is useful for large text files with undefined or unexplored contents.

Format

File The import format file to be used with the Inflected and Non-inflected methods.
Edit Runs Notepad to edit the selected import format file.

Record Type

The type of record (defined via Record, Define Types) to import records into when using the DBF File, Files As Records, Comma and Tab Separated methods.

From File

Specifies the file(s) from which records are to imported. *Idealist* currently reads from plain text, dBASE, Windows Write and Microsoft Word for Windows 2 files. The filename may contain wildcards.

Options

Strip Leading Blanks Removes any space or tab characters at the start of each imported line.
Strip Trailing Blanks Removes any space or tab characters from the end of each imported line.
Ignore Blank Fields Deletes blank fields from records after they are imported.
Ignore Auto Create Fields Do not update the automatic fields Created, Creator and CreatedTime in the records being imported.
Minimize During Import Reduces the main *Idealist* window to an icon while importing, and restores it afterwards.
Ignore Characters The specified characters will be removed from the text of fields when importing.
Character Conversion Converts the characters in the imported text from one character set to another as that text is imported. Text in an *Idealist* database is stored in the Windows (ANSI) character set.

Save/recall search dialog

This dialog is used to save and recall the contents of a Search-U-Like dialog, and associate a name with each saved search.

Search Name If saving a search, enter a name for the search of up to 31 characters.
Delete Deletes the currently-selected saved search.

Edit find dialog

Searches within a record or group of records for a given string of characters.

Search for

The text to search for. The text may include escape characters. The maximum length of the search text is 66 characters.

Options

Start from the beginning of the record moves the cursor to the start of the first field in the current record before each search.
Start from the beginning of the current field moves the cursor to the start of the current field before each search.
Start from the cursor position does not move the cursor before each search.
Continue to the end of the current field stops the search when the cursor reaches the end of the current field.
Continue to the end of the current record continues the search across fields until the end of the last field in the current record is reached.
Continue to the end of the hit list continues the search across fields and records until the end of the last field in the last record in the hit list is reached.

Match case Select this option to make sure that the search matches only text that is capitalized in exactly the same way as the search text.
Whole word Select this option if you want to make sure that the search matches with the text only when it is not part of a longer word.

Edit replace dialog

Searches within a record or group of records for a given string of characters and replaces them with another string of characters.

Search for

The text to search for. The text may include escape characters. The maximum length of the search text is 66 characters.

Replace with

The text that is to replace an occurrence of the search text. The text may include escape characters. The maximum length of the replace text is 66 characters.

Options

Start from the beginning of the record moves the cursor to the start of the first field in the current record before each search.
Start from the beginning of the current field moves the cursor to the start of the current field before each search.
Start from the cursor position does not move the cursor before each search.
Continue to the end of the current field stops the search when the cursor reaches the end of the current field.
Continue to the end of the current record continues the search across fields until the end of the last field in the current record is reached.
Continue to the end of the hit list continues the search across fields and records until the end of the last field in the last record in the hit list is reached.

Match case Select this option to make sure that the search matches only text that is capitalized in exactly the same way as the search text.
Whole word Select this option if you want to make sure that the search matches with the text only when it is not part of a longer word.

View field comment dialog

File options dialog

Create local field and record definitions

By default, field and record definitions are kept in a text file called IDEALIST.DEF and are accessible from all newly-created databases. However, you can create a separate set of field and record definitions to be used exclusively for each newly-created database. These are stored in the text file *database*.DEF. If you select this option and then create a new database, *database*.DEF is created by making a copy of the global field and record definitions file.

Create local button bar

When selected, this creates an new button bar for each database that is subsequently created. This works by creating a small text file called *database*.BAR, in which the button bar information is stored.

Stopword list...

The name of a text file which contains a list of word which *Idealist* is NOT to index. The contents of this file are copied into the database itself whenever a database is created.

Character value table...

The name of a file that will be used to replace the built-in character value table when a database is created. The character value table tells *Idealist* which characters to index, and what order to index them in.

File info dialog

This dialog gives information about the current database.

Indexable characters A list of the characters that *Idealist* considers to be part of an indexable word. This list is copied from an internal table when the database is created, and cannot be altered thereafter. See the section on Character Value Tables for more details.

Stopwords A list of the words which *Idealist* will not index. The list is copied from the text file STOPWORD.LST when the database is created.

Format version The format version of the database. This is not the same as the version of the software. Blackwell increase the database format version when the format of the database files is changed to accommodate new functionality. *Idealist* will not open a database when the database format version is newer than the software version. For example, the Oct94 EXE will not open a Jan95 database.

Codepage The codepage on which this database was created, and with which it is designed to be used.

Records A count of the number of records in the database.

Open mode The sharing mode of the current database. Can be one of read-only, multi-user or single-user.

Command line search dialog

Into this dialog, you type search commands in *Idealist*'s native search command syntax, for example 'Name=Terry Wogan' or '(Text)==New York' or 'PostedDate=21/03/93 [to] 28/03/93'. This is the most powerful way to search a database, since all the searching features are available. Regrettably, it can also be the hardest to use, since you have to learn and remember at least some of the *Idealist* search syntax.

See the topics on searching and the search command syntax for more details.

Groups

Use this dialog to create and define the permissions of groups of users that are allowed access to the current database.

Groups lists the names of groups of users you have defined. There are three pre-defined groups: Administrators, Users and Guests.
Attributes defines the access for each group.
- **Change security** when checked, this allows a member of the group to change the security attributes associated with a database. This is typically only checked for the Administrators (or equivalent) group.
- **Edit record** when checked, this allows a member of the group to edit (make changes to) a record.
- **Create record** when checked, this allows a member of the group to create a new record.
- **Delete record** when checked, this allows a member of the group to delete a record from the database.
- **Reindex** when checked, this allows a member of the group to reindex the database.

Add... allows you to add another group to the list of user groups. Groups names contain up to 31 characters.
Delete deletes the currently selected group.

Users

Use this dialog to create and define the users that are allowed access to the current database. There are two pre-defined users; Guest (a member of the Guests group) and one with your login name (you are a member of the Administrators group).

Users lists the names of all allowed users of the current database.
Group lists the names of all groups of users associated with the current database. The currently highlighted user is a member of the currently highlighted group. Each user must belong to one group.
Password gives the password that the user must supply when opening the database. If this is left blank, then the user is not required to enter a password.
Add... adds a new user to the list of users. User names are up to 31 characters in length.
Delete deletes the current user.

Log

Use this dialog to define the events that will be recorded in *database*.LOG. The log file contains details of one or more of the following events, detailing the time, date and user name associated with each:

Create database
Open database
Close database
Create record
Delete record
Edit record
Search the entry in the log file includes the full search command.

File

This dialog allows you to import/export the current security settings to/from a comma-separated file.

Each line in the file has the format:

```
user,password,group,flags
```

where flags (corresponding to the attributes of a group) is an addition of:

```
1    change security
2    edit record
4    create record
8    delete record
16   reindex
```

Beware these CSV files contain all the security information, including passwords: do not leave them laying around your hard disk!

Field define dialog

The big list box displays the fields that have already been defined, together with their data type and options.

Add allows you to add a new field type.

Options... produces a further dialog box allowing you to set the options for the currently-selected field type.

Delete deletes the currently-selected field type. You will not be able to delete the default field type, called 'Text'.

Record define dialog

Record type, the list box on the left, contains a list of the record types that been defined. The one pre-defined record type that is always available is called 'Default'.

Add, the button under the Record type list box, allows you to add a new record definition. When pressed, you will be prompted for the name of a new record type. This must start with an alphabetic character, and be followed by up to 30 alphanumeric characters. Each record type may contain up to 160 fields.

Delete, the button under the Add button, deletes the currently-selected record type from the Record type list box. You will not be able to delete the 'Default' record type.

Fields in record, the list box in the middle of the dialog, list the fields in the currently-selected record type. Each record type may contain up to 160 fields.

Remove, the button underneath the Fields in record list box, removes the currently-selected field from the currently-selected record type. If the record type only contains one field, you will not be able to remove it.

Defined fields, the list box on the right of the dialog, contains a library of fields defined in the Field define dialog.

<< Insert, the button underneath the Defined fields list box, inserts the currently selected Defined field into the Fields in record list box.

<< Append, the button underneath the << Insert button, appends the currently selected Defined field to the end of the Fields in record list box. Double clicking on a field in the Defined fields list box has the same effect.

Add field definition dialog

Field name Enter a field name of up to 31 alphanumeric characters. Field names cannot contain spaces. The first character of the field name will be changed to uppercase automatically.

Data type See the section on field data types for more information.

Import record and field definitions dialog

View options dialog

Windows

Menu Bar If you deselect this option, the menu bar is removed from the *Idealist* window. This prevents users from selecting menu commands. Press CTRL+O to display the View Options dialog box.
Status Bar If you deselect this option, the status bar is no longer visible at the bottom of the *Idealist* window. This allows slightly more space for database windows, but you will be denied the information normally displayed in the status bar.
Button Bar If you deselect this option, the button bar is removed from the window. This allows more space for database windows, but you can no longer use the buttons. Since the button bar can only be accessed and modified with a mouse, you can deselect this option if you only use the keyboard.
Horizontal Scroll Bar If you select this option, a horizontal scroll bar appears at the bottom of the current database window. By clicking on this bar, you can move to parts of fields that are beyond the edge of the database window. You may wish to deselect the horizontal bar to allow slightly more space for fields in the database window if your fields are short, or if you use Wrapped Text fields for long fields. Records cannot be scrolled horizontally (for example by pressing HOME or END) if the horizontal scroll bar is disabled.
Vertical Scroll Bar If your records contain relatively few short fields only, you can disable the vertical scroll bar to allow slightly more space in the database window. Records cannot be scrolled vertically (for example by pressing PGUP or PGDN) if the vertical scroll bar is disabled.
Center Dialogs When on, all *Idealist* dialogs are centred within the main *Idealist* window. When off, they cascade from the top left of the main *Idealist* window.

Fields

Hide Empty Fields When on, empty fields are not displayed. Bear in mind the side-effect of this when looking at an empty record.
Source If you deselect this option, *Idealist* displays the results of Calculated, Name and Graphic fields, and you will not be able to edit these fields. If the option is selected, *Idealist* shows the expression, name text or graphic file name, and allows you to alter the field contents. Other field types are not affected by Source and Result mode. Keyboard shortcut: F9.
Highlight hit words If you select this option, hit words in plain and rich text fields will be highlighted. Hit words in Notepad text fields cannot be highlighted. Turning off this option may increase the display speed.

Plain text fields

See ¶ in Wrapped Fields This option only affects wrapped plain text fields (*i.e.* not rich text fields). If the option is selected, “hard” line breaks (inserted by pressing ENTER) in Wrapped Text fields are indicated by pilcrow (¶) symbols. “Soft” line breaks, which are inserted and adjusted when necessary by *Idealist* so that the text does not extend beyond the edge of the database window, are not shown (whether or not this option is selected).
See Invisible Characters This option only affects wrapped plain text fields (*i.e.* not rich text fields). When selected, spaces are displayed as small dots and tabs as chevrons.
Highlight mis-spellings This options only applies to plain text fields (*i.e.* not rich text fields). When selected, any indexable words not found in the spelling checker dictionary will be highlighted in red.

Hit word color

Foreground Change the color used to display the text of hit words. This setting is independent of the rest of the color settings in *Idealist* (derived from the Control Panel) so choose it with care.

Name...

Select the format to use when displaying lists of names in fields with the Name data type in Result mode. The name formats are stored in a text file called NAME.LST. For further details see Name Formatting.

Modify AutoOpen commands dialog

When *Idealist* starts, it looks for a small text file called IDEALIST.CMD in the application directory: if it exists, it runs the commands listed in the file.

When a database is open, *Idealist* looks for a small text file called *database*.CMD in the same directory as the database. If such a file exists, *Idealist* runs the commands listed in the file.

This dialog lets you edit these .CMD files. If no database is open, or if the current database is minimized, then IDEALIST.CMD is edited. Otherwise, *database*.CMD is edited. The name of the .CMD file being edited is displayed in the title bar.

Pressing the **Insert** button pops up a list of available commands.

Typically, IDEALIST.CMD might contain:

```
MoveWindow(0,0,639,479)
```

Typically, *database*.CMD might contain:

```
ShowOver()
Find("!Home")
```

Field options dialog (text fields)

The name of the field type for which the options apply is give in the title bar of the dialog.

Text

Plain text fields can contain embedded command buttons, and use minimal disk space. However, they can only contain one font. Plain text fields can hold up to 64,000 characters.
Rich text field cannot contain embedded command buttons, and consume quite a lot of disk space. However, they can contain more than one font, including bold, italic, underline, superscript and subscript text. Rich text fields can hold up to 250,000 characters (please approach this limit sparingly - it can cripple performance).
Notepad text fields use the same text editor as Windows Notepad. They cannot contain embedded command buttons, cannot perform hit word highlighting and can only contain upto 27,000 characters of text. They are normally only used for Arabic text, or other specialized uses.

Lines

Single line text fields can only contain one line of text.
Normal text fields can contain many lines of text, and each line will end when terminated by a newline. These are the most efficient fields.
Wrapped text fields adjust their contents to fit within the current field window width. These fields are less efficient than normal text fields.

Content

Vocabulary provides fields of this type with a controlled vocabulary list (*i.e.* a list of words and phrases from which the contents of the field must be selected). Any number of words and phrases may be chosen from the vocabulary by the user. When editing a record, the vocabulary is popped up by either double-clicking the mouse in the field or by pressing the INS key or by selecting Edit, Insert, Vocabulary from the menu.
Pick list provides fields of this type with a controlled list of words and phrases from which the contents of the field must be selected. Only one word or phrase may be chosen from the list by the user. When editing a record, the list is popped up by either double-clicking the mouse in the field or by pressing the INS key or by selecting Edit, Insert, Pick list from the menu.
Edit... pops-up a text edit that lets you edit the contents of either the vocabulary or pick list (a field can have one or the other - not both).

Flags

Mandatory fields must have something entered into them before the record can be saved.
Unindexed are, er, not indexed. This is useful for large text fields that will never be searched.
Spellcheck fields have their contents spellchecked after they have been edited.

Help

Comment enabling this check box lets you provide each field type with a short piece of explanatory text. This can be displayed by the user by pressing CTRL+F1 when the cursor is in a field.
Edit... pops-up a text edit that lets you edit the field comment.

Case

UPPER forces the contents of the entire field to upper case when it is saved.
Mixed forces the first letter of each word to uppercase, and the remaining letters to lowercase, when the field is saved.
lower forces the contents of the entire field to lower case when it is saved.

Font

Change... brings up another dialog that lets you change the font and color used to display the field.

Other options

Max. chars restricts the maximum number of characters than be entered into a field.
Visible lines provides a life-saver for large text fields.
Template defines a character pattern that the contents of the field must correspond to.
Default defines the default content of the fields type.

Field options dialog (scalar fields)

The name of the field type for which the options apply is give in the title bar of the dialog.

Flags

Mandatory fields must have something entered into them before the record can be saved.
Unindexed are, er, not indexed. This is useful for large text fields that will never be searched.

Help

Comment enabling this check box lets you provide each field type with a short piece of explanatory text. This can be displayed by the user by pressing CTRL+F1 when the cursor is in a field.
Edit... pops-up a text edit that lets you edit the field comment.

Font

Change... brings up another dialog that lets you change the font and color used to display the field.

Default defines the default contents of the field type.

Calculated field expressions

Note: use the << and >> browse button to see the list of available functions, or press the Help button in the Insert Function dialog in *Idealist*.

A field defined to have the Calculated data type can contain a BASIC-like expression containing functions, numeric and string constants, field names and operators. For example:

```
Left(Text,3)+Right(Text[2],3)
If(RcvdDate>Date(), "Early", "Late")
(2143/22)+AmtRcvd
PurchPrice*Glossary(TaxRate)
```

When in Result mode, these expressions are evaluated and their results shown in place of the expression. Fields in the record are evaluated as they are loaded, progressing from the first field to the last.

Note that the functions in a Calculated field are not the same as commands in embedded or button bar buttons.

Calculated fields are not indexed.

All variables and constants are represented as text. If a function requires a number, then the text is automatically converted into a number, and vice versa. For examples, it might appear that: Abs(2134) is an error (how can you have an absolute value of a string of characters?), but the string is converted into a number (2134) before being passed to the function. Likewise, Ucase(2143) simply returns 2143.

Field names may either be given on their own (*e.g.* Text) or with a suffix (*e.g.* Text[3] - referring to the third instance of a Text field in this record). Pseudo-field names such as {@Database} can also be used.

Field expression operators (from highest to lowest precedence):

```
NOT
^
* / MOD
+ -                          (+ doubles as string concatenator)
= < > <= >= <>
AND
OR
```

Pseudo-field names:

{@Database}	Database path and file name: For example: `"This database is " + {@Database}`
{@Date}	Current date in short format: For example: `"The date is " + {@Date}`
{@LongDate}	Current date in long format: For example: `"Today is " + {@LongDate}`
{@Name}	Record name. For example: `"This record is called " + {@Name}`
{@Position}	Position in hit list. For example: `"This is the " + {@Position}`
{@Time}	Current time. For example: `"The time is " + {@Time}`
{@Type}	Record type. For example: `"This record is of type " + {@Type}`

Abs(*n*)

A field expression function.

Returns the unsigned value of *n*.

AddDays(*Date, Number*)

A field expression function.

Returns the date that is *Number* days from *Date*. For example, the following returns "1/1/94":

```
AddDays("31/12/93", 1)
```

Age(Date1, Date2)

A field expression function.

If *Date1* is a date of birth, and *Date2* a date, then Age() returns the number of years that have passed between the birth date and the date. For example, the following returns 31:

```
Age("29/04/62", "28/08/93")
```

Asc(*String*)

A field expression function.

Returns the ANSI character code of the first character in *string*.

Chr(*n*)
A field expression function.

Returns the character whose ANSI code is *n*.

Clipboard()
A field expression function.

Returns the textual contents of the Clipboard.

Concat(*String1*, *String2*)
A field expression function.

Returns a string consisting of *string1* followed by *string2*.

Date()

A field expression function.

Returns today's date.

DateFormat()

A field expression function.

Returns the current Windows short date format string.

Day(*Date*)

A field expression function.

Returns the day portion of *date* as an integer. For example, the following returns 24:

```
Day("24/02/93")
```

DayName(*Date*)

A field expression function.

Returns a string containing the name of the day of the week on the given date. For example, the following returns "Sunday":

```
DayName("29/04/1962")
```

DayOfWeek(*Date*)
A field expression function.

Returns the day of the week of *date* as an integer between 1 (Sunday) and 7 (Saturday).

DaysBetween(*Date*, *Date*)
A field expression function.

Returns the number of days between the two dates.

DdeExecute(*Service*, *Topic*, *Commands*)

A field expression function.

Establishes a DDE conversation with *service* about *topic* and then sends the string *commands.* Finally, the conversation is terminated.

DdeRequest(*Service*, *Topic*, *Item*)

A field expression function.

Establishes a DDE conversation with *Service* about *Topic* and tries to obtain *Item*. For example:

```
DdeRequest("Excel", "Sales.xls", "Jan92")
```

Glossary(*Name*)
A field expression function.

Searches the glossary for an entry called *name* and returns that entry if it exists. Useful for storing numerical constants (*e.g.* sales tax rate, discount level *&c*).

Hour(*Time*)

A field expression function.

Extracts the hour portion of *time* and returns it as an integer.

If(*Test*, *TrueResult*, *FalseResult*)
A field expression function.

If *test* is true (non-zero) then return *trueresult* else return *falseresult*.

Instr(*Source*, *Search*)
A field expression function.

Searches for *search* in *source*. Returns the number of the character where *search* started, or zero if *search* is not found in *source*.

Int(*n*)
A field expression function.

Returns the integer part of *n*.

IsField(*FieldExpr*)

A field expression function.

Returns 1 if *FieldExpr* can be found in the current record, 0 if it cannot. For example:

```
IsField("Text")
IsField("Text[3]")
```

Julian(*Date*)

A field expression function.

Returns the Julian number corresponding to *Date*. See also **ToDate()**, which turns a Julian number into a date.

Lcase(*String*)
A field expression function.

Return *string* converted to lower case.

Left(*String*, *Count*)
A field expression function.

Returns the *count* leftmost characters of *string*.

Len(*String*)
Length(*String*)
A field expression function.

Returns the number of characters in *string*.

Ltrim(*String*)
A field expression function.

Returns *string* with any leading spaces or tabs removed.

Max(*Number1*, *Number2*)
A field expression function.

Returns the larger of the two *number*s.

Mid(*Source, Index, Count)*
A field expression function.

Returns *count* characters from *source*, starting at character *index*.

Min(*Number1, Number2*)
A field expression function.

Returns the smaller of the two *number*s.

Minute(*Time*)

A field expression function.

Extracts the minute portion of *time* and returns it as an integer.

Month(*Date*)

A field expression function.

Returns the month number, extracted from *Date*. For example, the following returns 4:

```
Month("1993.04.29")
```

MonthName(*Date*)

A field expression function.

Returns a string containing the name of the month on the given date. For example, the following returns "April":

```
MonthName("29/04/1962")
```

Right(*String, Count*)
A field expression function.

Returns the *count* rightmost characters from *string*.

Rtrim(*String*)

A field expression function.

Returns *string* with any trailing spaces or tabs removed. A synonym for **Trim**(*String*).

Second(*Time*)
A field expression function.

Extracts the second portion of *time* and returns it as an integer.

Sgn(*Number*)
Sign(*Number*)
A field expression function.

Returns the sign of *number*; 1 for a positive number, -1 for a negative number or 0 for zero.

Strim(*String*)
A field expression function.

Returns *string* with any spaces or tabs removed.

String(*Count, String*)
A field expression function.

Returns the first character of *string*, repeated *count* times.

System()
A field expression function.

Returns the current version number of Windows.

Time()

A field expression function.

Returns the current time.

TimeFormat()

A field expression function.

Returns the current Windows time format string.

ToCurrency(*Number*)

A field expression function.

Returns a number formatted as a currency. For example

```
ToCurrency(2143/22)
```

might produce:

```
£97.40
```

ToDate(*Number*)

A field expression function.

Returns the date corresponding to the given Julian number. See also **Julian()**, which turns a date into a Julian number.

ToTime(*Seconds*)
A field expression function.

Takes an integer number of seconds as its argument and returns a correctly formatted time string.

Trim(*String*)
A field expression function.

Returns *string* with any trailing spaces or tabs removed.

Ucase(*String*)
A field expression function.

Returns *string* converted to uppercase.

Version()
A field expression function.

Returns the current version of *Idealist*.

Xtrim(*String, Chars*)

A field expression function.

Returns *String* with any occurrence of any character in *Chars* removed. For example, the following returns "HellWrld"

```
Xtrim("Hello World", "o ")
```

Year(*Date*)

A field expression function.

Returns the year number, extracted from *Date*. For example, the following returns 1993:

```
Year("93.04.29")
```

Automatic fields

Automatic fields are those which will be filled in by *Idealist*, rather than by the user. They are recognized by *Idealist* by their name and/or type:

Created
If you define a date field called 'Created', *Idealist* will automatically fill in that field with today's date whenever a record containing a 'Created' field is created.

CreatedTime
If you define a time field called 'CreatedTime', *Idealist* will automatically fill in that field with the current time whenever a record containing a 'CreatedTime' field is created.

Creator
If you define a field called 'Creator', *Idealist* will automatically fill in that field with the name of the current user whenever a record containing a 'Creator' field is created.

Modified
If you define a date field called 'Modified', *Idealist* will automatically fill in that field with today's date whenever a record containing a 'Modified' field is modified.

ModifyTime
If you define a time field called 'ModifyTime', *Idealist* will automatically fill in that field with the current time whenever a record containing a 'ModifyTime' field is modified.

Modifier
If you define a field called 'Modifier', *Idealist* will automatically fill in that field with the name of the current user whenever a record containing a 'Modifier' field is modified.

RecNo
If you define a field of any type called 'RecNo', *Idealist* will automatically fill in that field with the current record number whenever a record containing a 'RecNo' field is created.

Use record numbers in records with great caution:

1. *Idealist* does not guarantee that record numbers will be sequential, or indeed in any order.

2. Merging two databases with record numbers inserted into fields can create duplicates.

Character sets

In Windows and MS-DOS, a codepage is a table that maps the ANSI or ASCII character number (in the range 0 to 255, eg 171) to the character that appears on the screen. Each codepage is refered to by a number. By default, Windows uses a codepage called 1252 (sometimes known as "Latin1") that maps ANSI character 171 to « (a left pointing double chevron). By default, MS-DOS uses a codepage called 437 (or sometimes 850), that maps ASCII character 171 to the symbol for one-half (which I can't show here, because I'm writing this in Windows, not MS-DOS!). If your computer is set up to run in a country other than the US or western Europe, then you computer will have a different set of codepages installed (for example, if you are in Eastern Europe, Windows will use codepage 1250, and character 171 may well appear as something quite unexpected).

The most commonly used Windows character set is called Windows Latin 1, which is based on ISO 8859-1. It includes characters and symbols for English, French, German, Spanish, Italian, Danish, Dutch, Faeroese, Finnish, Icelandic, Irish, Norwegian, Portuguese and Swedish. By default, *Idealist* uses the Windows Latin 1 character set.

The following is a list of the character sets used by Windows, and their code page numbers:

```
1250 Eastern European (Latin 2)
1251 Russian/Cyrillic
1252 Default (Latin 1)
1253 Greek
1254 Turkish
1255 Hebrew
1256 Arabic
1257 Farsi
```

Note that these are different to the code pages used by MS-DOS, even if their descriptions appear similar. For example, the characters in the Windows 1252 code page are different to the characters in the 'equivalent' MS-DOS code pages, 437 or 850, even though 1252 and 850 are both known as 'Latin 1'.

Idealist contains built-in support for codepages 1250 and 1252. If you are using Windows installed with any other codepage, then you will have to create a customized Character Value Table.

Data types

Each field in *Idealist* belongs to one of the following data types:

Calculated
Calculated fields contain BASIC-like expressions. These expressions are entered with *Idealist* in Source mode, and those expressions are evaluated, and the results displayed, when *Idealist* is in Result mode.

Calculated fields are not indexed.

Currency
Currency fields contain numbers which are displayed as an amount of money (*e.g.*1.2 might be displayed as $1.20). Currency fields are stored as eight-byte double precision floating point numbers.

Currency fields are formatted according to the setting in Control Panel, Regional Settings, Currency.

Currency fields are indexed in a separate currency index.

Date
Date fields contain dates entered in the short date format defined in Control Panel.

Date fields are formatted according to the setting in Control Panel, Regional Settings, Date.

Date fields are indexed in a separate date index.

Graphic
Graphic fields contain the filename of a graphic file. The graphic is displayed in Result mode.

Graphic fields are not indexed.

Identifier
Identifier fields contain a single string of characters which is indexed as it appears in the field - no attempt is made to break the string up into words before it is indexed. This is very useful for indexing things that contain non-indexable characters, for example serial or part numbers, ISBNs and so on. Each identifier field can be up to 63 characters long.

Identifier fields are indexed in a separate identifier index.

Integer
Integer fields contain whole numbers in the range +/- 2,147,483,647. They are stored internally as 32-bit signed integers.

Integer fields are indexed in a separate integer index.

Names
Name fields contain a list of surnames and initials that can be parsed by *Idealist* and displayed in a specified format in Result mode.

Name fields are indexed in the main text index.

Number
Number fields are stored as eight-byte double precision floating point numbers.

Number fields are formatted according to the setting in Control Panel, Regional Settings, Number.

Number fields are indexed in a separate number index.

Text

Text fields contain text which is broken up into separate words as it is indexed.

Text fields can be single line, multiline or wrapped multiline. They can either contain plain text or rich text:

Features of plain text fields:

1. Up to 64,000 characters of text per field
2. Reliable hit term highlighting
3. Embedded command buttons in the text
4. One font per field
5. Compact when saved to disk

Features of rich text fields:

1. Up to 250,000 characters of text per field
2. Slightly weird hit term highlighting
3. No embedded buttons
4. Multiple fonts/font effects per field
5. Not compact when saved to disk

Text fields are indexed in a separate text index.

Time

Time fields contain times entered in the time format defined in Control Panel.

Time fields are indexed in a separate time index.

Dynamic Data Exchange

Idealist supports Microsoft Windows Dynamic Data Exchange (DDE). You can use *Idealist* as a DDE client by including DDE expressions in *Idealist* fields to access information from other Microsoft Windows applications, and you can use *Idealist* as a DDE server by sending instructions to *Idealist* from other Microsoft Windows applications that support DDE.

Idealist as a DDE Client

When viewing records in Source mode, type an expression into a Calculated field of the form:

```
DDERequest(Application,Topic,Item)
```

for example:

```
"Quote is:" + DDERequest("Excel","NYSE.XLS","IBMprice")
```

When viewed in Result mode, the field will contain the information from the IBMprice named region of an Excel spreadsheet called NYSE.XLS.

The database is opened automatically if it is not open already.

Idealist as a DDE Server

From another application, initiate a conversation with *Idealist*, using either "IDEALIST" or "IWIN" as the application name, and the name of an open *Idealist* database as the topic. A conversation will be initiated with the database window. Each database window can hold any number of concurrent conversations.

Then, send DDE execute, request or poke messages. Since DDE request messages can only request data from the current record of a database, DDE execute messages should be sent beforehand to ensure that the required record is the current record.

DDE execute

Send DDE execute commands, enclosed in square brackets [], of the form:

```
[Find("Name=Björk Gudmundsdottir")]
[Widen("Smith or Jones")]
[Narrow("\"soundex\"")]
[Stack Exclude("Björk Gudmundsdottir") Overview Backtrack]
[CrossRef("!Home")]
[Previous]
[Next]
```

A full list of the available commands can be found in the section on commands.

More than one command can be given at once, as in:

```
[Stack Find("!Home")]
```

These commands are executed as if they were invoked from the menu, except that any messages they may produce are suppressed. Note that character strings should be enclosed in quotes "".

DDE execute messages cannot be sent from *Idealist* to *Idealist*.

DDE execute example

The following Microsoft Word for Windows 2.0 macro starts a conversation with an *Idealist* database called 'Test', displays the full pathname of the database and then imports a text file called 'macbeth.txt' into it:

```
Sub MAIN
DDETerminateAll
Command$ = "[Import(4,"c:\\i\\imp\\macbeth.txt","c:\\i\\imp\
\macbeth.imp", "", "",0)]"
Channel = DDEInitiate("Idealist", "Test")
If Channel <> 0 Then
 MsgBox DDERequest$(Channel, "{@}")
 DDEExecute Channel, Command$
EndIf
DDETerminate Channel
MsgBox "DDE completed"
End Sub
```

Note that the quotes used inside the command string are those from the extended character set (147 and 148); *Idealist* accepts these as quotes, whereas WordBasic thinks they are part of the string. This trick allows strings for *Idealist*'s use to be embedded in WordBasic strings.

DDE requests

Send DDE request messages of the form:

Request	Returns
{Field}	the contents of first occurrence of *Field* in the current record
{Field[n]}	Nth occurrence of *Field* in the current record.
{@Checksum}	a checksum for the current record
{@Database}	the name of the database
{@Fields}	a list of fields in the current record
{@FieldName}	the name of the current field
{@FieldType}	the type of the current field
{@Name}	the name of the current record
{@Position}	the current record position in the hit list
{@Recno}	the internal "number" of the current record
{@Type}	the type of the current record

DDE pokes

Send DDE poke messages to a database window, with "Idealist" or "iwin" as the application name, the name of the database as the topic, and a field expression (*e.g.* "{Text}" or "{Address[2]}") as the item. The contents of the field will be replaced by the data accompanying the DDE poke message.

The SYSTEM topic

If the name of the topic is SYSTEM, rather than the name of a database, the following items are available for request:

Item	Returns
SYSITEMS	a list of SYSTEM items.
TOPICS	a list of open databases.
STATUS	either "Ready" or "Busy"

```
APPSTATUS...a hexadecimal number in an ASCIIZ string;
            see the IfStatus command for details.
FORMATS     a list of supported clipboard formats.
FIELDTYPES  a list of the defined fields
RECORDTYPES a list of the defined records
VERSION     the Idealist version string
```

Terminating conversations

End each conversation by sending a DDE terminate message.

Getting *Idealist* to send out DDE Advise messages

Idealist supports "hot" or "automatic" advise loops. You can start an advise loop with the "SYSTEM" topic and the following items:

```
Item          Returns

Open          Name of database opened
Close         Name of database closed
Activate      Name of database activated
Command       Name of command
```

You can start an advise loop with a fully-instantiated database name as the topic with the following items:

```
Item           Returns

CreateRecord   Record number
LoadRecord     Record number
SaveRecord     Record number
DeleteRecord   Record number
```

Escape sequences

These are combinations of a backslash character ('\') with a keyboard character. Escape sequences are used to represent untypeable characters. In *Idealist*, escape characters are used in import and export format files.

Escape	Meaning	ANSI
\b	Backspace	8
\f	Form feed	12
\n	New line	10
\r	Return	13
\s	Space	32
\t	Horizontal tab	9
\v	Vertical tab	11
digits	User defined	*digits*

Files and file types

BAR Textual. Contains configuration information for both the main window button bar (IDEALIST.BAR) and any database-specific button bars (*database*.BAR).
CLX Binary data file containing compressed spelling dictionary.
CMD Textual. Contains a list of *Idealist* commands. The commands in IDEALIST.BAR are executed whenever *Idealist* is started. The commands in *database*.CMD are executed whenever *database* is opened.
CMT Textual. Contains commentary information on a defined field.
CNT Textual. Contains the contents page of a Windows help file. This file is turned into a hidden .GID file by Windows.
CVT Textual. Contains a Character Value Table (CVT) that will replace the default CVT.
DBF Binary. Contains a database in one of the xBase/dBase formats.
DBT Binary. Contains memo fields belonging to *filename*.DBF.
DEF Textual. Contains *Idealist* field and record definitions. IDEALIST.DEF contains definitions global to all databases; *database*.DEF contains definitions local to *database*.
DIC Textual. Contains words from the user's personal spelling dictionary.
DIR Binary. Part of a database's index. (Contains the directory portion of an extendible hash table.)
DLL Binary. Dynamically linked library. Contains program code for doing odd jobs like image file decompression or spell checking.
EXP Textual. Contains export formats for a variety of record types.
FPT Binary. Contains memo fields belonging to *database*.DBF.
HIT Binary. Contains a saved hit list.
HLP Binary. Contains a Windows formatted help file.
IMP Textual. Contains an import format for one record type.
INI Textual. Contains initialisation options.
LAY Textual. Contains layout information for a variety of record types. *database*.LAY is opened automatically whenever *database* is opened.
LST Textual. Contains a list of things.
NET Binary. Used by *Idealist* to implement record and index locking when a database is open in "multi-user" mode. This file will be generated automatically if required; it need not be backed up as part of a database.
OCC Binary. Part of a database's index. (Contains compressed occurrence lists.)
PCK Textual. Contains a list of words or phrases that are selectable by a 'pick list' field.
PRN Textual. Contains print formats for a variety of record types.
ROT Binary. Part of a database's index. (Contains a Record Offset Table - an array of offsets into *database*.TEX where a particular record can be found.)
SEC Binary. Contains the security information for a database. If a database has a .SEC file, try very hard not to delete it.
TEM Binary. Contains a record template.
TEX Binary. Contains the records of a database.
TRM Binary. Part of a database's index. (Contains a series of pages, each page containing a sorted list of indexed words and an offset into *database*.OCC.)
TXT Textual.
VCB Textual. Contains a list of words or phrases that are selectable by a 'vocabulary' field.

Natural format

A protocol for storing *Idealist* databases in text files. It provides a compact, simple and robust method for storing and transferring *Idealist* databases.

Each record starts with a colon at the start of a line, followed immediately by the record type. This is optionally followed by an equals sign and the name given to the record.

Thereafter, each field within the record starts with a hyphen at the start of the line, followed immediately by the type name of the field. This is optionally followed by complete type information for that field. The line following this field-description line is the first line of the field.

For example, here is a simple Natural format file:

```
:Default
-Text
Mary had a little lamb
Whose fleece was ripe for plucking

:Default
-Text
When you were here before
-Text
Couldn't look you in the eye
```

Here is a more complex example, featuring record names and full field type information:

```
:Address=HeadOffice
-Company Type="Line Of Text" Font="Courier" Mandatory
Widgets Corp.
-Street Type="Line Of Text" Font="Courier"
42 Eris Avenue
-Town
Discordia
-Notes Type="Wrapped Text" Font="Times Roman"
```

The field type information is in the same format as that found in IDEALIST.DEF and *database*.DEF - except that the line is prefixed with a hyphen.

If *Idealist* imports a Natural format file that contains complete field type information, and the fields that are imported have not previously been defined in the receiving database, then field definitions are constructed automatically.

Regular expressions

Regular expressions are a notation for specifying and matching strings of characters. Regular expressions are widely used in UNIX programs, including its text editors and shell. Restricted forms of regular expressions also occur in MS-DOS as 'wildcard characters' for specifying sets of filenames.

In *Idealist*, regular expressions are used in import format files to describe the patterns of characters that delimit the data you wish to import from inflected and non-inflected text files.
The simplest regular expression is a string of letters and numbers, like `Asia`. This pattern matches the character string `Asia`.

Some characters in a regular expression have a special meaning, allowing special sequences of characters to be matched:

```
\ ^ $ . [ ] * + -
```

These are called 'metacharacters'. Since a regular expression consisting of a single non-metacharacter matches that character, a situation might occur in which we may want to match one of the characters designated as metacharacters, for example the dollar sign.

In this case, we prefix the character with a backslash. Thus, the regular expression `\$` matches the character `$`. If a character is preceded by a single `\`, we say the character is 'quoted'.

In a regular expression, an unquoted circumflex `^` matches the beginning of a line, an unquoted dollar sign `$` matches the end of a line, and an unquoted period `.` matches any single character. Thus:

`^C`	matches C at the start of a line
`C$`	matches C at the end of a line
`^C$`	matches a line consisting of the single character C
`^.$`	matches a line consisting of any single character
`...`	matches any three consecutive characters
`\.$`	matches a period at the end of a line

A regular expression consisting of a group of characters enclosed in brackets `[]` is called a 'character class'; it matches any one of the enclosed characters. For example, `[AEIOU]` matches any single vowel character.

Ranges of characters can also be abbreviated in a character class by using a hyphen. The character immediately to the left of the hyphen defines the beginning of the range; the character immediately to the right defines the end. Thus, `[0-9]` defines any digit, and `[A-Za-z][0-9]` matches any letter (upper or lower case) followed by a digit.

A 'complemented' character class is one in which the first character after the left bracket is a circumflex `^`. Such a class matches any character *not* in the group following the circumflex. Thus, `[^0-9]` matches any character except a digit. This use of the circumflex is distinct from its use to match the start of a line.

`^[ABC]`	matches A, B or C at the start of a line.
`^[^ABC]`	matches any character except A, B or C at the start of a line.
`[^ABC]`	matches any character except A, B or C.

Inside a character class, all characters have their literal meaning, except for the quoting character `\`, the circumflex at the start of the class and the hyphen between two characters. Thus, `[.]` matches a period

and `^[^^]` matches any character except a circumflex at the start of a line.

The asterisk and plus characters are used to specify repetitions in regular expressions. If *r* is a regular expression, then *r** matches any string consisting of zero or more sub-strings matched by *r*, and *r*+ matches any string consisting of one or more sub-strings matched by *r*.

`B*`	matches the empty string or B or BB or BBB *&c.*
`AB*C`	matches AC or ABC or ABBC *&c*.
`AB+C`	matches ABC or ABBC *&c*.
`[A-Z]+`	matches any string of one or more upper case letters.

Regular expressions match text on one line only: you cannot use them to match text that spreads over more than one line. For example `^$` is correct (start then end of line), but `$^` is not allowed (end then start of line).

Within regular expressions (and in export format files), escape sequences can be used to specify characters for which there may be no other notation.

The following escape sequences (which are very useful in exporting data) are made redundant by the use of `^` and `$` and should be avoided in regular expressions:

\r	Carriage return
\n	New line

Search syntax

The *Idealist* search engine processes search commands that are generally of the form **field operator term-expression**.

Field

The field part of a search command can be either:
1. The name of a field in the database, for example 'Text'.
2. The name of a field's data type, enclosed in parentheses. This directs the search to all fields of that data type. This must be one of:

```
(text)     all indexed textual fields
(date)     all indexed date fields
(time)     all indexed time fields
(number)   all indexed number fields
(integer)  all indexed integer fields
(currency) all indexed currency fields
(name)     record names, e.g. !Home
(type)     record types, e.g. :Default
```

3. The field part of a search command may be missing entirely. This is not recommended, since it forces *Idealist* to guess the part of the index to search in: with terms like 'elephant' this is not a problem (the textual terms in the index will be searched) but with a term consisting of a variety of alphanumeric characters - for example '1234' - this can produce unexpected results.

Comparator

The comparator part of a search command can be one of

```
=   Equal to/contains
==  Exactly equal to
<>  Not equal to/does not contain
><  Not exactly equal to
<   Less than
>   Greater than
<=  Less than or equal to
>=  Greater than or equal to.
```

If the field is specified, then a comparator must be specified too. If the field part is missing, then the comparator is optional: if one is not given then '=' is assumed. For example:

```
Text = Untouched   ; lovely, just fine.
= Bitter           ; text fields will probably be searched
(Text)=Pictures    ; all text fields will be searched
Tattooed           ; assumes '= Tattooed'
Text Laughter      ; error - no comparator
```

Term-expression

The contents of the term-expression part of a search command vary according to the type of field being searched.

Scalar fields

If the field is scalar (i.e. a number, integer, currency, date or time field) then the term is expected to be formatted according to the rules specified in Control Panel. For example:

```
Birthday=4/4/78
Salary>$34,000
(Time)=21:34
Ticket=2134
```

Range searches in scalar fields

You can search for a range of scalar values like so:

```
Date=1/1/94 [TO] 1/1/95
```

Textual fields

Simple textual searches specify one word to search for, for example:

```
Address=Bladon
(Text)=richmond
```

Phrase searches

More than one term separated by spaces will trigger a search for a field containing that phrase. One gotcha with phrase searches is that stopwords will not be in the index, and so can confuse the search. For example, searching for 'To Have And Have Not' will fail, because that phrase consists entirely of stopwords. For *Idealist* to search for a phrase, it must contain at least one word which is not a stopword.

Wildcards

You can make text searches more flexible using a wildcard. Wildcards allow you to search for all terms that begin with, end with, or include the search term you specify. The wildcard character is **`*`**. The asterisk can be thought of as a special character that means ‘replace me with 0, one or more occurrences of any character you like’. For example, if you were interested in chlorine compounds in a chemical database, you would want to find terms such as *chloride*, *chlorate*, *perchlorate* and *chlorination* as well as *chlorine* itself. A suitable search term that would match with all these terms would be **`*chlor*`**. To restrict the hit list to records containing terms that begin with *chlor-*, use **`chlor*`**.

Proximity searches

Search for one word within the same paragraph as another by using the [near] operator, for example:

```
Abstract=beetle [Near] wood
```

Search for one word within the same sentence as another by using the [.] operator, for example:

```
Abstract=beetle [.] wood
```

Search for one word within a given number of words from another word by using the [*n*] operator, for example:

```
Abstract=beetle [10] wood
```

Sound-alike searches

If you want to search for all occurrences of a word that sounds like another word, enclose the word in double quotes. For example:

```
(Text)="smyth"
```

might find records containing smith, smyth and smithe in any text field.

Record types and names

If the 'field' is a record type or name, then the term-expression must consist of one record type or name. If a record type is prefixed with a colon, then the record type index is searched. If a record name is prefixed with an exclamation point, then the record name index is searched.

```
(Type)=:Default
(Type)=Default
:Default
(Name)=!Home
!Home
```

If a term is not given, *Idealist* searches for an empty field.

Multiple search commands

More than one search command can be combined using parentheses and the and & or | and not ¬ symbols. For example

```
(black | ash | fire) & (sun | son)
```

Stack-based search engine

The *Idealist* search engine is stack based. The hit list stack is used internally to perform multi-command searches. So you can 'merge' hit lists by using the following method: stack a hit list, create a new one then search for just & to narrow the current hit list with the one below it on the hit list stack. Likewise, you could use | to widen the current hit list with the one below it on the hit list stack.

Internally, widening a hit list is implemented by prefixing the search command with an or symbol, for example 'sheets of empty canvas' becomes '|(sheets of empty canvas)'.

Creating your own database

Creating a new database requires four basic steps:

Define your objectives and understand your data.

Define field types. Unlike other more tabular database managers, in *Idealist* you define a library of field types independently of the record or records they will end up being used in. See the help in the Field, Define dialog for more details.

Define record types. This simply involves picking fields from the library of field definitions and copying them into one or more record types. See the help in the Record, Define dialog for more specific help.

Create/import records. Finally, you assemble a database by either manually creating new, blank, records and typing in data into each of the fields, or by importing records from an external textual source.

Editing

(under construction)

Inside a field
- Typing
- Cutting/pasting

Between fields
- Inserting/Appending fields
- Inserting things

Exporting

Exporting refers to the process of copying information from *Idealist* records to a text file. You can use *Idealist's* Natural format, standard formats such as tab or comma separated files, or define your own export format file to customize the layout, order, capitalization *&c* of fields.

Export format files

Export format files are used when copying groups of fields and records from an *Idealist* database to a text file. The idea is that, for each record, the text of each field will be output in place of the name of that field in the export format. Any literal text (including punctuation, spaces, tabs, new lines or escape sequences) that surround the field names are output just as they appear.

Each export format file can specify the format of as many record types as you like. For example:

```
:Default
Literal text:{Text}
------- Cut Here -----

:Paper
{#} {Authors} {Title}
```

Field names must be enclosed in braces to distinguish them from literal text. Field names may either be given on their own (*e.g.* Text) or with a suffix (*e.g.* Text[3] - referring to the third instance of a Text field in this record).

If a field name is given in UPPER CASE in the export format, then the contents of that field will be converted to upper case as it is copied to the output file.

Remember that fields are output as they appear on the screen; you may wish to toggle *Idealist* between Source and Result mode before exporting.

Special field names:

Some field names have special significance when they appear in an export format file:

```
Pseudo-field  Result

{@Count}      the number of records exported so far
{@Database}   the name of the database
{@Date}       the current date in short date format
{@Hits}       the current number of hits
{@LastSearch} the last search command
{@LongDate}   the current date in long date format
{@Fields}     a list of fields in the current record
{@Name}       the name of the current record
{@Time}       the current time
{@Type}       the type of the current record
```

Characters that are impossible or awkward to type may be expressed by typing escape sequences into the export format.

Editing export format files

Export format files can be created and edited using the *Idealist* Export Format Editor (accessed *via* the

Edit button in the Export dialog) or by any suitable text editor, such as Windows Notepad.

Exporting to the clipboard

If no destination filename is given, a temporary file is created to contain the exported records. After exporting has completed, the contents of this file are then automatically copied onto the Clipboard (if there is enough free memory!) and the temporary file deleted.

Importing

Importing refers to the process of copying information from another application into an *Idealist* database.

Idealist can import records directly from files in *Idealist's* Natural and database formats, from dBASE-compatible DBF files and standard formats such as tab or comma separated files. You can import records from text files in other formats by defining a suitable import format file to convert the data into a form *Idealist* can understand. The import methods are described below.

General points

If you do not specify a filename to imported, and you are not importing a DBF file, then *Idealist* will attempt to import text from the Clipboard.

Importing Natural format files

This method is used to import records that were exported in *Idealist*'s Natural format. The key features of this format are as follows:

o The record type is prefixed by a colon, which must be the first character on the line.

o If a record is named, the name appears on the same line as the record type, prefixed by an equals sign.

o Field names appear on separate lines, prefixed by hyphens, which must be the first character on the line.

o Field entries appear on a separate line below the field name, with line breaks where they occur in the field entries.

If the source program can export records as plain text files in this format, this may be the easiest way to transfer records to *Idealist*. You do not need to create an import format file to import records from a file in this format.

Importing *Idealist* databases

This method is used to import records from another *Idealist* database (*Idealist* needs the .TEX file and optionally the .ROT file to be present in order to import records from an *Idealist* database). All words in the imported records are indexed, and the records added to the existing database. You do not need to create an import format file to import records from a file in this format.

Importing inflected (tagged) files

If the program you are using cannot export records in a form that *Idealist* can immediately understand (*i.e.* Natural format, *Idealist* database format, dBASE files or tab or comma separated files), you need to decide whether the program produces inflected or non-inflected files, and create a suitable import format file.

Inflected files are a category of textual file formats, rather than a specific format. Inflected files differ from non-inflected files in that the field to which each piece of data belongs is identified explicitly in the file. The beginning and end of each field is indicated by recognizable sets of charcaters, *e.g.*:

```
AN 44031535
AU MORRIS-M. COOPER-A-S.
TI BIVALVE ENHANCEMENTS
SO J TRAD ENG
```

```
AN 44031536
AU ROE-J.
TI EXTRAPOLATING HERRING FECUNDITY TO STOCK LEVELS
SO J FISH EGG COUNTING
```

In this example, each field has a two-letter code, and the end of the field is indicated by a line break. The import format file describes these identifying strings in a format that *Idealist* can understand.

Because the fields are identified in inflected files, their order is unimportant.

Inflected import format file

The format of an inflected format file is:

```
:RecordType=RegularExpression
Field=RegularExpression
...
Field=RegularExpression
```

The regular expression following the record type defines the text pattern that is found at the end of the record in the source file.

The other regular expressions define the text pattern that is found before each field in the source file.

For example, given a source file containing:

```
TI To Have and Have Not
MO Black and White

TI The Big Sleep
MO Black and White
```

...a corresponding inflected import format file might look like:

```
:Movie=^$
Title=^TI\s
Mode=^MO\s
```

General points

If a field named in the import format file does not exist in the record, *Idealist* will attempt to append a field of that type to the record.

If the field name is omitted entirely, the text for the field will be matched but not imported. This is useful for discarding fields in the source file that are not required in the *Idealist* database.

Field names may either be given on their own (*e.g.* Text) or with a suffix (*e.g.* Text[3] - referring to the third instance of a Text field in this record).

Special fields

: or @Type Import record type (change the type of the record as you import it!)
! or @Name Import record name

Importing more than one record type

An import format file can only contain a description of how to import one record type. However, since the record type can be imported (into the pseudo-field ':') a record may be changed to a different type as it is

imported.

Importing non-inflected files

If the program you are using cannot export records in a form that *Idealist* can immediately understand (*i.e.* Natural format, *Idealist* database format, dBASE files or tab or comma separated files), you need to decide whether the program produces inflected or non-inflected files, and create a suitable import format file.

Like inflected files, non-inflected files are a category of textual file formats, rather than a specific format. Unlike inflected files, the only way to identify to which field a piece of text belongs is by its position. For example, a mailing list stored on a word processor might begin:

```
J Wilson| AGRP| 10 West St| Ukiah|CA 16752
M Yip| Trip Inc.| 9 Eastern Blvd| Napa|CA 67562
B Wise|Triplay Corp.| 17 Coney St| Napa| CA 67453
```

In this example, fields are separated by vertical bars, but the name of each field is not explicitly stated in the file. To tell *Idealist* how to break up the records (*i.e.* which character separates each field), and into which fields to import the various pieces of text, you need to create an import format file.

Hint If the fields are all separated either by tabs or commas and the records are separated by line breaks, you can use the tab or comma separated files import method.

Non-inflected import format file

The format of a non-inflected format file is:

```
:RecordType
Field=RegularExpression
...
Field=RegularExpression
```

Each regular expression defines the text pattern that occurs at the end of each field in the source file. The regular expression for the last field defines the text pattern that occurs at the end of each record.

For example, given a source file containing:

```
To Have and Have Not,B&W
The Big Sleep,B&W
```

...a corresponding noninflected import format file might look like:

```
:Movie
Title=,
Mode=$
```

General points

If a field named in the import format file does not exist in the record, *Idealist* will attempt to append a field of that type to the record.

If the field name is omitted entirely, the text for the field will be matched but not imported. This is useful for discarding fields in the source file that are not required in the *Idealist* database.

Field names may either be given on their own (*e.g.* Text) or with a suffix (*e.g.* Text[3] - referring to the third instance of a Text field in this record).

Special fields

: or @Type - Import record type (change the type of the record as you import it!)
! or @Name - Import record name

Importing more than one record type

An import format file can only contain a description of how to import one record type. However, since the record type can be imported (into the pseudo-field ':') a record may be changed to a different type as it is imported.

Importing DBF files

DBF files are dBASE database files. Many other databases can also export records in this format. *Idealist* recognizes dBASE II, III, III+ and IV format files, including their associated memo files (.DBT).

Before you can import a DBF file, you need to define a record type whose fields match those of the DBF file. The field names in the *Idealist* record type must be exactly the same as the equivalent fields in the DBF file (although the case of the field names does not have to match exactly) and date and number fields in the DBF file must be imported into fields defined as Date and Number in the *Idealist* record type. The order of the fields can differ between the *Idealist* and DBF records. This record type should be selected in the Record type box in the Import dialog box before you click the OK button. If desired, the field and record definitions can be imported from the DBF file using the File tab in the Define Fields and Define Records dialog boxes.

Note:

o If the DBF file contains fields that you do not want to import, omit the fields from the *Idealist* record type.

o If you add extra fields in the *Idealist* record type which do not appear in the DBF file, these fields will appear as empty fields in the imported *Idealist* records.

o Once imported, Date and Number fields will follow the date and number formats specified in the International section of Windows Control Panel.

You do not need to create an import format file to import records from a file in this format.

What happens when you import from a dBASE file to an *Idealist* database

1. The DBF (and possibly DBT) file is opened and checked for a valid format.
2. The field definitions are read from the DBF file.
3. The records from the DBF file are imported:
    - 3a. Records marked as deleted are ignored.
    - 3b. A blank record of the type specified in the import dialog is created.
    - 3c. Each DBF field is searched for in the *Idealist* record, by name. If it is not found, the field isn't imported. Field names are not case sensitive.
    - 3d. DBF character and memo fields are imported as is. Dates and numbers are parsed into the format specified in the International section of WIN.INI. Logical fields are imported as single characters (Y, N, T, F). Trailing space is stripped from all fields.

Importing files as records

If you have one or more small text files (up to 64,000 characters) which you want to import *Idealist,* you can import a series of files as records. The file will be imported as a single record of the selected type, with the entire contents of the file in the first field. There is no need to create an import format file, as *Idealist* does not need to know anything about the contenst of the file.

Note Make sure that the first field of the selected record type is suitable to hold the contents of the file - usually the first field should be a Text, Line of Text or Wrapped Text field, unless you are sure that the contents of the file are suitable for a field of another type.

Importing tab and comma separated files

These methods allow you to import data from files which have one record per line, with fields separated by a tab character or a comma respectively.

Before you can import these files, you need to define a record type whose fields match those of the data file. The field names of the *Idealist* record type can be any name you choose, but the order of the fields must be exactly the same as the equivalent fields in the file from which you are importing. This record type should be selected in the Record type box in the Import dialog box before you click the OK button.

Note:

o If you add extra fields in the *Idealist* record type, these must be at the end of the record, and will appear as empty fields in the imported *Idealist* records.

o If you define fewer fields in the *Idealist* record type than exist in the tab or comma separated file, the extra fields in the data file are ignored.

You do not need to create an import format file to import records from a file in these formats.

Importing simple

This method allows you to import data from a text file which has records separated by blank lines. All the text between blank lines is imported into the first field of the chosen record type.

You do not need to create an import format file to import records from a file in this format.

Laying out your data on the screen

By default, the on-screen appearance of fields is controlled by *Idealist*, which arranges them one below the other, and places the names of the fields in a margin on the left of the screen.

Layouts provide a way to explicitly arrange fields, and other information, on the screen. For example, fields can be placed side-by-side, and the size and position of each field can be precisely controlled.

Layouts are stored in files on the disk. Each layout file contains one layout type for each record type. For example, if your database contains Book, Article and Chapter record types, one layout file would contain a separate layout for each of these record types. The technically-minded reader may care to read the section on the contents of layout files for more information.

Whenever a database is opened, *Idealist* attempts to open a layout with the same name as the database. For example, when you open a database called MACBETH, *Idealist* will automatically open and use a layout stored in a file called MACBETH.LAY, if such a file exists.

Each layout type is composed of one or more items. An item typically defines the size and position of a field in the layout, but it might also be one of:

- A string of characters, that can be used as a label.
- A button, like the buttons in the button bar, that can be used to carry out commands.
- A graphic image, that can be used for decoration.
- A filled or hollow rectangular shape, that can be used for decoration.

A layout is always edited with *Idealist* in Source mode. When in Result mode, the layout becomes "fixed" and cannot be altered.

When editing a layout, a floating button box appears toward the right of the screen. The buttons in this button box are used for inserting and deleting items, toggling a snap-to grid on and off, moving items and changing the properties of each layout item.

Printing

(under construction)

Searching

When you search an *Idealist* database, you always get back a "hit list" - a list of records that are matched by the search.

Search dialogs

Idealist features three types of search dialog that can be used to give searching commands. These are:

1. The command line search dialog. Into this dialog, you type search commands in *Idealist*'s native search command syntax, for example 'Name=Edward Vedder' or '(Text)==Seattle' or 'PostedDate=21/03/93 [to] 28/03/93'. This is the most powerful way to search a database, since all the searching features are available. Regrettably, it can also be the hardest to use, since you have to learn and remember at least some of the *Idealist* search syntax.

2. The Search-O-Matic dialog. This guides you through the process of creating a search command. The Search-O-Matic dialog is quite easy to use, and suited to the casual user. However, it cannot be used to perform some of the more esoteric searches, notably proximity and range searches.

3. Search-U-Like dialogs. These are customisable search dialogs that display a list of fields and let you type in the text to find in each field. They are set up by an administrator and are usually intended for a specific type of search on a specific type of data.

Keyboard commands

F1	Help
CTRL+F1	Display a field's comment
F2	Create new record
SHIFT+F2	Create new record of same type as current
F3	Expand glossary entry
SHIFT+F3	Change case of selection
F4	Repeat last command
CTRL+F4	Close current database
ALT+F4	Quit *Idealist*
SHIFT+F4	Repeat last find/replace
F5	Find (create new hit list)
SHIFT+F5	Find all records
CTRL+F5	Move cursor to next hit word
F6	Widen hit list
CTRL+F6	Next database window
F7	Narrow hit list
F8	Exclude records from hit list
F9	Toggle source/result mode
F10	Menu
ALT+F10	Type in a command
F11	Execute selection
CTRL+F11	Auto dial telephone number at cursor
F12	Explicitly save current record
CTRL+F12	Open a database
SHIFT+F12	Explicitly save current record
CTRL+SHIFT+F12	Print
GRAY PLUS	Next record
SHIFT+GRAY PLUS	Last record
GRAY MINUS	Previous record
SHIFT+GRAY MINUS	First record
GRAY MULTIPLY	Toggle mark on current record
CTRL+SHIFT+SPACE	Non-breaking space Useful in wrapped text and name fields.
INS	Insert: Function into Calculated field Current date into Date field Current time into Time field Graphic file into Graphic field Vocabulary item(s) into Vocabulary field Pick list item into Pick list field Command into an embedded button
CTRL+DEL	Delete word right
CTRL+B	Toggle bold (rich text field only)
CTRL+C, CTRL+INS	Copy selection to clipboard
CTRL+I	Toggle italic (rich text field only)
CRTL+O	Display View Options dialog
CTRL+U	Toggle underline (rich text field only)
CTRL+V, SHIFT+INS	Paste from clipboard

CTRL+X, SHIFT+DEL	Cut selection to clipboard
CTRL+Y	Delete current line
CTRL+Z, ALT+BKSP	Undo changes to record
CTRL+UP ARROW	Move to previous field
CTRL+DOWN ARROW	Move to next field
CTRL+HOME	Move to start of field
CTRL+END	Move to end of field
CTRL+PGUP	Move to beginning of first field in record
CTRL+PGDN	Move to end of last field in record

Limits

The following maximum limits are defined in this version of *Idealist*:

```
Number of open databases            Approx. 50
Number of records in one database       1,024,000
Number of records in one hit list   1,024,000

Size of a regular text field        64,000 characters
Size of a rich text field           250,000 characters
Number of fields per record         160
Size of a record                    160 * 250,000 bytes

Length of field and record names    31 characters
Length of indexable word            66 characters
Number of different words in index  Approx. 40 million

Number of record types              Unlimited
Number of defined field types       64,000

Size of stopword list               Unlimited
Size of synonym list                Unlimited
Size of glossary                    Unlimited

Number of buttons in a button bar       50
```

In databases created after Jan96, dates, times, numbers and currency values have the following ranges:

```
Dates       Jan 1st, 100 AD to Dec 31st, 9999 AD
Times       0:0:0           to 255:59:59
Numbers     (unlimited - double precision floating point number)
Integers    -2147483647     to  2147483647
Currencies  (unlimited - double precision floating point number)
```

Registry

Characters that *Idealist* indexes

Warning! This help topic contains details of the internal workings of the *Idealist* indexing engine. It will only be necessary to understand this information if you wish to customize the characters that *Idealist* indexes.

Idealist does not index characters based on the ANSI value of each character. For example, the letter 'A' is not exposed as 65 to the indexing engine. Instead, the software looks up a value for each character in a Character Value Table (CVT), and passes this value to the indexing engine. A CVT contains one value in the range 0 to 63 for each character. Two or more characters can have the same value, in which case the indexing engine considers them to be identical. For example, the characters 'A', 'a' and 'Ä' could all have a value of 20. If a character does not have a value in a CVT, then it is not indexed.

For example, a fragment of a CVT may look like this:

```
Character  Characters
Value      Having This Value
---------------------------
10         AaÀÁÂÃÄàáâãä
11         Åå
12         Ææ
13         Bb
14         CcÇç
15         Dd
16         Ðð
```

A good way of inspecting a CVT is to examine the contents of the File, Info... dialog. Each line in the 'Indexable characters' list box corresponds to one value; all the characters with the same value will therefore be listed on the same line.

The *Idealist* EXE currently contains three in-built CVTs, one for new databases created on systems running codepage 1252, one for codepage 1250 (aka Windows Latin 2, used in Eastern Europe), and a special 'compatibility' CVT for opening databases created using pre-Jan95 software. This final CVT ensures that old databases can be opened, searched and modified without having to reindex them. If these old databases had extra indexable characters defined, *Idealist* automatically creates entries for them in the compatibility CVT.

When a new database is created, a CVT is copied into the database itself, so that the database can then be safely used on systems that differ in their interpretation of an alphanumeric or indexable character.

The CVT for a database cannot be changed once that database has been created.

Customizing the CVT

The default CVT for new databases (which CVT depends on the codepage in use) can be overwritten by a customized CVT. This is essential for indexing characters not included in the default CVT (for example, the hyphen), and for indexing Scandinavian, Greek or Arabic text. A customized CVT is stored in a text file with the following format:

```
; This is a comment line
;
8   _
9
10  AaÀÁÂÃÄàáâãä
11  Åå
```

```
12  Ææ
13  Bb
14  CcÇç
15  Dd
16  Ðð
```

The file typically contains 64 lines, numbered 0 to 63. Each line can contain a list of characters, separated from the number by blanks or tabs. Lines can be left 'empty', meaning that the value has no corresponding characters. Any lines starting with a semicolon are comment lines. In theory, it is best to spread the characters evenly over the 0 to 63 range of values, as this should give better indexing performance.

You tell *Idealist* to load the customized CVT when creating a new databases via the File, Options dialog.

For example, if you wish to add the hyphen to the list of characters that *Idealist* will index, do the following:
1. Run Notepad. Open the file IDEA1252.CVT, which should be in your IDEALIST directory. This file, which is a textual version of the built-in CVT for codepage 1252, is supplied by Blackwell for use as a template.
2. Decide where you want the hyphen to appear in the list of indexed characters. Do you want it to be indexed before the alphanumeric characters? If so, change the line

```
1
```

so that it reads

```
1    -
```

Otherwise, if you want the hyphen to appear after the alphanumeric characters, change the line

```
60
```

so that it reads

```
60    -
```

3. Save the file, exit Notepad and start *Idealist*.
4. Select File, Options and then press the "Char. Value Table" button. Find the file you have just created, and press OK.
5. Create a new database. From now on, hyphens in this database will be indexed.

Notes to upgrading users

The CVT mechanism was first introduced in the Jan95 software, replacing the 'extra index characters' and 'fold table' features.

Since the 'extra index characters' feature has been retired, you must now create and edit a CVT file to change the characters that *Idealist* will index. Blackwell supply a 'template' CVT file, called IDEA1252.CVT, mirroring the built-in CVT for codepage 1252. This is easily customized using Notepad, and saves the effort of creating a CVT from scratch. (Warning! Be sure to edit this file using a Windows text editor, with a copy of Windows that is using the codepage for which the CVT is designed. Editing it using a DOS text editor will be confusing, since DOS uses a different system of character sets.)

The new CVT for codepage 1252 differentiates between Œ and O, Ø and O, Å and A, Æ and A.

If you previously used an IWIN????.BIN file to customize the character folding table (another feature that has been retired), then you must recreate (reindex or reimport) a new database containing your records.

Software with a version greater than or equal to Jan95 does not support loading IWIN????.BIN files.

Command line

The command line that is used to start *Idealist* normally only contains the folder and name of the *Idealist* executable file, for example:

```
c:\idealist\idealist.exe
```

This command line is given in one of several places, for example:

1. In a Program Manager icon;
2. In the Windows 95 Start menu
3. In a Windows 95 desktop shortcut

You have the option of including three other things on the command line:

1. The name of the *Idealist* application folder, i.e. the folder where *Idealist* looks for all its configuration files. If this is given on the command line, it must follow the EXE path name and be preceded by a @ symbol, for example:

```
c:\idealist\idealist.exe @c:\albert\idealist
```

2. The name of a database to open;
3. An initial search command. This can only be given if it is preceded by the name of a database, for example:

```
c:\idealist\idealist.exe Contacts !Home
```

Customized search dialogs

Customized search dialogs present you with a list of fields, each corresponding to fields in your database. After you type words or numbers into these fields, *Idealist* searches the database and finds the records that contain the specified words or numbers in the specified fields. Any field that you leave empty is not involved in the search.

This type of search dialog is sometimes known as "query by example". *Idealist* refers to them as 'Search-U-Like' or 'Sulk' dialogs.

The selection and arrangement of fields in the dialog box is determined by a system administrator, not by *Idealist*.

Dialog buttons

Find Creates a new hit list containing records that match those specified by the dialog.
Widen Extends an existing hit list to also include those records that match those specified by the dialog.
Narrow Contacts an existing hit list so that it only contains records that match those specified by the dialog.
Exclude Drops records from the hit list that match those specified by the dialog.
Index Presents a pop-up index browser for the field where the cursor is currently situated. If the field is defined as a vocabulary or pick list field, then a list of the vocabulary or picklist items is presented instead.
Save Saves the contents of all the fields currently in the dialog.
Recall Recalls a previously saved dialog.
Recall Last Recalls the field contents that were used in the previous search.

Examples

Given a text field in a Search-U-Like dialog called, say, 'Notes', you could type the following into that field in the dialog:

arrow - would find records containing the word arrow in a Notes field.
arrow* - would find records containing a word starting with arrow in a Notes field.
***arrow** - would find records containing a word ending with arrow in a Notes field.
"arrow" - would find records containing a word sounding like arrow in a Notes field.
arrow head - would find records containing the phrase arrow head in a Notes field.
arrow [5] head - would find records containing the words arrow and head within five words of each other in a Notes field.
arrow [.] head - would find records containing the words arrow and head within the same sentence in a Notes field.
arrow [NEAR] head - would find records containing the words arrow and head within the same paragraph in a Notes field.

Given a number field called, say, "Expected", you could type the following into that field in the dialog:

2143 - would find records containing 2143 in an Expected field.
>2143 - would find records containing a number greater than 2143 in an Expected field.
<2143 - would find records containing a number less than 2143 in an Expected field.
2143 [TO] 2200 - would find records containing a number between 2143 and 2200 (inclusive) in an Expected field.

See the help topic on the Sulk() command for more details.

Directory structure

Idealist knows about two folders/directories:

1. The one the current database is stored in. This directory is searched for all database-specific information, such as button bars, local field definitions, field vocabularies, field pick lists and field comments.
2. The "application" directory. This is determined when *Idealist* starts, and is either:
    a) the directory which contains the *Idealist* executable, or,
    b) the directory named on the command line.

This application directory (named in the Help, About, More dialog) is where *Idealist* searches for all global configuration information, such as global field and record definitions, synonym lists, glossaries and the help file.

Local versus global information

Global	Local
IDEALIST.DEF	*database*.DEF
IDEALIST.BAR	*database*.BAR
IDEALIST.CMD	*database*.CMD

Name formatting

Idealist includes a number of sample formats for Name fields, which are displayed when you click the Name button in View, Options. However, you may wish to use a different format. *Idealist* allows you to create custom formats in the file NAME.LST.

1 Run Windows Notepad from *Idealist* as follows:
a Type **NOTEPAD** in a field.
b Move the insertion point so that it appears within the word **NOTEPAD**.
c Press F11.
2 Click on File in the menu bar, then click on Open.
3 In the File Name box, type **C:\IDEALIST\NAME.LST**.
4 You see the NAME.LST file, which looks something like this:

```
Smith Jones and Brown
Harvard=Smith, S.S.; Jones, J.J and Brown, B.B.
Vancouver=Smith SS, Jones JJ, Brown BB
Smith, S.S., Jones, J.J. & Brown, B.B.
SMITH JONES BROWN
SS Smith JJ Jones and BB Brown
SMITH-S-S JONES-J-J BROWN-B-B
```

5 Using the same three example names (SS Smith, JJ Jones and BB Brown), type an extra line at the end of the file. Add any extra text and punctuation required. To force the surnames to be displayed in capitals, use **SMITH**, **JONES** and **BROWN**; to display them in mixed upper and lower case, type **Smith**, **Jones** and **Brown**. Similarly, use uppercase initials to force the initials to be displayed in capitals. For example:

```
SMITH, S.S., JONES, J.J. and BROWN, B.B.
```

would force all surnames and initials to be displayed in uppercase.
6 If you want to give the format a name (for example, so that you can remember which journal uses the format), insert the name at the beginning of the line, followed by an equals sign, *e.g.*:

```
TradEng=SMITH, S.S., JONES, J.J. and BROWN, B.B.
```

7 Click on File and select Exit from the menu.
8 When you are prompted to save current changes, click the Yes button.

When you click the Name button in *Idealist's* View Options dialog box, the new format will appear at the bottom of the list.

When parsing the contents of a Name field, *Idealist* ignores the following words:

```
et al
Ed
(Ed)
Eds
(Eds)
Prof
Dr
Jnr
and
```

These terms will not be taken to be surnames or initials. The terms to be ignored in Name fields are stored in a file called IGNORE.LST, which you can edit in the same way as NAME.LST.

Note Avoid adding items which could be initials or surnames. For example, adding **`M`** (for Monsieur) to IGNORE.LST will cause *Idealist* to ignore all instances of M as an initial.
The word **`and`** illustrates a useful way to combine the use of IGNORE.LST and NAME.LST. If you specify **`and`** in IGNORE.LST and use the equivalent in another language (such as *und*, *et* or *y*) in a format in NAME.LST, then when you type:

```
H Gambon and S Jackson
```

in a Name field, it could change the field to read:

```
GAMBON H und JACKSON S
```

Networking

The *Idealist* program allows you to open databases in "multi-user" mode; allowing simultaneous read/write access to shared databases amongst a network of users. Multi-user access is enabled by checking the "Multi-user" checkbox in the File, Open dialog (or by setting a flag in the FileOpen() command).

Idealist implements a "peer to peer" network model, giving each workstation equal status when sharing a database. If you wish, one workstation can be used as a file server, although this distinction would be arbitrary.

The *Idealist* program does not contain any network-specific features, and so should work with any PC network.

File classification

It is convenient to classify *Idealist* files into one of three groups:

1. Database files

Files that "belong" to a database and which are shared amongst users. For a database called SAMPLE, these files would be:

```
SAMPLE.TEX   the main database file
SAMPLE.DIR   index directory
SAMPLE.OCC   occurrence lists
SAMPLE.ROT   record offset table
SAMPLE.TRM   index
SAMPLE.DEF   local field & record definitions (optional)
*.VCB        local vocabulary files (optional)
*.PCK        local pick list files (optional)
*.CMT        local field comment files (optional)
SAMPLE.LAY   record layout definitions (optional)
SAMPLE.CMD   command file (optional)
SAMPLE.BAR   button bar (optional)
```

These should be placed in a shared area of the network. *Idealist* also creates a network control file for each database in the same directory (in this example it would be called SAMPLE.NET). This file is only required while the database is being accessed and need not be backed up.

2. View files

Files which pertain to a particular user's view of a database. These include saved hit lists and import, export, print and layout format files. These should be kept in a separate directory for each user. Example of these files are:

```
IDEALIST.BAR  ; global button bar configuration
IDEALIST.DEF  ; global field and record definitions
GLOSSARY.LST  ; edit glossary
SYNONYM.LST   ; search synonyms
REPORT.EXP    ; export format file
SAMPLE.LAY    ; view layout
```

3. Executables

General *Idealist* files, such as the executables, DLLs, help files and so on. *Idealist* will load and operate faster if copies of each of these are kept on each workstation.

File organization

Given a network of users who all wish to have access to a shared database, one workstation would be nominated to hold a copy of the database itself. This would typically be the machine with the largest hard disk - we'll call the workstation "Jumbo". In a directory of Jumbo's hard disk, put the database files. When run, *Idealist* creates a .NET file in this directory.

The same workstation would probably be used to store configuration files shared by all users.

Each workstation user should then be given their own "view" directory on their hard disk, containing all the files that relate to their view of that database, and which are not shared by other users. These would typically be saved hit lists, import, export, print and layout format files, but could also include personalized copies of configuration files.

The executable and help files are shared amongst all users, but copies of these are also typically stored on each separate workstation.

The default file directory should be set on each workstation to point at the directory in which the configuration files are stored. This can be done by specifying the directory on the *Idealist* command line. Then, you can create a Program Manager icon on each workstation that sets the working directory to the directory containing that user's view of the database and runs the *Idealist* executable.

Operation

The (File, Reindex), (Hit List, Deduplicate) and (Record, Undelete) options are not available when a database is opened in multi-user mode. Each of these commands requires exclusive read-write access to the database. If you need any of these commands, open the file in "single user" mode (*i.e.* don't check the "multi-user" checkbox in the File, Open dialog).

Operation is significantly more efficient when a database is opened in "single user" mode than in "multi-user" mode. When in single user mode, access will be faster and the database will be more compact. Only open a database in multi-user mode if you really have to.

The normal *Idealist* executable is quite happy to open databases in multi-user mode. However, it defaults to opening databases in "single user" mode. If this is a problem, a special version of the software exists which defaults to opening databases in "multi-user" mode.

Editing records

When opened in multi-user mode, the first user to look at a particular record gets exclusive read/write access to that record - the user can read and edit the record. If other users look at that record while the first is still viewing it, they will be denied write access to the record - they will be allowed to look at it, but not edit it. The word "Locked" will also appear in the status bar. They will not be able to edit the record until the first user leaves the record to look at another.

While an edited record is being updated, or a group of records are being imported, the index is locked, denying all other users access to it. The message "Index busy. Retry?" appears if you try to access the index while it is locked.

> Click the Yes button to retry the search or indexing operation.
> Click the No button to abort the operation.
> Click the Cancel button to abandon the operation.

Searching

When the index is locked, for example by another user importing a group of records or viewing the index, searching cannot take place; the message described above appears.

Deleting records

If you create a hit list, and a record in that hit list is then deleted by another user, the message "Record Unavailable" will appear in the status bar if you try to look at that record again as part of the hit list. (Of course, were you to perform the search again, the record would not be found.)

The "Record Unavailable" message also appears if you backtrack to a hit list and move to a record that has been deleted.

Note The (Record Undelete) and (Hit List, Deduplicate) commands are unavailable when opening a database in multi-user mode.

How it works (for advanced users)

The *database*.NET file contains a 1-byte header and a byte for every record in the database.

Record locking Whenever a user views a record in a shared database, their software attempts to lock the byte in *database*.NET corresponding to that record (the byte at 1 + record number). If the lock succeeds, that user gets read/write access to the record, and their software unlocks the byte when they have finished with the record. If the lock fails (because another user has already locked the byte) then the user gets read-only access to the record.

Index locking To ensure that only one user gets access to the index at once, the network software keeps the index files (OCC, TRM, DIR) closed. Whenever a user needs access to the index (to add/modify a record, or to search) then the software tries to lock the first byte in *database*.NET. If this succeeds, the software opens the index files, performs the operation and finally closes the files. If the lock fails (because another user has already locked the byte) then the user is informed that the index is busy, and invited to retry.

This very simple scheme has proved highly portable and reliable, and gives acceptable results among small workgroups doing searches, record updates and manual entry of records. It is recommended that databases are opened in "single user" mode when importing or deleting large numbers of records, as this software will be both faster and produce more compact databases. This is because in "single user" mode the .EXE, knowing that it has exclusive access to a database, can cache the database in memory and reuse deleted file space.

The file sharing mode used to open the database files varies as follows:

Single user mode
The database files (.TEX, .ROT, .OCC, .TRM, .DIR) are opened in read-write mode using the 'share deny write' flag. This gives the software exclusive write access to these files, but allows other instances of *Idealist* to open the database in read-only mode.

Multi user mode
The database files are opened in read-write mode using the 'share deny none' flag. This allows other instances of *Idealist* to open the database in either multi-user or readonly mode.

Readonly mode
The database files are opened in readonly mode with the 'share deny none' flag.

Idealist does not use the 'compatibility mode' flag.

Security

Access to each of your *Idealist* databases can be controlled by enabling security for each database. Do this by selecting File, Security.

Protecting *Idealist* databases is a three-step process:

1. Organize your users into groups, then define those groups and each group's access privileges.
2. Give each user a name, assign each user to a group and (optionally) give each user a password.
3. Tell users what their passwords and access privileges are.

Stopwords

Idealist will not index common words like "the", or "and". These are called 'stopwords'.

By default, a list of stop words is stored in the text file STOPWORD.LST, which you can edit with Windows Notepad. The actual name and location of this file can be specified via the File, New and File, Reindex option dialog.

Whenever you create a new *Idealist* database, a copy of the words in the stopword list file is stored in the database; thereafter, the database does not need to refer to STOPWORD.LST.

Synonyms

Because *Idealist* searches require exact matches, sometimes a search on a single term may not find all the relevant records. For instance, if you searched for **`BBC`**, you would not find records that referred to **`B.B.C.`** or the **`British Broadcasting Corporation`**. This problem is likely to occur:

If the records have come from several sources.
If the records have been entered by more than one person.
If the search term is an abbreviation or other shortened form.
If the search term has several common alternative forms.

To overcome problems of this kind, you can set up lists of synonyms of commonly used search terms. Each list of synonyms consists of a **trigger term** and a series of one or more synonyms.

IDEALIST.BAR/*database*.BAR

Each line in this file describes one button bar button. A blank line represents a blank button.

Each line contains three fields, separated by ANSI 127 characters.

1. The first field is numeric, and contains the identifier of the button to be drawn.
2. The second field contains the command or commands that will be run when the button is pressed.
3. The third field contains a description of the button, that appears in the tooltip and in the status bar. For example:

```
126□Find□Find
101□Prev□Previous record
116□SearchFindAllRecords()□Find all records
102□ViewNext()□Next record
154□ViewMode□Toggle source/result mode
151□Overview□Overview
218□Sort□Sort
144□EditBold□Bold
145□EditItalic□Italic
146□EditUnderline□Underline
```

IDEALIST.CMD/*database*.CMD

This text file contains a list of commands, separated by blanks, tabs or newlines.

The commands in IDEALIST.CMD are run when the *Idealist* application starts. IDEALIST.CMD is expected to be in the application directory.

The commands in *database*.CMD are run when *database* is opened. *Database*.CMD is expected to be in the same directory as the rest of the database files.

.CMT

A *field*.CMT file contains descriptive text for a field. The text is displayed when the user presses CTRL+F1 when the cursor in the field.

.CVT

IDEALIST.DEF/*database*.DEF

A DEF file contains field and record definitions. IDEALIST.DEF, found in the application directory, contains global definitions: *database*.DEF contains definitions local to *database*.

The file contains two sections:

[records]

Contains the definition of one record per line. The first name on each line is the name of the record type, the remaining names are the fields in that record. For example:

```
Contact Surname FirstName Address Notes
```

[fields]

Each line contains the definition of one field. The first name on each line is the name of the field type, and the remaining items are a combination of:

Bold The font will be bold.
Color Specifies the color of the font. The color is given as a COLORREF number.
Comment Indicates that this field has a comment, stored in *field*.CMT.
Default Specifies the default contents of the field.
Encrypted The contents of the field will be encrypted when stored on disk. This prevents someone from viewing the contents of a database with any other software.
Font Specifies the face name of the font.
Format Specifies the template for the field.
Hidden The field will not be displayed.
InitialCaps The first letter of each world in the field will be capitalized.
Italic The field font will be italic.
Lines Specifies the maximum number of lines that will be visible without scrolling when the field is displayed without a layout.
Lowercase The contents of the field will be forced to lowercase.
Mandatory The field must be filled in before saving.
MaxChars Specifies the maximum number of characters that can be entered into the field.
PickList The field has a pick list, stored in *field*.PCK.
Points The height of the field font, in points.
Readonly The field will be readonly.
RTL The field contains right-to-left text. Only useful for Notepad fields containing Arabic text.
Spellcheck The contents of the field will be spellchecked before saving.
Strikeout The font will appear stuck-out.
TabSize Specifies the tab width used in the field, in average character widths.
Type Specifies the data type of the field. This must be one of:

- "Calculated"
- "Currency"
- "Date"
- "Graphic"
- "Identifier"
- "Integer"
- "Line of note"
- "Line of rich"
- "Line of text"
- "Name"
- "Notepad"
- "Number"

"Rich"
"Text"
"Time"
"Wrapped note"
"Wrapped rich"
"Wrapped text"

Underline The field font will be underlined.
Unindexed The field will not be indexed.
Uppercase The contents of the field will be converted to uppercase before saving.
Vocabulary The field has a controlled vocabulary, stored in *field*.VCB.
WORM The field is write-once, read-many. After you type something into the field and save it, the field becomes readonly.

.EXP

.IMP

There are two types of import format files:

Inflected (tagged)

Non-inflected

.LAY

Layout files contain an ANSI description of how fields, buttons and static items (text, graphics and shapes) are laid out on the screen.

Each layout file is divided into sections, each section describing the layout for one record type. Each record type sections begins with a line consisting of a colon in column one, followed by the record type, for example:

```
:Address
```

Each line within each section contains the placement and properties of one item. Each line has the format:

string x y width height [*optional attributes*]

The type of each item is interpreted from the contents of *string*:

1. If *string* is enclosed in curly braces, then this is a field item. For example:

```
{Text}
{Text[3]}
```

2. If the first character of *string* is ANSI 17, this is a button. The title of the button is terminated by an ANSI 127. The commands in the button are terminated by an ANSI 16. For example:

```
<17>Press Me<127>Next() Overview("Text")<16>
```

3. If *string* consists of a single character, ANSI 128, then this item is a frame.
4. If *string* consists of a single character, ANSI 129, then this item is a filled rectangle.
5. If *string* corresponds to the filename of a graphic image, then this is a graphic item.
6. Otherwise, this item is a piece of static text.

The four integer constants that follow - *x*, *y*, *width* and *height* - are measured in pixels.

The rest of the line contains zero or more optional attributes, taken from the following list:

Autosize
This keyword is no longer supported.

Bold
This item uses a bold font.

Border
A shaded gray border is drawn around the item.

Center
Centre
Static text is centered within it's frame.

Color
Colour
The color to used for the item's font. The value is a 24-bit RGB value, expressed as (RED<<16)+(GREEN<<8)+(BLUE). Each color component has the range 0..255. Thus, white becomes

(255<<16)+(255<<8)+(255) = 16777215, and green is (0<<16)+(255<<8)+(0) = 65280. For example, **Color=255**.

Font
The face name for the item's font. For example, **Font="Tms New Roman"**.

Italic
This item uses an italic font.

Left
Static text is left-justified within it's frame.

Points
The point size of the item's font. For example, **Points=32**.

Right
Static text is right-justified within it's frame.

StrikeOut
This item uses a struck out font.

Underline
This item uses an underlined font.

GLOSSARY.LST

STOPWORD.LST

This text file contains a list of the words that Idealist will ignore when it indexes a record.

Within the file, the words can be in any order, although it is convenient to sort the words into alphabetical order. The words should be separated by spaces, tabs ot newlines.

SYNONYM.LST

fieldname.PCK

This text file contains a list of the pre-defined contents a field. Each line in the file contains one pick list item, which may be a single word or a phrase. For example, the content of COLOR.VCB might be:

```
black
dark green
grey
light green
red
white
```

.SUL

fieldname.VCB

This text file contains a list of the defined vocabulary for a field. Each line in the file contains one vocabulary item, which may be a single word or a phrase. For example, the content of COLORS.VCB might be:

```
black
dark green
grey
light green
red
white
```