Contents

Chapter 1

Features

1.1 4GL

Informix 4GL was co-designed by Roger Sippl (founder of Informix) and Chris Maloney. They combined the elements of Perform (the screen package designed by Betty Chang), Ace (the report writer written by Bill Hedge), and the SQL engine written by Roy Harrington into a Pascal-like language which Informix released in 1986 - the same year that the first ANSI standard for SQL was promulgated.

Informix 4GL complied with the SQL86 standard. I4GL was phenominally successful in the marketplace. More applications were written in I4GL in the 1990s than in any other language.

1.2 Aubit 4GL

Aubit 4GL is a free opensource work-alike for Informix 4GL. The project was started by Mike Aubury of Aubit Computing Ltd, who continues to contribute to it. A number of other notable contributors include Andrej Falout, John O'Gorman, Sergio Ferreira, Walter Haslbeck.

Where Informix 4GL was locked into working only with Informix's own database engines: SE and IDS, Aubit 4GL can work with any SQL compliant engine. Currently supported engines are Informix SE and IDS, PostgreSQL, [SAPDB?], SQLite, and using ODBC (unixODBC or iODBC on Linux/Unix) any other engine for which ODBC interfaces exist.

1.3 Aubit4GL Benefits

1.3.1 GNU, GPL, OpenSource

Aubit4GL is free and opensource. It will cost you nothing, but there are much more important implications that this, in our view. Its future does not depend on anyone but you. To find out more about implications of this feature, please see `http://www.opensource.org` and `http://www.gnu.org`.

1.3.2 Commercially Supported

Commercial support is available from Aubit Computing Ltd if you want it. Aubit Computing Ltd is Mike Aubury's company.

This will guarantee you can use Aubit4GL in business-critical situations with confidence, and bring together the best of both worlds. To learn more, visit `http://www.aubit.com`.

Support is now available for Portuguese and Spanish speakers from Moredata (URL?) and for German speakers from Ventas.

1.3.3 Productive

Based on a robust, mature, stable, efficient, and productive language, x4GL is dedicated to writing business-related, database oriented applications, and this is what it does, in our opinion, best in the world.

It is easy to learn, implement, and maintain. And most of all, it is at least 3 times more productive in all aspect of the software lifecycle than third generation languages like C, and at least twice as productive as the best component development environments.

1.3.4 Fast

It's FAST! Full n tier deployment, native C code generation compiled by optimized C compilers bring together the advantages of a high-level, human-like development language, and low-level machine-code runtime execution performance. Not to mention that you can interpolate C code directly into 4GL code!

1.3.5 Compatible

Aubit4GL is compatible with a number of commercial products giving you the freedom to mix and match features and environments based on any particular situation. You will never again be locked into one compiler, one company, or one database. You can develop with commercial products, deploy with Aubit 4GL, and save on runtime licensing, at the same time gaining the speed of a C compiled runtime. Or you can use 4GL Wizard functionality and templates in development, and deploy using a commercial runtime that supports client side functionality that is not present in Aubit 4GL at the moment.

1.3.6 Engine Independent

Database, OS, platform, and user interface independent ODBC means that choosing a database engine is no longer an issue.

You can develop and deploy wherever a GCC compiler is available with a single recompile. And because of full n-tier support, you can use CUI, GUI and Web interfaces from the same code, and the same compiler program, at the same time, just by setting environment variables. Informix 4GL already has a big developer base, and a large existing applications base.

This is not a new language, just a new implementation of a mature and successful language. So you will not need to look hard to find developers for your projects. And since 4GL is English-like in syntax, programmers with experience in any language will be productive in just a few days. On top of that, you will not need to look far to find commercial, tried and tested applications in any field of business oriented database applications.

1.4 Aubit4GL Extensions

A4GL fully supports the features and syntax of Informix 4GL, but we have extended the language with many enhancements to increase the productivity of the 4GL developer. These enhancements are fully described in the Aubit4GL Extensions chapter of this manual.

Chapter 2

Installation - Quick Start

2.1 A Quick Start ?

If you are ready to just jump right in, the following sections highlight the most common configurations.

2.1.1 Downloading and installing

Installation should be possible on most Linux distributions, and with some tweaking on most Unix and Windows machines too.

There are several possible sources for download

- MARS Binary releases (`http://www.aubit.com/aubit4gl/`)
- MARS Source releases (`http://www.aubit.com/aubit4gl/src`)
- Nightly builds (`http://aubit4gl.sourceforge.net/files/aubitdownload.htm`)
- CVS (see `http://sourceforge.net/cvs/?group_id=32409`)

2.1.2 Installing Source/CVS

Basically - Follow the normal

```
./configure && make && make install
```

routine..

You may need to specify `--prefix=/home/aubit4gl` or something similar..

2.1.3 Binary

Just `tar xvzf` the tarball (you can optionally make install)

2.2 Next steps

You can skip this if you read the rest of this document - otherwise please read on - it will save a lot of confusion later!

Take a moment to note the libraries created in the lib directory.

E.g. (This is my list - yours will be different!) :

- `libaubit4gl.so`
- `libA4GL_file.so libA4GL_HTML.so libA4GL_string.so`
- `libDATA_menu_list.so libDATA_module.so libDATA_report.so libDATA_struc`
- `libESQL_INFORMIX.so libESQL_POSTGRES.so`
- `libEXDTYPE_mpz.so`
- `libEXREPORT_NOPDF.so libEXREPORT_PDF.so`
- `libFORM_GENERIC.so libFORM_NOFORM.so libFORM_XDR.so`
- `libHELP_std.so`
- `libLEX_C.so libLEX_CS.so libLEX_EC.so libLEX_PERL.so`
- `libLOGREP_CSV.so`
- `libLOGREP_TXT.so`
- `libLOGREP_PROC_CSV.so libLOGREP_PROC_TXT.so`
- `libMENU_NOMENU.so`
- `libMSG_NATIVE.so`
- `libPACKER_MEMPACKED.so libPACKER_PACKED.so libPACKER_PERL.so libPACKER_XDR.so libPACKER_XML.so`
- `libRPC_NORPC.so libRPC_XDR.so`
- `libSQL_esql.so libSQL_esql_s.so libSQL_FILESCHEMA.so libSQL_ifxodbc.so libSQL_nosql.so libSQL_sqlite.so libSQL_sqliteS.so libSQL_unixodbc.so`
- `libUI_CONSOLE.so libUI_HL_TUIN.so libUI_HL_TUI.so libUI_TUI.so libUI_TUI_s.so`

- libXDRPACKER_menu_list.so libXDRPACKER_module.so libXDRPACKER_report.s
  libXDRPACKER_struct_form.so

The correct selection of these libraries is pretty critical to the operation of Aubit4GL, because everything is so highly configurable.

You'll notice that most of them have a libXXX_YYY.so format (except libaubit4gl.so) so for example :

libSQL_esql.so XXX=SQL YYY=esql

libUI_HL_TUI.so XXX=UI YYY=HL_TUI

The XXX represents the module type, the YYY the module name. Although Aubit4GL is distributed in a form which will be mostly Informix4GL compatible - you will almost certainly need to adjust some of these settings.

2.2.1 Module types

2.2.1.1 A4GL

eg : libA4GL_file.so libA4GL_HTML.so libA4GL_string.so

These are miscellaneous extra libraries.

2.2.1.2 DATA

eg : libDATA_menu_list.so libDATA_module.so libDATA_report.so libDATA_struct

These are internal libraries for reading data files.

2.2.1.3 ESQL

eg : libESQL_INFORMIX.so libESQL_POSTGRES.so

These are helper libraries used when A4GL_LEXTYPE=EC. The library used is taken from the A4GL_LEXDIALECT variable. This library is used to copy between native types and aubit types (eg for decimals, dates etc)

Not used when A4GL_LEXTYPE=C

2.2.1.4 EXDTYPE

eg : libEXDTYPE_mpz.so

Example extended datatype library (implements the GNU mpz datatype).

2.2.1.5 EXREPORT

eg : `libEXREPORT_NOPDF.so libEXREPORT_PDF.so`

Extended report handling. `libEXREPORT_PDF.so` relies on having pdflib installed. It will not be generated otherwise... PDF reports are experimental.

2.2.1.6 FORM

eg : `libFORM_GENERIC.so libFORM_NOFORM.so libFORM_XDR.so`

This is used to read, write, and process a form file. The library is specified by the `A4GL_FORMTYPE` variable. e.g.: `A4GL_FORMTYPE=GENERIC`

If you have `libFORM_XDR.so` - that is probably the best one to use, so

```
$ export A4GL_FORMTYPE=XDR
```

If you don't have `libFORM_XDR.so`, you'll need to use the GENERIC packers

```
$ export A4GL_FORMTYPE=GENERIC
```

You will then also need to specify the GENERIC packer by setting `A4GL_PACKER` (see PACKER)...

2.2.1.7 HELP

eg : `libHELP_std.so`

Always set to std - can be ignored

2.2.1.8 LEX

eg : `libLEX_C.so libLEX_CS.so libLEX_EC.so libLEX_PERL.so`

Specifies the output format - currently only C and EC are supported.

For C generation, calls are made to internal SQL functions within the library specified by `A4GL_SQLTYPE` (see SQL)

For EC generation, a .ec file is generated which should be compiled used native database tools (like esql for informix and ecpg for postgres). If you can use EC generation - use it, performance will be better...

2.2.1.9 LOGREP

eg : `libLOGREP_CSV.so libLOGREP_PROC_CSV.so libLOGREPPROC_TXT.so libLOGREP_TXT.so`

Logical report handling - ignore for now.

2.2.1.10 MENU

eg : `libMENU_NOMENU.so`

GUI Menu handling - obsoleted (probably).

2.2.1.11 MSG

eg : `libMSG_NATIVE.so`

Ignore..

2.2.1.12 PACKER

eg : `libPACKER_MEMPACKED.so libPACKER_PACKED.so libPACKER_PERL.so libPACKER_XDR.so libPACKER_XML.so`

This specifies the packer to use for reading and writing data files. The library is specified via the `A4GL_PACKER` variable. Do not use MEMPACKER and PERL unless you know what you are doing.. PACKED, XML and XDR are all reasonable packers. The packer library is only used when FORMTYPE etc is set to GENERIC.

2.2.1.13 RPC

eg : `libRPC_NORPC.so libRPC_XDR.so`

Specifies which RPC protocol to use - advanced stuff - still experimental.

2.2.1.14 SQL

eg : `libSQL_esql.so libSQL_esql_s.so libSQL_FILESCHEMA.so libSQL_ifxodbc.so libSQL_nosql.so libSQL_sqlite.so libSQL_sqliteS.so libSQL_unixodbc.so`

This is probably the most important setting, specified through SQLTYPE - this determines how Aubit is going to talk to the database. There are two distinct times that this is done:

- At compile time
- At runtime

2.2.1.14.1 EC generation

2.2.1.14.1.1 COMPILE TIME the library controls detecting datatypes for LIKE and RECORD LIKE etc.

2.2.1.14.1.2 RUN TIME The runtime usage is limited to handling LOAD and UNLOAD statements. For postgreSQL, setting `A4GL_ESQL_UNLOAD=Y` will call the ecpg builtin load and unload statements so this library doesn't need to be used at all...

Special notes : `A4GL_SQLTYPE=esql` The Informix ESQL/C 'connector', both runtime and compile time. This requires Informix ESQL/C to be installed & configured.

2.2.1.14.2 For C generation

2.2.1.14.2.1 COMPILE TIME the library controls detecting datatypes for LIKE and RECORD LIKE etc,

2.2.1.14.2.2 RUN TIME This handles all I/O with the database.

2.2.1.14.2.3 ODBC unixodbc/ifxodbc - These require that ODBC has been correctly installed and configured. You **must** specify a username and password to connect to most databases this is done using SQLPWD and SQLUID:

```
$ export SQLUID=maubu
$ export SQLPWD=mypasswd
```

2.2.1.15 UI

```
libUI_CONSOLE.so libUI_HL_TUIN.so libUI_HL_TUI.so libUI_TUI.so libUI_TUI_s.
```

This specifies how data will be displayed to the user. This handles all the UI controls (prompt,display, input etc)

CONSOLE - is a simple I/O module which does not use any control codes. Just printfs and fgets etc..

TUI - THIS IS THE ONE YOU SHOULD BE USING

HL_TUI - The next version of TUI, abstracted to help make other HL_.. modules

HL_TUIN - Ignore

HL_GTK - VERY experimental GTK version - don't expect this to work...

2.2.1.16 XDRPACKER

eg : `libXDRPACKER_menu_list.so libXDRPACKER_module.so libXDRPACKER_report.so libXDRPACKER_struct_form.so`

This is a helper module when `FORMTYPE` etc are set to XDR. These contain the actual XDR routines.

2.2.2 Standard settings

If you've got to here - check that you have set :

`A4GL_UI=TUI`

`A4GL_SQLTYPE=esql` (and you have libSQL_esql.so) if you want to connect to an Informix database or

`A4GL_SQLTYPE=unixodbc` - ensure that you have SQLUID and SQLPWD set... or

`A4GL_SQLTYPE=ifxodbc` - ensure that you have SQLUID and SQLPWD set... or

`A4GL_SQLTYPE=pg` (if you have libSQL_pg.so) if you want to connect to a postgres Database

`A4GL_LEXTYPE=EC` for Esql/c generation (you must have Informix esql or Postgres ecpg installed)

`A4GL_LEXDIALECT=INFORMIX` or

`A4GL_LEXDIALECT=POSTGRES`

`A4GL_FORMTYPE=XDR` (if you have it) or

`A4GL_FORMTYPE=GENERIC` if you dont and also set `A4GL_PACKER=PACKED`

2.2.2.1 Finally...

Set up the the Unix environment so the whole thing will actually run!

```
export AUBITDIR=/directory/where/this/all/is
```

```
export PATH=$PATH:$AUBITDIR/bin
```

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$AUBITDIR/lib
```

2.3 Troubleshooting

2.3.1 For Informix

2.3.1.0.1 Get the client SDK Configure the SDK (normally the $INFORMIX/etc/sqlhosts etc)

2.3.1.0.2 Check the SDK Try a simple esql/c program like :

```
main() {
  $whenever error stop;
  $database somedb;
}
```

```
$ esql somefile.ec -o somefile $ ./somefile
```

Compile and run - it if doesn't this is an Informix setup problem - check out why...

Most likely : `.rhosts/hosts.equiv` is not set up properly, user doesn't exist on the remote machine, `/etc/services` isn't set up, `/etc/hosts` isn't set up, or a remote server isn't allowing tcp connections only shared memory ones.

2.3.1.0.3 Set Up Aubit `export AUBITDIR=/aubits/path`

`export A4GL_SQLTYPE=esql` ** check that you have a `$AUBITDIR/lib/libSQL_esql.so` **

`export A4GL_LEXTYPE=EC`

2.3.1.0.4 Try to compile a simple 4gl

```
database somedb
main
  display "Hello World"
end main
```

```
4glpc simple.4gl -o simple.4ae
```

2.3.1.0.5 Try to run it Try to run that `simple.4ae`

2.3.2 For PostgreSQL

2.3.2.0.1 Install postgreSQL If possible - use PostgreSQL with the Informix compatibility patches : (See `http://gborg.postgresql.org/project/pginformix`

Make sure you have the ecpg module.

2.3.2.0.2 Configure postgreSQL and create database if required

2.3.2.0.3 Check you're ecpg setup

```
main() {
  exec sql database somedb;
  printf("Status=%d\n",sqlca.sqlcode);
}
```

Remember to replace `somedb` with the database you created!

```
$ ecpg myprog.ec
```

```
$ gcc myprog.c -o myprog -I/usr/local/pgsql/include -L/usr/local/pgsql/lib
-lecpg
```

(You'll need to adjust /usr/local/pgsql/include etc to your environment)

If Status=0 then you're fine - if not - its an ecpg/postgres problem...

2.3.2.0.4 Set Up Aubit `export AUBITDIR=/aubits/path`

```
export A4GL_SQLTYPE=pg
```

** check that you have a $AUBITDIR/lib/libSQL_pg.so **

if not - go to $AUBITDIR/lib/libsql/postgresql and do a make ** if that doesn't make it - find out why...

```
export A4GL_LEXTYPE=EC
```

```
export A4GL_LEXDIALECT=POSTGRES
```

2.3.2.0.5 Try a 4gl program `simple.4ae:`

```
database somedb
```

```
main
  display "Hello World"
end main
```

```
$ 4glpc simple.4gl -o simple.4ae
```

```
$ ./simple.4ae
```

Chapter 3

Installation - Full

3.1 Platforms

You can install Aubit on recent versions of Linux (e.g Redhat 7 or later, SuSE 8 or later), other Unix systems, or Microsoft Windows 95, NT, 2000 or later with or without Cygwin.

3.1.1 Source or Binary

You can install A4GL from source or binary. Binary is easier if we have created the binary on the same system that you have. Otherwise you need to get the source and compile it.

3.2 Get Source

Create and select a directory like `/opt/aubit` or `/usr/local/aubit` for installing the source. Call this `SRCDIR e.g.`
`export SRCDIR=/opt/aubit`

The Aubit4GL development site is `aubit4gl.sourceforge.net`.

You have 3 ways to get the source:

- From a source tar ball (e.g. aubit4glsrc-n.nn-r.tgz)
- From a source RPM
- From CVS.

3.2.1 Tarball

If you are getting the source from a gzipped tar ball, point a browser at the project home page. Follow the links to the download page and download the file to $SRCDIR then:

```
tar xvfz aubit4glsrc0.47-32.tgz
```

The above assumes that you downloaded the tar file into the AUBITDIR directory. If you have put it somewhere else supply the full path to the tar file e.g.

```
tar xvfz /home/informix/aubit4glsrc0.47-32.tgz
```

On systems where the z option does not work, run gzip and pipe its output into the tar command e.g.:

```
gzip -cd aubit4glsrc0.47-32.tgz | tar xvf -
```

You can examine the contents of a gzipped tarball by using the command:

```
tar tvfz aubit4glsrc0.47-32.tgz
```

3.2.2 SRPM

As for the tar ball, point the browser at the project home page. Follow the links to the download page and down load the file to $SRCDIR then:

```
rpm -Uvh aubit4glsrc0.47-32.rpm
```

The U option is really intended for update but it works equally well for a new install and saves you the bother of learning a separate install syntax. The v option (verbose) will tell you what rpm is doing. The h option will display conforting hash marks on screen for blocks of characters loaded.

You can examine the contents of an rpm file before intalling it, the command is:

```
rpm -qpl aubit4glsrc0.47-32.rpm
```

3.2.3 CVS

You can get the bleeding edge current version of Aubit4GL from the CVS (Concurrent Versions System). To do this:

1. cd $SRCDIR
2. Set an environment variable CVSROOT as follows:
   `export CVSROOT=:pserver:anonymout@cvs.sourceforge.net:/cvsroot/aubit4g`
3. Login to the aubit cvs pserver
   `cvs login`
   When it prompts for a password, just hit RETURN.
4. Checkout the module you want: aubit4glsrc or aubit4gldoc
   `cvs -z3 co aubit4glsrc`

Be warned that from time to time the cvs version may be broken. Development is ongoing and you cannot make an omelet without breaking eggs.

Note: Put the CVSROOT value in a file called (say) AUBITCVSROOT. Then whenever you wish to checkout or update from cvs, you can set CVSROOT using the command
`$export CVSROOT=$(cat AUBITCVSROOT)`

The above works for ksh and bash. If your shell does not accept the $(...) syntax, then use backticks instead:
`$CVSROOT=‘cat AUBITCVSROOT‘ export CVSROOT`

3.3 Prerequisites

3.3.1 C Compiler

The source for Aubit4GL is in the C programming language with some Unix shell scripts. So if you need to install from source, you will need a full GNU GCC or equivalent compiler and GNU make. (These tools come with nearly all distributions of Linux.)

3.3.2 Options

The following software and/or libraries can be exploited by Aubit 4GL. They will be discovered and linked by the autoconfig `configure` script when you install A4GL:

- ODBC manager libraries for database connection: unixODBC, iODBC (or Windows ODBC)
- Native database connections for Informix, PostgreSQL, SAPDB, or SQLite
- PDF library for fancy reports
- Curses Library (for screen DISPLAY, MENU, etc statements)
- GTK+ Library (for GUI frontend)
- Jabber IM library (for instant messaging)
- SUN RPC package (for n-tier applications using Remote Procedure Calls)
- RPC-XML libraries (for communicating with XML format files)
- Perl interpreter
- SWIG libraries (for Perl output instead of C)

Run the configure script to see which of these you have (or don't have). If configure reports something missing when you know you have it, you may have installed it in as unexpected location. Rerun configure
`./configure --help`

To see how to point autoconfig to where you have installed the library. For example, if you installed pdflib in `/usr/john/pdflib` then you can run configure with the command:
`./configure --with-pdflib=/usr/john/pdflib`

On Linux systems the command `rpm -qa` will give you a (huge) list of all software installed using rpm (RedHat Package Manager). To find any rpms related to, say, PDF run the the following:
`rpm -qa | grep -i pdf`

On Linux systems you can find non rpm installed software with the locate command:e.g.
`locate pdf`

3.3.3 Architecture

Aubit4GL uses an abstraction layer for many of its functions. This means that the way Aubit4GL works can be controlled very tightly by the setting of various variables. These variables specify which library functions will be called from the compiler and/or 4GL program and hence affect the following areas:

Variable	Function	Library
A4GL_LEXTYPE	Set generation language	libLEX_
A4GL_LEXDIALECT	Set language dialect (used for ESQL/C generation)	libESQL
A4GL_PDFTYPE	Specify the enhanced report handler	libEXRH
A4GL_HELPTYPE	Specify the help handler	libFORM
A4GL_MENUTYPE	Specify the extended menu handler	libMENU
A4GL_MSGTYPE	Specify the message handler	libMSG_
A4GL_PACKER	'packer' to use saving forms/reports etc (eg. XML)	libPACK
A4GL_RPCTYPE	Specify the Remote Procedure Call handler	libRPC_
A4GL_SQLTYPE	Specify the SQL handler	libSQL_
A4GL_SQLDIALECT	Specify the SQL dialect to use	libSQLP
A4GL_UI	output module to use to display the program	libUI_?'

3.3.4 Database

For most people, the most important component of the Informix 4GL language is its embedded SQL. (At whim, you can put SQL code into any x4GL program). For this feature to work, you must have a database engine which both the A4GL compiler and your 4GL program can connect to. Classical Informix 4GL has a builtin native connection to Informix engines (SE, or IDS 7, or IDS 9).

3.3.4.1 Engines

Informix, Ingres, PostgreSQL, and Sybase engines have their origins in Unix at Berkeley in the 1970s. They share some features which are counter to the ANSI SQL standards which were later defined in 1986.

- Lower Case. By default, they downshift all words before parsing (unless the words are protected by quotes). This is natural for Unix users but is the inverse of the ANSI standard which upshifts all unquoted words. The standard was dominated by mainframe system vendors (IBM DB2, Oracle, SAP, etc).
- Database concept. Each instance of an Informix or PostgreSQL engine can have many databases. In contrast, IBM, Oracle, SAP, etc have only one database per engine instance. The Informix concept of separate databases is implemented on these other systems each as a SCHEMA.
- Outer Joins. These were originally a controversial concept and not defined in the 1986 SQL standard. The 1992 SQL standard added a JOIN clause to SQL SELECT statements to implement outer joins. Prior to that each database vendor had its own extension to the standard to implement outer joins.

- Temporary tables. The SQL standard did not provide for capturing the rows from a SELECT statement into a temporary table. Informix and Postgres both allow this but with differing syntax.

- SERIAL datatype. Not part of the SQL standard but an Informix extension. PostgreSQL has a SERIAL type but it is used differently. With Informix, you supply a zero as the SERIAL value, and the engine replaces the zero with the next serial number. With PostgreSQL, you don't supply a value and the engine supplies the next serial number as a default. If you supply a zero, it is accepted!

- Functions. Informix has a number of functions TODAY, CURRENT, USER, MDY(m,d,y), EXTEND, etc which are not in the SQL standard or have different names (e,g NOW(), CURRENT_DATE, etc).

- MATCHES clause. Informix, in addition to the SQL standard LIKE clause, allows you to SELECT rows which match patterns using the Unix shell wildcard symbols ([]*?). PostgreSQL has a ~ operator which matches RE (regular expression) patterns in the manner of perl.

- Mandatory FROM clause. In Informix, the SELECT statement must have a FROM clause. PostgreSQL (and others like Sybase) does not require a FROM clause.

- MONEY datatype. A variant on DECIMAL which is suitable for financial systems.

A4GL allows you to connect to different database engines. This leads to difficulties when you are coding into your 4GL programs any of the above Informix idioms which are not part of the SQL standard. To use Aubit4GL with non Informix engines, you need to confine yourself to just the ANSI standard, or rely on Aubit4GL's translation mechanism to convert to Informix, or get a special version of the engine which supports the Informix variations. Nearly all major applications written in 4GL exploit the Informix SERIAL behaviour and the 4GL code usually relies on getting the serial value for the `sqlca.sqlerrd` record. For this you need an Informix compatible engine.

Aubit4GL can connect directly to

- Informix SE, IDS 7, or IDS 9 or later. Best of breed commercial engines with full SQL92 compliance. You must purchase a licence from IBM-Informix in order to use it. Has a multi-threaded architecture which gives it a performance advantage over all of its rivals. Now that it is owned by IBM, it will gradually be absorbed into IBM's own DB2 range of products and will gradually disappear.

- PostgreSQL a free opensource engine now with full SQL92 compliance. Fully free and opensource. Shares its origins with Ingres at UCB (University of California Berkeley). Unlike Informix IDS, it is not based on a threaded architecture and each frontend connection results in a separate backed process being spawned to service it. You can get postgresql from :
  `www.postgresql.org.`
  At the time of writing, the current version is 8.3. Each Linux distribution has its own RPMs which you get from the distribution site (try a Google search). Mike Aubury has created a native connection dubbed pg8 which works with this version.
  In the past there was a special version of PostgreSQL 7.4 patched to imitate the Informix behaviour mentioned above: The site for this project was:
  `gborg.postgresql.org/pginformix/download/download.php`
  and you could get the source tarballs there. You could get the RPMs from
  `informix.postgresintl.com.`
  The patch project has not been updated for PostgreSQL 8.0 onwards and we recommend that you change to using the pg8 library instead. These RPMs are known to install OK on SuSE 9.0 and you may be lucky on similar systems of equivalent vintage. The RPMS are patched from version 7.4. If you are installing the RPMs on a system with PostgreSQL RPMs already installed, you may need to add the `--oldpackage` argument to the rpm -Uvh command if the installled version is 7.4.1 or 7.4.2. It is hoped that future versions of PostgreSQL will fold these Informix patches into the regular distribution. The latest patched postgres version is also available from Aubit website `http://www.aubit.com`

- SAPDB a recently free and opensource engine up to version 7.4 with threaded architecture. The engine is SAP's tried and true commercial product and is solid and very fast. Unfortunately, MySQL have acquired the rights to develop the next version of SAPDB (to be renamed MAXDB) and the interfaces will no longer be free (GPL but not LGPL licensed). Best avoided unless a project based on the LGPL base is spawned.

- SQLite a free and opensource embeddable engine with nearly full SQL92 compliance. A small engine (only 25K lines of C source code) which we actually deliver statically linked into our binary distributions of Aubit4GL. It supports most of the SQL92 standard but is typeless (everything is either a char type or numeric and the distinction is not enforced). Get it from `www.sqlite.org`

- Any other database engine with an ODBC interface including PREPARE and SCROLL CURSOR statements.

3.3.4.2 No SQL

It is possible to use 4GL without using any embedded SQL. The 4GL language can be used as a general purpose programming tool. A dummy set of SQL functions is invoked with the A4GL_SQLTYPE=nosql option.

More usually of course, you will want to use SQL within your 4GL programs. You can use ODBC or one of several possible native connections to a RDBMS (Relational Database Management System). You tell the Aubit4GL compiler (or programs compiled by it) where to send its SQL statements by setting an environment variable: A4GL_SQLTYPE.

3.3.4.3 ODBC

ODBC (Open Database Connectivity) is an X/Open and ANSI standard CLI (Call Level Interface) for communicating with database backends through a common library called a Driver Manager which in turn uses another library (called a driver) appropriate to the backend desired. All ODBC libraries implement common functions (an API or Application Programming Interface) with the details of the functions tailored to the particular backend,

ODBC comes in two broad categories:

1. Driver Managers (e.g. unixODBC, iODBC, Windows ODBC) which act as a go-between and can plug in vendors' drivers
2. Direct (e.g. Informix, PostgreSQL, SAPDB, SQLite) which link directly to the vendors' drivers

Aubit4GL can handle embedded SQL with a library of ODBC (Open Database Connectivity) functions intended for passing to an implementation of ODBC. You need to install the ODBC application as well as the database vendor's odbc library files. (These latter may or may not come with the ODBC application).

On Unix/Linux platforms the ODBC options supported are

- unixodbc a free opensource ODBC manager with a supplied SQL frontend (good for testing the database). See `www.unixodbc.org`
- iodbc an ODBC manager from OpenLink, commercial but free to use. See `www.iodbc.org`
- ifxodbc direct ODBC to Informix engines (using libraries from Informix CSDK)
- pgodbc direct ODBC to PostgreSQL engines (free opensource)
- sapodbc direct ODBC with SAPDB (a free opensource Database Engine up till version 7.3)

3.3.4.4 ODBC config files

ODBC configuration is held in files: `/etc/odbcinst.ini` (driver info) and `/etc/odbc.ini` (datasources). Each user may have his own configuration in `~/.odbc.ini` (where ~ means the user's home directory). Applications often supply nice GUI applications to simplify editing these files. Unfortunately implementation of ODBC is so inconsistent between database suppliers, that these GUIs are useless. Use vi and edit the files by hand. Then observe the notes for each vendor and copy or link the files appropriately.

3.3.4.4.1 Sample odbcinst.ini The file odbcinst.ini holds a list of ODBC drivers. An example:

```
[Informix]
Driver=/opt/informix/lib/cli/libifcli.so
Setup=/opt/informix/lib/cli/libifcli.so
APILevel=1
ConnectFunctions=YYY
DriverODBCVer=03.00
FileUsage=0
SQLLevel=1
smProcessPerConnect=Y

[PostgreSQL]
Driver=/usr/lib/libodbcpsql.so
Setup=/usr/lbi/libodbcpsqlS.so
FileUsage=1
Threading=2

[SAPDB]
Driver=/opt/sapdb/interfaces/odbc/lib/libsqlod.so
Setup=/usr/lib/libsapdbS.so
FileUsage=1
CPTimeout=
CPReuse=
```

The Informix drivers will not tolerate whitespace (blanks or tabs) in the above file.

3.3.4.5 ODBC Datasources

Access to ODBC databases is configured in odbc.ini files which contain all the information required by the vendor's drivers to allow a connection. For example:

```
[infstores]
Description=Informixstores demo database
Driver=/opt/informix/lib/libifcli.so
Database=stores7
LogonID=fred
pwd=zxcv132
ServerName=elvis
CLIENT_LOCALE=en_us.8859-1
TRANSLATIONDLL=/opt/informix/lib/esql/igo4a304.so
[pgstores]
Description=Postgres stores demo database
Driver=PostgreSQL
Trace=Yes
Tracefile=sql.log
Database=pgstores
Servername=localhost
UserName=
Password=
Port=5432
Protocol=6.4
ReadOnly=No
RowVersioning=No
ShowSystemTables=No
ShowOidColumn=No
FakeOidIndex=No
ConSettings=
[SAPstores]
Description=SAP stores demo database
Driver=SAPDB
ServerNode=elvis
ServerDB=stores
```

In principle, the Server property should be the name from the odbcinst.ini list of drivers, but the Informix driver needs the full path to the driver library file.

The Informix driver will not find the /etc/odbc.ini file unless you point to it with the environment variable: ODBCINI

```
export ODBCINI=/etc/odbc.ini
```

Note that the different vendors use different keywords for naming the same things, and they have different sets of properties.

3.3.4.6 Informix ODBC Drivers

Informix give a choice of 4 ODBC drivers. They are installed in $INFORMIXDIR/lib/cli (usually /opt/informix/lib/cli on Linux systems). There appear to be 7 files but 3 of them are links to other files. Informix does not use separate files for setup; each library file contains both driver and driver setup functions.

	Static		Dynamic	
Threaded	`libthcli.a`		`libthcli.so`	or `oclit09b.so`
Unthreaded	`libcli.a`	or `libifcli.a`	`libifcli.so`	or `iclis09b.so`

3.3.4.6.1 Informix Driver Manager Informix supplies a driver manager replacement (DMR) file with 2 links:
`libifdmr.so`
`idmrs09a.so`

3.3.4.7 PostgreSQL Drivers

PostgreSQL ODBC drivers are installed by default in /usr/lib

	Static	Dynamic
driver	libodbcpsql.a	libodbcpsqlso
driver setup	libodbcpsqlS.a	libodbcpsqlS.so

Note that there is a separate file Postgres driver setup.

3.3.4.8 SAPDB Drivers

SAPDB drivers are installed by default in `/opt/sapdb/interfaces/odbc/lib/`

	Static	Dynamic
Driver	`libsqlod.a`	`libsqlod.so`

For SAPDB, use driver setup file from unixODBC: `/usr/lib/libsapdbS.so`

SAPDB will not find its odbc.ini file unless it is in `/usr/spool/sql/ini` (which it will have created at install time). You must either copy or link `/etc/odbc.ini` to that directory:

```
cd /usr/spool/sql/ini
ln -s /etc/odbc.ini .
```

On Linux systems `/usr/spool` with be a symbolic link to `/var/spool`

3.3.4.9 ODBC Warning

There are different versions of ODBC (2.5, 3.0, 3.5) - each with its own peculiarities. There are also big differences between what is *required* and what is *optional* - not all drivers implement the full ODBC functionality.

3.3.4.10 Native

Aubit 4GL can process DATABASE statements directly if it has a native interface to the database engine. To achieve this, we need the database vendor's ESQL/C compiler (Embedded SQL in C) available when we compile the A4GL compilers.

Embedded SQL/C is an ANSI SQL standard for allowing you to embed SQL statements into C source files. The SQL statements are enclosed within `EXEC SQL ... END SQL` tags. Traditionally the ESQL/C file has a `.ec` suffix. A vendor supplied pre-compiler then replaces the SQL statements with appropriate calls to functions in the vendor's libraries. The result of the compile is a C code `.c` file which can be compiled and linked to make executables, modules, or `.so` or `.a` library files.

At install time, the Aubit 4GL configure program looks for vendors ESQLC files and builds an interface to each of the vendor databases detected.

Backend	ESQL compiler	Suffix	
Informix	`/opt/informix/bin/esql`	`.ec`	
PostgreSQL	`/usr/bin/ecpg`	`.pgc`	
SAPDB	`/opt/sapdb/interfaces/precompiler/bin/cpc`	`.cpc`	

SQLite??? Help here please!

A4GL Native Connections			
SQLTYPE	RDBMS	Compiler	Comment
esql	Informix	esqlc	
esqlPG	PostgreSQL	ecpg	
pg	PostgreSQL	ecpg	
esqlSAP	SAPDB	cpc	
esqlQ	Querix	esqlQ	
sqlite	SQLite	???	
sqliteS	SQLite	???	Setup? Static link to 4glc?

The environment variable A4GL_SQLTYPE determines which connection is used when program (or 4glc compiler) is run.

3.3.5 Curses

If you want A4GL to use 4GL's character screen control statements (e.g. MENU, DISPLAY, DISPLAY ARRAY, etc), you will need the curses library:

NCURSES v 1.78 or later.

3.3.6 PDFLib

An extension to the 4GL language allows A4GL to exploit a PDF (Portable Data Format) library to produce fancy reports. This is an optional feature and if unavailable when you build the A4GL compiler, a library of do-nothing dummy PDF functions will be built-in to the compiler.

To get the PDFLib library go to the site: `www.pdflib.org`

3.3.7 GTK

A4GL has the ability to:

- display normal screen statements using Graphical Display widgets.
- support a set of extensions to the language to interface with Graphical objects such as Checkboxes, Pulldowns, Buttons, etc.

To exploit the graphical capabilities of A4GL, you need the GNOME GTK (Graphical Tool Kit) development library available at installation and run time.

3.3.8 Install Source

Having downloaded the source, whether from a tarball or via cvs

- cd $SRCDIR/aubit4gl
- Read the file README or README.txt
- Run the configure command:
  `./configure`
  - This will search for all the prerequisites and options and build Makefiles appropriately
  - configure has a lot of options. Try: `./configure --help`
  - The configure script will report all the prerequisites and options it finds and report any missing elements. If there are prerequisites missing or in non-standard locations, you can deal with this and run configure again.

- If configure seems OK then run the make command:
  `make`

- There are other arguments to make which may be useful to you, especially if things go wrong and you have to alter your setup (e.g. by installing some missing optional software):
  `make cleanall`
  `make log`
  The cleanall target will undo the effects of a previous make.
  The log target will save all the output from make into a file `make.log` which you can email to the aubit email lists when you want help with an install problem.

- You will know the make succeeded if you see a message like the following
  `A4GL compiled successfully`

- If make runs with no untoward error messages then you can install. You need root permissions to do this:
  `su`
  `make install`

The install program will install the compiler in /opt/aubit4gl. It will create 2 links in /usr/local/bin:
`/usr/local/bin/aubit` and `/usr/local/bin/aubit-config.`

You may now remove the contents of $SRCDIR - they have served their purpose.

3.3.9 /usr/local/bin/aubit

This program reads the Aubit configuration files and sets its environment variables so that the commands you submit to it will run correctly. e.g.

```
aubit 4glc hello.4gl -o hello
```

will compile the module hello.4gl and create an executable hello. The `aubit` program obviates the need to have `4glc` and its friends in your PATH.

Note: If you put `/opt/aubit4gl/bin` in your PATH, and set up your environment vars to match the contents of `aubit4glrc`, then you could dispense with the aubit command and simply type:

```
4glc hello.4gl -o hello
```

3.3.10 /usr/local/bin/aubit-config

`/usr/local/bin/aubit` uses `/usr/local/bin/aubit-config` to find the Aubit configuration file settings and sets its own environment variables to match. You can use it to inspect 1 or all of these settings. The values set are held the file:`/etc/opt/aubit4gl/aubitrc` which is created by the make install command.

Run the command `aubit-config -a` to see what has been configured.

Run the command `aubit-config A4GL_SQLTYPE` to get the value of that variable.

- configure your .a4glrc defaults; they are used in the compiling of the compiler, and later when the 4glpc script is invoked.
  - Especially be careful to point ODBC_LIB_DIR to the location of your ODBC shared library: For Openlink it should be `libiodbc.so` file in openlink_inst_dir/lib.
  - The a4glrc file in $AUBITDIR is read first. If you have one in $HOME, it will override the one in $AUBITDIR
- If your make process exited with message "Aubit 4GL compiler is now compiled", go to section "Testing the installation".

Please be aware that to perform database related operations, you will need in addition to a database engine installed, and a database created:

- an ODBC manager and ODBC driver installed
- odbc.ini file appropriate for your database, database engine and ODBC manager.
  You can use odbc.ini.example file in "test" directory as example. This will unfortunately not eliminate the need to read the documentation for these products.

3.4 Install Binaries

We will in future distribute binaries as RPM (RedHat Package Manager) files. The command to install an RPM file is:
`rpm -Uvh aubit4glbin-0.47-32.rpm`

- Prerequisites are same as for Installation:

- Please note: the binary only distribution is not available at the moment. Please contact the development team if you can assist in making RPM distribution
- copy the a4glrc.sample to $HOME/.a4glrc
- When your compiler is installed, read "Testing the installation of compiler"
- Please see Aubit 4GL download page for available binary builds. I Plan to provide both tar.gz and RPM, and make this scriptable using main makefile, and stuff it in corn, so I can provide nightly builds...

3.4.1 Testing the compiler

- cd to $AUBITDIR/test directory:
  `cd $AUBITDIR/test`
- compile the hello.4gl program:
  `aubit 4glc hello.4gl -o hello`
- Test the form compiler:
  `aubit fcompile form`
- Run the hello program:
  `./hello`
  and you should see a little program with with three options in the menu on the top:
  `Hello test:  window prompt form config exit`
  If you do, congratulations, you have just compiled your first 4GL program using Aubit 4GL!

There are a few more test files there. Most important is `hello_db.4gl`, that is connecting to the database. For it to compile and run, you will need to have a database, and ODBC DSN configured.

Chapter 4

Compiling 4GL programs & Forms

4.1 A4GL compilers

A4GL provides the following compilers:

- `4glc` which translates x4GL code into C
- `4glpc` which is a wrapper to call 4glc and gcc (or esql/c)
- `fcompile` which creates a binary form file from source
- `mcompile` which creates a binary menu file from source
- `amkmessage` which creates a binary help file from source

On Linux/Unix systems these programs are usually invoked as arguments to the 'aubit' script, e.g.

```
aubit 4glpc myprog.4gl -o myprog
```

The aubit program sets the environment from Aubit4GL configuration files and ensures that LD_LIBRARY_PATH includes the appropriate A4GL libraries. You can omit it and use 4glc/4glpc etc directly if you setup LD_LIBRARY_PATH & PATH correctly, as well as any settings specific to your installation.

This file if first read from Aubit 4GL installation directory, as specified by $AUBITDIR, and then, if it exists, from users home directory, as specified by $HOME, effectively overriding settings from $AUBITDIR/.a4glrc that exist in both places. It also accepts a number of command line switches, and environment variables.

4.2 4glpc

The 4glpc compiler is really just a wrapper around the 4glc, gcc, and esql/c compilers. The idea is that the type of each file passed on the command line is determined, as well as the output object type, and the relevant compilers are called in stages to generate that output. For example :

4glpc myprog.4gl -o myprog.4ae

Assuming we are compiling using A4GL_LEXTYPE=EC, then we know that we must :

- compile myprog.4gl -> myprog.ec using 4glc
- compile myprog.ec -> myprog.c using the esql compiler
- compile myprog.c -> myprog.o using 'gcc' or some other C compiler
- link myprog.o -> myprog.4ae

For A4GL_LEXTYPE=C, we can just remove the myprog.ec -> myprog.c and generate myprog.c directly from the 4gl.

4.2.1 Usage

Basic Aubit 4GL compiler usage

```
4glpc [options] -oOutFile.ext file.ext [file.ext...]
```

Extensions (.ext):

In files list, all .4gl files will be compiled to c or .ec etc as applicable , other files passed to linker.

The extension specified on the file passed to the '-o' flag will normally decide type of linking:

ao=object

aox=static library

aso=shared lib

4ae=executable.

Options

Option	Meaning
-L	Passed directly to the C compiler (specifies where libr
-o	Specify the output file
-c	Compile only - no linking
-e	Just generate the .c file
-I	Passed directly to the C compiler (specifies where inc
-G or –globals	Generate the globals map file
-S or –silent	no output other then errors
-V or –verbose	Verbose output (-V1.. -V5 for increasing levels of ver
-N name	Prefix all functions with name (default 'aclfgl_')
–namespace name	Same as -N option
-n or –noprefix	remove any prefix from function names (= -N ' ')
-v or –version	Show compiler version and exit
-f or –version_full	Show full compiler version and details
-h or -? or –help	Show this help and exit
-t TYPE or –lextype TYPE	output language, TYPE=C(default), EC, or PERL
-td TYPE or –lexdialect TYPE	Specify the output language dialect for ESQL/C gene
-k or –keep	keep intermediate files (default)
-K or –clean	clean intermediate files when done
-s[01] or –stack_trace [01]	Include the stack trace in file: 0-Don't generate 1-Ge
--use-shared/–use-static	compile with shared libraries
-echo	Don't really compile (ignored for now)
-d dbname	Specify an alternative database name to use for comp
–database dbname	same as -d option (note ignores that specified in the .
-4 or –system4gl	Used internally - Ignores any clashes with builtin libr
–map	Generate an unload file with some 4GL code metrics
–as-dll	Generate a shared library as the output type
–make-compile	Compare file times and only recompile where required

Examples:

```
$ 4glpc sourcefile.4gl -o executablename.4ge
```

```
$ 4glpc sourcefile.4gl -c -o objectname.o
```

```
$ 4glpc -shared file.4gl -o file.4ge
```

```
$ 4glpc -static -echo file.4gl -o file.4ge
```

```
$ 4glpc -debug file.4gl -o file.debug 4glpc -map -echo file.4gl
```

As a matter of interest - the 4glpc compiler itself is written in Aubit4GL.

The 4glpc compiler will use a number of configuration files ($AUBITDIR/tools/4glpc/settin
to control what commands will be used and what options will be passed to
them. These will normally be setup correctly, but if you wish to change them
(for example if you are porting to a new database backed, or a new platform),
then you may need to know the order in which they are read.

This will depend on the A4GL_LEXTYPE, A4GL_LEXDIALECT, TARGET_OS, TARGET.

For an example, assume A4GL_LEXTYPE is set to EC, A4GL_LEXDIALECT=POSTGR
TARGET_OS=linux (this is set by the ./configure script at compile time),
and TARGET=i686-pc-linux-gnu (this is also set by the ./configure)

Files will be read as :

tools/4glpc/settings/EC

tools/4glpc/settings/EC_POSTGRES

tools/4glpc/settings/linux

tools/4glpc/settings/linux___EC

tools/4glpc/settings/i686-pc-linux-gnu

tools/4glpc/settings/i686-pc-linux-gnu___EC

tools/4glpc/settings/i686-pc-linux-gnu___EC_POSTGRES

Settings in any later configuration file will overwrite those in any previous file. This gives the maximum configurability possible.

4.3 4glc

Aubit 4GL source compiler 4glc is generally invoked using the 4glpc wrapper. It can be involked directly :

```
aubit 4glc <filename>.4gl
```

For historic reasons, the 4glc compiler can also compile most modules to an executable. In order to do this the 4glc compiler uses the normal C compiler and passes unknown options on to it e.g.:

```
aubit 4glc file.4gl -c -o file.o
```

```
aubit 4glc -shared file.4gl -o file.4ge
```

```
aubit 4glc -static -echo file.4gl -o file.4ge
```

```
aubit 4glc -debug file.4gl -o file.debug
```

```
aubit 4glc -map -echo file.4gl
```

compiles to an object file rather than a linked executable

It is now best practice, unless there is a very good reason otherwise, to not call 4glc directly as all, and to invoke it via the 4glpc compiler instead.

4.4 Compiling forms

```
$ aubit fcompile file.per
```

fcompile compiles form compatible with both GUI and CUI run-time modes.

4.5 Compiling help files

Compile these using amkmessage

```
$ amkmessage helpfilename (without .msg extension)
```

This will generate the compiled help message file with a .hlp extension. Please note that many Informix-4GL programs assume that compiled help file will have extension ".iem". You can just rename created .hlp file to .iem if needed.

For format and syntax of help files, please see example file in test/ directory. It is fully compatible with Informix standard definition.

4.6 Compiling menu files

Menu files are currently not used, so you can safely ignore them (for now...)

Chapter 5

Configuration

5.1 Introduction to configuration

Classical Informix 4GL does the following:

1. Processes Informix SQL statements embedded in 4GL code.
2. Outputs a program in the C language.
3. Connects to an Informix Database Engine (IDS or SE)
4. Interfaces with the user using `curses` on a terminal (or xterm emulator)

Aubit 4GL does all of this but is much more versatile.

1. It can process other dialects of SQL: (PostgreSQL, Oracle, SAPDB, ODBC)
2. It can output (by design at least) in Perl. One day it may produce Java.
3. It can connect to other database engines:(PostgreSQL, Oracle, SAPDB, SQLite, ODBC)
4. It can interface using Graphics (via GTK Gnome Tool Kit) or even a no curses console. In addition, a special graphical menu structure is available.
5. It can optionally output reports in PDF format (using A4GL enhancements to the 4GL language.
6. It can use RPC (Remote Procedure Calls) for n-tier applications

By the magic of Dynamically Linked Libraries (called shared objects in Unix/Linux), most of the these options can be chosen at runtime. Different libraries implement the same set of required functions for each option.

The programmer or the user can choose from these myriad options by editing by setting environment variables before invoking the compiler (or the compiled program).

Usually however, you setup the default options in `aubitrc` files then on Unix/Linux systems you invoke the `aubit` program to run the programs you want to call. The aubit program sets environment variables from the aubitrc files and also ensures the library files are available.

5.1.1 configurator

Aubit 4GL supplies a 4GL program configurator which shows you all the switches and their permitted values. You run it with the command:

```
aubit configurator
```

Note on Microsoft systems there is no `aubit` script. configurator will be in your PATH, so just type `configurator`

A full list of all configuration options can be found in Appendix A.

5.1.2 Essential Configuration flags

5.1.2.1 A4GL_SQLTYPE

This switch chooses where to send SQL embedded in the 4GL code. Default is nosql. (which means use dummy do-nothing functions for SQL code).

Other options available may include :

Option	Database Backend
esql	Informix esql/c native
nosql	SQL statements are effectively ignored
pgodbc	postgreSQL odbc
sapodbc	SAPDB odbc
iodbc	Openlink ODBC
unixodbc	unixodbc (free opensource)
ifxodbc	Informix CLI (now called ODBC)
odbc32	Only on Windows systems

The list of available options will depend on what was detected at compile time - check the $AUBITDIR/lib directory.

5.1.2.2 A4GL_UI

This switch determines which frontend you are using. Options are CONSOLE, TUI, GTK.

5.2 aubitrc files

The aubit program sets A4GL_... environment variable then executes the arguments on the command line. It reads configuration options in the following files in order:

1. `/etc/opt/aubit4gl/aubitrc`
2. `$AUBITDIR/etc/aubitrc (AUBITDIR=/opt/aubit4gl by default)`
3. `~/.aubit4gl/aubitrc`
4. `./.aubitrc`
5. Environment variables

As each of these files is read, it overwrites the values of previous files. The environment variables have final precedence.

To set up your A4GL system, you edit these files with text editor (e.g. vi)

For system wide configuration, edit /etc/opt/aubit4gl/aubitrc whereas for personal idiosyncrasies edit your home directory's .aubitrc file.

Chapter 6

4GL Language

6.1 Introduction

The 4GL programming language was born in Informix Corp in 1986. Because of that, and not to conflict with with 4GL as general programming concept (BASIC is in principle also a Fourth Generation Language, as opposed to C, which is a Third Generation Language), we should refer to basic 4GL syntax as I4GL.

Today, even among Informix-distributed products, there is distinction between classic I4GL and D4GL (Informix name for 4J's 4GL compiler), which introduced a number of language enhancements. Then Informix implemented some of these enhancements back into classic 4GL, and added some of it's own (v 7.3), which 4J in turn implemented in its *Universal Compiler V3* (this is the actual name for 4Js product that Informix distributes under the name *D4GL - Dynamic 4GL.*)

We refer to the syntax of different implementations as:

- I4GL - Informix non-GUI, a.k.a. "classic" products syntax, V 7.3
- D4GL - 4Js extended syntax, including I4GL
- A4GL - Aubit 4GL specific syntax, including I4GL
- x4GL - all of the above as general name for all

Luckily for us, Querix decided not to change the language, but instead do all GUI related configuration from separate configuration files.

Aubit 4GL, as a package, and A4GL, as a language definition, is a superset of I4GL.

Our first aim is to provide full *unconditional* compatibility with I4GL. Since this means that 90% of the syntax used in A4GL will be I4GL, and since this document is not intended to be an I4GL manual, we strongly suggest that you refer to existing Informix documentation and tutorials downloadable from their web site, and books about 4GL, like:

Informix Unleashed, (ISBN 0672306506) a complete book in HTML format about Informix products, by John McNally. You will find several complete chapters about 4GL language there, including chapters on Informix database servers. You will also learn there that "To develop with a 4GL, the developer does not have to be an expert programmer".

(I have asked the author for permission to include his book in Aubit 4GL distribution, but received no answer)

The rest of this page will serve as a quick and dirty crash course to give you some idea of what the I4GL looks like, as a language.

For A4GL extensions. please refer to the appropriate sections of this manual.

6.2 Summary:

* To learn I4GL, refer to Informix manuals for Informix-4GL version 7.3 (http://www.informix.com or direct links to *Informix 4GL by example*,*Informix 4GL Concepts and Use*, *Informix 4GL Reference Manual* - please remember that exact locations can change, and if they do, use the search function on the Informix web site to find new locations of this documents), and third-party books.

* To learn about A4GL extensions, read this manual

* To get some idea about what I4GL looks like, and to get some idea about combined I4GL and A4GL functionality, continue reading this page

* To get 4GL code examples, go to http://www.informix.com/idn and look for the Example application, or download one of GNU 4GL programs from http://www.falout.com

6.3 Short Intro to x4GL

- 4GL Programs
- Structure of a program
- DATABASE section
- GLOBALS section

- Functions
- MAIN block
- DEFINE section
- 4GL Commands

6.3.1 4GL Programs

A 4GL program consists of a series of modules and forms. Each 4GL module can contain functions and reports and each program must contain exactly one 'main' section and must end in a .4gl extension. C modules can also be included in programs.

6.3.1.1 Structure of a program

database section

globals section

function/report/main block

.

.

.

.

function/report/main block

6.3.1.2 DATABASE section

This section is optional and is of the format :

```
DATABASE database-name
```

The database name is actually the DATA SOURCE NAME (DSN) from the ODBC drivers.

6.3.1.3 GLOBALS section

This optional section allows you to define variables which are accessible to all modules. There is normally a single file (typically called 'globals.4gl') where variables are defined. All other modules which need these variables

then include that file using the `GLOBALS` statement .eg.
`globals.4gl:`

```
GLOBALS
DEFINE a INTEGER
END GLOBALS
```

`module.4gl:`

```
GLOBALS "globals.4gl"
```

Note: In Aubit 4GL the 'globals' module (containing the GLOBALS / END GLOBALS) must be compiled first.

6.3.1.4 Functions

A function in 4GL is a sequence of commands which are executed when called from another block of code. A function can accept parameters and can return values.

A function is defined :
`FUNCTION function-name ( parameter-list )`
define-section
commands
`END FUNCTION`

Values are returned using the RETURN keyword:
`RETURN value`

6.3.1.5 MAIN block

Each program must contain a main section - it is the starting point in any program.

```
MAIN
define-section
commands
END MAIN
```

6.3.1.6 DEFINE section

This optional section allows you to define variables which may be used subsequently. In its simplest form:

`DEFINE variable_names datatype`
or

```
DEFINE CONSTANT constant_name "Value"
DEFINE CONSTANT constant_name Number-Value
```

More than one variable can be defined as any type in the same statement by separating the names with a comma:

```
DEFINE a,b,c INTEGER
```

Available datatypes are :
SMALLINT (2 byte integer)
INTEGER (4 byte integer)
CHAR (Single character 'string')
CHAR(n) (n character string)
MONEY
DECIMAL (These are not fully implemented)
FLOAT (8 byte floating point number - (C double))
SMALLFLOAT (4 byte floating point number - (C float))
DATE (date - number of days since 31/12/1899)
DATETIME
INTERVAL
BYTE
TEXT
VARCHAR Unimplemented yet
LIKE tablename.columnname
RECORD LIKE tablename.*
- can only be used when the module has a DATABASE statement. These copy the datatypes directly from the database either for a simple column, or to generate an entire record (see below)

Special datatypes are :

ARRAY[n] OF datatype defines an array

RECORD .. END RECORD defines a record structure

ASSOCIATE [CHAR](m) WITH ARRAY[n] of datatype .... defines an associative array (hash table).

6.3.1.7 Arrays Syntax:

DEFINE vars ARRAY[n] datatype eg.
DEFINE lv_arr ARRAY[200] OF INTEGER defines an array of 200 elements each being an integer. Elements of an array are indexed from 1 to the number of elements specified.
IMPORTANT: *No bounds checks are made.* Accessing elements which are outside those defined (i.e. <1 or > n) will result in an

error (Usually a core dump). Eg
LET lv_arr[1]=1
LET lv_arr[200]=200
LET lv_arr[201]=201 # this will cause a program fault!

6.3.1.8 Records

Records are structured groups of data, with the entries separated by commas. Elements within a record are accessed via the syntax: record name '.' element name.

6.3.1.8.1 Syntax

```
DEFINE recordname RECORD
 element datatype,
 element datatype
 ...
END RECORD
```

eg.

```
DEFINE lv_rec RECORD
 elem1 CHAR(10),
 elem2 INTEGER
END RECORD
```

defines a record with two elements. eg.

```
LET lv_rec.elem1="string1"
Record may also be nested and used in conjunction with arrays. The fol
DEFINE lv_record ARRAY[20] OF RECORD
 elem1 CHAR(20),
 elem2 INTEGER
END RECORD
DEFINE lv_record RECORD
 a ARRAY[200] of INTEGER,
 b CHAR(20)
END RECORD
DEFINE lv_record RECORD
 subrecord1 RECORD
  elem1 CHAR(10),
  elem2 INTEGER
 END RECORD,
```

```
 subrecord2 RECORD
  elem2 DATE
 END RECORD
END RECORD
```

6.3.2 Associative Arrays

Associative arrays allow you to access data from an array using a string as a subscript rather than an integer. For example:

```
LET age<<"bob">>=40
DISPLAY age<<"bob">>
```

This can be especially useful when dealing with codes and code desciptions:

```
LET lv_desc<<"A">>="Active"
LET lv_desc<<"I">>="Inactive"
LET lv_desc<<"R">>="Running"
LET lv_desc<<"D">>="Deleted"
LET lv_state="A"
.
.
DISPLAY lv_desc<<lv_state>>
```

(This is for illustration, the data would normally be read from a database!)

To define an associate array:
`DEFINE name ASSOCIATE [CHAR] (nc) WITH ARRAY [nx] OF datatype`
Where nc is the number of characters to use for the index, and nx is the total number of elements that may be stored.

6.3.2.0.1 Performance Note Internally, associate arrays are stored using hash tables, for performance reasons always declare 'nx' much larger than is actually required. A factor of two is optimum in most cases.

Again the datatype used in this form of array may be a RECORD, ARRAY etc. Eg.

```
DEFINE lv_asoc1 ASSOCIATE CHAR(10) WITH ARRAY[10] OF INTEGER
DEFINE lv_asoc3 ASSOCIATE (10) WITH ARRAY[10] OF INTEGER
DEFINE lv_asoc2 ASSOCIATE CHAR(10) WITH ARRAY[10] OF RECORD
element1 CHAR(10),
element2 CHAR(20)
END RECORD
```

6.3.3 Constants

```
Constants are defined using:
DEFINE CONSTANT name value eg.
DEFINE CONSTANT max_num_vars 30
DEFINE CONSTANT err_string "There is an error"
IF num_vars>max_num_vars THEN
  ERROR err_string
END IF
It is also possible to use constants in any subsequent define sections
DEFINE CONSTANT num_elems 20
DEFINE lv_arr ARRAY [num_elems] OF INTEGER
IF num_vars<=num_elems THEN
  LET lv_arr[num_vars]=1
END IF
```

You can think of DEFINE CONSTANT statements as being equivalent to C `#define` statements (except that you cannot use them to define macros as you can with C).

6.3.4 Packages

The current system allows programs to call shared libraries using the syntax:

```
call library::function(..)
```

(See `tools/test/file.4gl` or `lib/extra_libs/pop/pop_killer.4gl` for some example usage)

Packages take this one step further in that the calls are coded like any other functions. They are detected at compile time by referencing a list of possible function name mappings specified by an `import package` statement. Syntax :

```
IMPORT PACKAGE packagename
or
USE packagename
```

The packagename should be the name of a file in the `$AUBITDIR/etc/import` directory.

A file called `default` exists in this directory which is included for all compilations - this allows you to add calls to your own subroutines just as if they were builtin functions with no need to add them to the compile line as object or library modules..

This file should contain a series of lines, each containing:

```
library functionname
```

(In this way a *package* can contain functions from more than one library...)

e.g.

```
A4GL_pcre pcre_match
A4GL_pcre pcre_text
```

Whenever the compiler sees a call to `pcre_match` it will call `pcre_match` in the A4GL_pcre library - in this way it's equivalent to `A4GL_pcre::pcre_match`

So a full .4gl may look like :

import package a4gl_pcre
main
if pcre_match("cat|dog","There was an old cat") then
display "Matches to ",pcre_text(1)
else
display "No match"
end if
end main

Compile and run

```
$ 4glpc pcre_test.4gl -o pcre_test
$ ./pcre_test
Matches to cat
```

(Note - you don't need to link against the library - it's done at runtime!)

(If you've got `pcre` installed - you can compile up the pcre library by doing a `make` in the `lib/extra_libs/pcre` directory)

6.4 4GL Quick Reference

Start of insert

6.5 Aubit4GL Quick Reference

6.5.1 Data Types

```
ARRAY[m,n,...] OF type
BYTE
CHAR(n)
CHARACTER(n)
DATE
DATETIME(f TO l)
DEC
DEC(precision)
DEC(precision,scale)
DECIMAL
DECIMAL(precision)
DECIMAL(precision,scale)
DOUBLE PRECISION
DOUBLE PRECISION(precision)
INT
INTEGER
INTERVAL(f TO l)
LIKE table.column
MONEY
MONEY(precision)
MONEY(precision,scale)
NUMERIC
NUMERIC(m)
NUMERIC(m,n)
REAL
RECORD LIKE table.*
RECORD name type ,... END RECORD
SERIAL
SERIAL(n)
SMALLFLOAT
SMALLINT
TEXT
VARCHAR
VARCHAR(max)
VARCHAR(max,min)
```

Precision = No of significant digits (default 16)
Scale=No or digits after the decimal pt (default 2), can be -ve.

max = number of chars (upper limit 254 for Informix IDS)
min = minimum number of chars.

Current Engines also support large integers: int8 and serial8.

6.5.2 Constants

```
TRUE=1
FALSE=0
NOTFOUND=100
```

6.5.3 Global Variables

Flags: INT_FLAG QUIT_FLAG

Vars: STATUS SQLCA.SQLCODE

SQLCA Record:

```
SQLCA RECORD
SQLCODE INTEGER,
SQLERRM CHAR(71),
SQLERRP CHAR(8),
SQLERRD ARRAY[6] OF INTEGER,
SQLAWARN CHAR(8)
END RECORD
```

SQLCA.SQLERRD Array:

SQLERRD[1]:estimated row count
SQLERRD[2]:serial value returned
SQLERRD[3]:no of rows processed
SQLERRD[4]:estimated CPU cost
SQLERRD[5]:error offset
SQLERRD[6]:last rowid processed

Warning: Not all of the above work for all backends. For PostgreSQL they may need a patched version of the engine.

6.5.4 Syntax Conventions

The remainder of this chapter uses the following conventions to indicate the syntax of 4GL language constructs

- KEYWORDS are in UPPERCASE. You enter them literally but in upper or lower case
- Lower case indicates terms for which you must enter your own identifiers or expressions
- "string" indicates a quoted string. Informix allows either single or double quotes but non-Informix engines may enforce one or the other.
- string (without quotes) indicates an unquoted string used for example, in naming cursors, prepared statements, forms, windows, etc.
- m and n are used to denote a numeric value
- "c" denotes any quoted character
- [] and {} delimit options. {} indicates a mandatory option. [] a non-mandatory toption. Within the [] or {} elements are separated by the pipe symbol |. e.g. {a|b|c} means you must choose a or b or c.

Expressions in red are Aubit 4GL extensions and will not compile on Informix, 4J, or other 4GL compilers.

-
- Expressions in green work with Informix SE only.

- Expressions in blue work with Informix IDS only.
- relop means a relational operator (see below)
- expr means an expression
- charexpr means a character expression (e.g. `filename || ".4gl"`)

6.5.5 Operators

Numeric: + - * / ** mod

String: , [m,n] || USING "string" CLIPPED

Relational: = <> != >= < <=

Boolean: expr relop expr
charexp LIKE charexpr
charexpr LIKE charexpr ESCAPE "c"
charexp NOT LIKE charexp
charexp NOT LIKE charexp ESCAPE "c"
charexpr MATCHES charexpr
charexpr NOT MATCHES charexpr ESCAPE "c"
charexpr MATCHES charexpr
charexpr NOT MATCHES charexpr ESCAPE "c"
expr IS NULL
expr IS NOT NULL
boolexpr AND boolexpr]
boolexpr OR boolexpr
NOT boolexpr
[NOT] IN ({expr,...
|selectstatement
|[NOT] EXISTS (selectstatement)

6.5.6 Attribute Constants

An attlist is a set of the following elements:

BLACK, WHITE, RED, GREEN, BLUE, MAGENTA, CYAN, YELLOW, REVERSE,DIM, BOLD, BLINK, INVISIBLE, BORDER, UNDERLINE

6.5.7 Key Constants

A keylist is a set of the following elements:

```
F1 to F64
CONTROL-c (but c not in (A,D,H,I,
  K,L,M,R,X)
ACCEPT, DELETE, DOWN, ESC, ESCAPE,
HELP, INSERT, INTERRUPT, LEFT,
RIGHT, NEXT, NEXTPAGE, PREVIOUS,
PREVPAGE, RETURN, TAB, UP
```

6.5.8 Table Privileges

```
ALTER, INDEX, DELETE, INSERT,
SELECT[(colname ,...)]
UPDATE[(colname ,...)
```

6.5.9 Comments

Characters on a line after the following are ignored by 4GL compilers:

–	ANSI SQL Standard for commenting out rest of line
#	Unix convention for commenting out rest of line

Curly braces are used to comment out lines of code (not nestable):

{ ... }	Compiler ignores everything between the braces
{! ... !}	Aubit 4GL compiles the enclosed code. Informix 4GL ignores it.

6.5.10 4GL Statement Syntax

```
ALLOCATE ARRAY name, size
ALTER INDEX indexname TO [NOT] CLUSTER
ALTER TABLE tablename
  {ADD (newcolname newcoltype
     [BEFORE old-colname][,...])
   |DROP (oldcolname[,...])
   |MODIFY (oldcolname newcoltype [NOT NULL]
     [,...])
  }[,...]
```

```
  AT TERMINATION CALL function([args])
```

```
BEGIN WORK
  statement ...
  {COMMIT WORK | ROLLBACK WORK}
CALL [packet{::|.}]function([args])
  [RETURNING arglist]
CASE [(expr)]
  WHEN {expr | booleanexpr}
    statement
    ...
    [EXIT CASE]
  ...
  [OTHERWISE]
    ...
    [EXIT CASE]
    ...
  END CASE
```

```
  CHECK MENUITEM name
  CHECK MENUITEMEM2 name
  CLEAR STATUSBOX name
  CLASS name [EXTENDS class]
    definestatements ...
    {FUNCTION func([arglist])
      statements
      [...]
     END FUNCTION}
     ...
    END CLASS
```

```
CLEAR {SCREEN |WINDOW windowname
   |WINDOW SCREEN
   |FORM

   [TO DEFAULTS]


   |fieldlist}
CLOSE cursor
CLOSE DATABASE
CLOSE FORM

   CLOSE SESSION name
   CLOSE STATUSBOX name


CLOSE WINDOW name

   CODE
     Cstatement;
     ...
     ENDCODE


COMMIT WORK

   CONNECT TO name

CONSTRUCT {BY NAME charvar ON collist
  |charvar on collist FROM { fields
     | screenrecord[[n]].*}  [,...]}
  [[ {BEFORE|AFTER} CONSTRUCT statements]
   [,...]
   [ {BEFORE|AFTER} FIELD field statements]
   [,...]
   {ON KEY (keylist)
      statement
      ...
      [{EXIT|CONTINUE} CONSTRUCT]
   ...]
END CONSTRUCT]
CONTINUE CONSTRUCT
CONTINUE DISPLAY
CONTINUE FOR
CONTINUE FOREACH
CONTINUE INPUT
CONTINUE MENU
CONTINUE PROMPT
CONTINUE WHILE

   CONVERT REPORT TO "filename" AS
     {"SAVE"|"PDF"|"CSV"|"TXT"}


CREATE AUDIT FOR tabname in
"pathname"
CREATE [UNIQUE|DISTINCT][CLUSTER] INDEX
 indname ON tabname( colname [ASC|DESC]
      [,...])
CREATE DATABASE {name| charvar}
   [WITH LOG [IN path]]
CREATE SCHEMA AUTHORIZATION
CREATE PRIVATE SYNONYM
CREATE PUBLIC SYNONYM
CREATE SYNONYM name FOR tabname
CREATE TABLE
CREATE [TEMP] TABLE name
   (colname coltype [NOT NULL][,...])
CREATE DISTINCT CLUSTER INDEX
CREATE VIEW
CURRENT WINDOW IS name
CURRENT WINDOW SCREEN
CURRENT WINDOW IS SCREEN
DATABASE name [EXCLUSIVE]
DEALLOCATE ARRAY name
DECLARE name [SCROLL] CURSOR FOR
  {select_statement
    [FOR UPDATE OF collist
  |insert_statement
  |statementid}
DEFER INTERRUPT
DEFER QUIT
DEFINE varlist datatype [,...]

   DEFINE CONSTANT id {"string"|number}
   DEFINE linkid LINKED TO tabname PRIMARY KEY (coln
   DEFINE name ASSOCIATE [CHAR](n)
      with ARRAY[m] OF datatype


DELETE FROM tabname
   [WHERE {condition|CURRENT OF cursor}]
DELETE USING linkid
DISABLE FORM name

   DISABLE MENUITEM name
   DISABLE MENUITEMS


DISPLAY {BY NAME varlist
  | varlist TO {fieldlist|screenrec[[n]].*}
       [,...]
     | AT screenrow,screencol]}
   [ ON KEY (keylist)
      statement
      ...
      [EXIT DISPLAY]
   ...
   END DISPLAY]
DISPLAY ARRAY id TO screenarray.*
   [ ATTRIBUTE( attlist )]
   {ON KEY (keylist)
      statement
      ...
      [EXIT DISPLAY]
   ...
   END DISPLAY| [END DISPLAY]}

DISPLAY FORM name [ATTRIBUTE( attlist)]
DROP AUDIT FOR tabname
DROP DATABASE {name | charvar}
DROP INDEX name
DROP SYNONYM name
DROP TABLE name
DROP TRIGGER name
DROP VIEW name
ENABLE FORM form

   ENABLE MENUITEM name
   ENABLE MENUITEMS


ERROR displaylist [ATTRIBUTE (attlist)]
EXECUTE [IMMEDIATE] statementid
EXIT CASE
EXIT CONSTRUCT
EXIT DISPLAY
EXIT FOR
```

```
EXIT FOREACH
EXIT INPUT
EXIT MENU
EXIT PROGRAM [expr]
EXIT PROMPT
EXIT WHILE
FETCH [NEXT
      |PREVIOUS|PRIOR|FIRST|LAST
        |CURRENT|RELATIVE n
        |ABSOLUTE n]
     cursorname [INTO varlist]
FINISH REPORT name

   FINISH REPORT name
       CONVERTING TO {{"filename"|EMAIL|}
         [AS {"SAVE"|"PDF"|"CSV"|"TXT"|MANY}
         [USING "filename" AS LAYOUT]}


FLUSH cursor

   FONT SIZE n


FOR var = expr TO expr [STEP expr]
   {statement|CONTINUE FOR|EXIT FOR}...
   END FOR
FOREACH cursor [INTO varlist]
   [statement|CONTINUE FOREACH|EXIT FOREACH]...
   END FOREACH
FREE {statementid|cursor|blobvar}

   FREE REPORT name


FUNCTION function([arglist])
   [ definestatement ]...
   statement ...
   END FUNCTION
GO TO label
GOTO label
GRANT {tabpriv ON tabname
   | CONNECT|RESOURCE|DBA }
   TO {PUBLIC|userlist}
HIDE OPTION name
HIDE WINDOW name
IF boolexpr THEN
  statement
  ...
  [ELSE
    statement
    ...
  END IF]

   IMPORT PACKAGE name


INITIALIZE varlist
  {LIKE collist| TO NULL}
INPUT ARRAY array [WITHOUT DEFAULTS]
  FROM screenarray.* [HELP n]
  [{BEFORE {ROW|INSERT|DELETE|FIELD list
    [,...]
    |AFTER {ROW|INSERT|DELETE|FIELD list
       INPUT}[,...]
     |ON KEY (keylist)}
     statement
     ...
     [NEXT FIELD field]
     ...
     [EXIT INPUT]
     ...
   ...
   END INPUT
INSERT INTO tabname[(collist)]
  {VALUES(vallist)| selectstatemet}

   INSERT USING linkid


LABEL name:
MESSAGE displaylist [ATTRIBUTE (attlist)]
LABEL label-name :
LET id = expr

   LET hasharray<<"code">> = "string"


LOAD FROM filename INSERT in tabname [(collist)]
LOCATE varlist in {MEMORY|FILE [filename]}
LOCK TABLE name IN {SHARE|EXCLUSIVE} MODE
MENU "name"
   COMMAND {KEY (keylist)
       | [KEY (keylist)] "option"
            [HELP n]}
      statement
      ...
      [CONTINUE MENU]
      ...
      [EXIT MENU]
      ...
      [NEXT OPTION "option"]
      ...
    ...
    [ON KEY (keylist)
      statement
      ...
      CONTINUE MENU]
      ...
      [EXIT MENU]
      ...
      [NEXT OPTION "option"]
      ...]
    END MENU

   MENU name
      {OPTION opt [IMAGE="path/name.xpm"] "Label"
      |SUBMENU subname "[_]Label"
        {USE menu
        |{statement,...
         END SUBMENU}}
      | statement
       ,...}
      END MENU
   MENUHANDLER name
      [ definestatement [,...]]
      [ statement
        |{{DIS|EN}ABLE MENUITEM[S] item [,...]}
        | ON item
            statement
            [...]]
      END MENUHANDLER
   MESSAGEBOX message


SET BUFFERED LOG
SET CONSTRAINTS ALL IMMEDIATE
```

```
SET LOG
START DATABASE identifier WITH LOG IN "..."
  [MODE ANSI]
START REPORT name
   [TO {file|PIPE program|PRINTER
|
    CONVERTABLE

}]
MOVE WINDOW
NEED n LINES
NEXT FIELD "fieldname"
NEXT FORM NEXT OPTION "optname"
OPEN cursor [USING varlist]
OPEN FORM name FROM "filename"
OPEN SESSION id TO DATABASE db
   [USING user [PASSWORD pwd]]
OPEN STATUSBOX name
OPEN WINDOW name AT row, col
   WITH {r ROWS, c COLUMNS
         | FORM "file"}
   [ATTRIBUTE(attlist)]
OPTIONS {MESSAGE LINE line
   |PROMPT LINE line
   |COMMENT LINE line
   |ERROR LINE line
   |FORM LINE line
   |INPUT {[NO] WRAP}
   |INSERT KEY key
   |DELETE KEY key
   |NEXT KEY key
   |PREVIOUS KEY key
   |ACCEPT KEY key
   |HELP FILE "file"
   |HELP KEY key
   |INPUT ATTRIBUTE(attlist)
   |DISPLAY ATTRIBUTE (attlist)}
   [,...]
OUTPUT TO REPORT name(exprlist)
PAUSE "charexpr"
PREPARE id from "charexpr"
PRINT exprlist
PRINT FILE "filename"
    PRINT IMAGE "name"


PROMPT displaylist FOR [CHAR] var
  [HELP n]
  [ON KEY (keylist)
    statement
    ...
  ...
  END PROMPT]
PUT cursor FROM varlist
RECOVER TABLE name
RENAME DATABASE name TO newname
RENAME COLUMN table.oldcol TO newcol
RENAME TABLE oldname TO newname
RESIZE ARRAY name, size
EXIT REPORT
RETURN  exprlist
REVOKE { tabpriv ON tabname
   | CONNECT | RESOURCE | DBA}
   FROM {PUBLIC | userlist
ROLLBACK WORK
ROLLFORWARD DATABASE name
```

```
RUN command [RETURNING n
   |WITHOUT WAITING]
SCROLL {fieldlist| screenrec.*}[,...]
   {UP|DOWN}[BY n]
SELECT sellist [INTO varlist] FROM collist
   [joinclause] [fromclause]
   [groupclause [havingclause]]
   [orderclause]

   SELECT USING linkid
   SET PAUSE MODE OFF
   SET PAUSE MODE ON
   SET CURSOR
   SET SESSION TO name
   SHOW MENU menu USING handler
      [FROM "file"]
   SHOW OPTION "optname"
   SHOW WINDOW name


SKIP n LINE[S]

   SKIP BY nval
   SKIP TO nval


SKIP TO TOP OF PAGE
SLEEP n
SQL sqlstatement [,...] END SQL
SET EXPLAIN OFF
SET EXPLAIN ON
SET ISOLATION TO COMMITTED READ
SET ISOLATION TO CURSOR STABILITY
SET ISOLATION TO DIRTY READ
SET ISOLATION TO REPEATABLE READ
SET LOCK MODE TO NOT WAIT
SET LOCK MODE TO WAIT
   START EXTERNAL FUNCTION


START REPORT name
   [TO {"filename"|PIPE program|PRINTER}]

   STOP ALL EXTERNAL
   TERMINATE REPORT
   UNCHECK MENUITEM name
   UNCHECK MENUITEMS name


UNLOAD TO filename selectstatement
UNLOCK TABLE name
UPDATE tabname SET
   {colname = expr [,...]
   |{(collist}|table.*|*}=
      {(exprlist)| record.*}}
   [WHERE {condition|CURRENT of cursor}
UPDATE STATISTICS
UPDATE STATISTICS FOR TABLE name

   UPDATE USING linkid
   USE packagename
   USE SESSION name


VALIDATE var LIKE collist
WHILE boolean
   [statement| EXIT WHILE | CONTINUE WHILE]...
   END WHILE
```

6.5.11 Report Syntax

```
REPORT repname( arglist)
  definestatement ...
[OUTPUT
  [REPORT TO
    {file|PIPE program|PRINTER}]
  [LEFT MARGIN n]
  [RIGHT MARGIN n]
  [TOP MARGIN n]
  [BOTTOM MARGIN n]
  [PAGE LENGTH n]
[ORDER [EXTERNAL] BY sortlist]
FORMAT
  { EVERY ROW
   | {[FIRST] PAGE HEADER
      |PAGE TRAILER
      |ON EVERY ROW
      |ON LAST ROW
      |{BEFORE|AFTER} GROUP OF argvar}
        statement
        ...
        [...]}
END REPORT
```

6.5.12 Report Statement Syntax

```
NEED n LINES
PAUSE "string"
PRINT [[exprlist][;]| FILE "filename"]
SKIP {expr LINE[S]| TO TOP OF PAGE}
```

6.5.13 Report Expressions

```
COLUMN expr
[GROUP]{COUNT(*)|PERCENT(*)
  |{SUM|AVG|MIN|MAX}(expr)}
   [WHERE expr]}
DATE
LINENO
PAGENO
TIME
WORDWRAP
```

6.5.14 PDF Report Syntax

PDF reports are an Aubit 4GL extension.

- nval means an numeric expr followed by 1 of the following units:

  POINTS, INCHES, MM, or nothing (which means char spaces). Example: 2.54 mm

```
PDFREPORT name(arglist)  definestatement ...[OUTPUT  [REPORT TO  {file|PIPE program|PRINTER}]
[ORDER [EXTERNAL] BY sortlist]FORMAT  { EVERY ROW  | {[FIRST] PAGE HEADER  |PAGE TRAILER
```

6.5.15 PDF Report Expressions

```
COLUMN nvalreportexpression
```

6.5.16 PDF Statements

```
PRINT IMAGE blobvar AS    "{GIF|PNG|TIFF|JPEG}"
```

6.5.17 PDF_FUNCTION arglists

There are many libpdf functions. For a full list look at the PDFlib documentation. Here are some useful examples:

```
"set_font_name", "{Times-Roman|Helvetica| ..}""se
```

Note: Font names are case sensitive.

6.6 Builtin Functions

Informix 4GL has a set of 40 or more functions built in to the language. Aubit4GL implements all of these.

Aubit4gl also implements a few functions to make the compiler compatible with programs written for D4GL.

Finally, Aubit4GL has added some builtins of its own to allow you to exploit Aubit4GL's special features such as GUI interfaces, different database engines, etc.

6.6.1 Standard 4GL Builtin Functions

The following functions in 4GL work in Aubit4GL:

Function	Comment
arg_val(n)	returns a string
arr_count()	returns smallint
arr_curr()	returns smallint
downshift(s)	returns string with chars downshifted to lowercase
err_get(n)	returns a string
err_print(n)	displays a string
err_quit(n)	displays a string then exits
errorlog(s)	logs message s to logfile
fgl_drawbox(h, w, y, x [,clr])	
fgl_getenv(s)	returns string
fgl_keyval(s)	returns integer code
fgl_lastkey()	returns integer code
length(s)	returns smallint
num_args()	returns smallint
scr_line()	returns smallint
set_count(n)	
showhelp(n)	displays help message n
sqlexit(n)	returns 0, after closing connection to database
startlog(s)	
upshift(s)	returns string with chars upshifted to uppercase

Operator	Comment
ascii(n)	returns a char, e.g. ascii(64) returns 'A'
date(s)	returns a date
date	returns a string e.g. Wed Aug 15 2006
day(d)	returns 1..31
extend(d or dt, format)	returns a date or datetime
field_touched(rec.field)	returns TRUE or FALSE
get_fldbuf(rec.field)	returns string contents of field
hex(n)	returns string e.g. 0x0000001c
in()	
infield(rec.field)	returns TRUE or FALSE
mdy(m,d,y)	returns date from args month, day, year
month(d or dt)	returns 1:12
ord(c)	returns smallint
time	returns string
today	returns date
year(date)	returns smallint

6.6.2 Standard 4GL Operators

The following functions are described by Informix 4GL as builtin operators. They work in Aubit4GL:

6.6.3 D4GL Builtin Functions

The following are do-nothing functions which allow 4J's D4GL programs to compile:

Function	Comment
ddeconnect()	
ddeexecute()	
ddefinish()	
ddefinishall()	
ddegeterror()	
ddepeek()	
ddepoke()	

6.6.4 Aubit Builtin Functions

Function	Return Values
_variable(name)	pointer to object (e.g. cursor, form, window, etc)
abs(n)	absolute value of n
a4gl_get_info("o","id","p")	See below
a4gl_get_page()	
a4gl_get_ui_mode()	0|1 (0=TUI, 1=GTK)
a4gl_run_gui()	
a4gl_set_page()	
a4gl_show_help(n)	
dbms_dialect()	"INFORMIX"|"POSTGRES"
fgl_buffertouched(s)	TRUE|FALSE
fgl_dialog_get_buffer()	string
fgl_dialog_getfieldname()	string
fgl_dialog_setbuffer(value)	
fgl_dialog_setcurrline(n)	
fgl_dialog_setkeylabel("key","label")	
fgl_getkey_wait(n)	
fgl_setkeylabel("key","label")	
fgl_prtscr()	
fgl_scr_size(srec)	returns int
fgl_set_arr_line(n)	
fgl_keysetlabel("key","label")	
fgl_set_scrline(n)	
fgl_strtosend(s)	returns string
fgl_winmessage(s)	
load_datatype(s)	
set_window_title(s)	
sqrt(n)	returns square root of n
winexec(s)	
winexecwait(s)	

6.6.5 a4gl_get_info()

Synopsis: a4gl_get_info("object", "id", "property")
where "object" in ("Form"|"Window"|"Connection"|"Statement")
and "id" is the quoted variable name of instance of the object
and property is an element of the set of properties of the object as follows:

In the properties below, replace the % with a value 1 .. maxelement.

6.6.5.1 Connection

Synopsis: a4gl_get_info("Connection", "", "Database")

Database in the only property available. The id argument is ignored.

6.6.5.2 Form

Form Property	Return Value
Database	char
Delimiters	char
ScreenRecordCount	int
ScreenRecordName%	int
AttributeCount	int
CurrentField	int
Width	int
Height	int
Field%	long?
ScreenName%	char
TableName%	char
AliasName%	char
FieldType%	char
FieldSize	int
FieldBytes	int
FieldDets	long
Screens	long

6.6.5.3 Statement

Atatement Property	Return Value
NoColumns	int
NoRows	int
Name%	char
Type%	char
Scale%	int
Nullable%	int
Length%	int

6.6.5.4 Window

Window Property	Return Value
Height	int
Width	int
BeginX	char
BeginY	char
Border	int
Metrics	int,int,int,int (x, y, h, w)

6.7 Form Syntax

```
DATABASE
   {database|FORMONLY}[WITHOUT NUKK INPUT]
SCREEN
{
   text[tag    ]
   ...
}
[TABLES name [,...]]
ATTRIBUTES
  tag=tagdescr
  ...
[INSTRUCTIONS
   [DELIMITERS "fl"
   [SCREEN RECORD name[[n]]
     ({tablename.*
       | tabname.colname THRU tabname.colname
       | tabname.colname}[,...])]]
```

In the SCREEN statement, the {} and [] are literal and do not indicate optional syntax.

6.7.1 Tag Description

```
tag=[table.]column[, attrlist];
tag=FORMONLY.field
   [TYPE [type|LIKE table.col]]
     [NOT NULL][, attrlist];
```

A tag's attrlist is a set of values:

```
AUTONEXT, COLOR=color [WHERE boolean],
COMMENTS="string", DEFAULT="value",
DISPLAY LIKE "table.col", DOWNSHIFT,
FORMAT="string", INCLUDE=( list ),
NOENTRY, PICTURE="string", PROGRAM="name",
REQUIRED, REVERSE, UPSHIFT, VERIFY,
VALIDATE LIKE table.col, WORDWRAP [COMPRESS]
```

6.8 Aubit4GL Builtins

6.8.1 a4gl_get_info()

This function is an Aubit4GL extension. It is used to obtain properties from opened forms, windows, cursors (not implemented yet), or from prepared statements, or from the current connection.

6.8.1.1 Synopsis

```
let h = a4gl_get_info("Window","mywin","Height")
let dbname = a4gl_get_info("Form","myform","Database")
let ncols = a4gl_get_info("Statement", "pquery", "NoColumns")
```

6.8.1.2 Input Parameters

The function always takes 3 parameters:

1. Object type: a char value: 1 of
   (a) "Form"
   (b) "Window"
   (c) "Connection"
   (d) "Statement"
   (e) "Cursor" (Not yet implemented)
2. Object pointer: the quoted variable id of the object (window, form, or whatever). In the case of parameter 1 being "Connection", this parameter is needed but ignored.
3. Property Name: e.g. "Database" or "Height". The list of acceptable properties depends on the object type of parameter 1.

6.8.1.3 Return value(s)

The return value is usually a single value (char, integer, or long) but 4 integers are returned if you invoke the call:

```
call a4gl_get_info("Window", "mywin", "Metrics")
    returning x, y, h, w
```

An example calling sequence:

```
define l_retval char(64)
open window mywin at 2,3 with form "myform"
let l_retval = a4gl_get_info("Window","mywin","Height")
```

In the above example, we are asking Aubit4GL to tell us the value of the Height property of the window mywin.

It is OK to assign the return value to a char variable (as in our example) because then 4GL will humour you by coercing numeric return values to type char.

6.8.1.4 Properties

Replace a trailing % with an index in the range 1 .. n where n is the number of columns or records or fields or whichever is appropriate.

6.8.1.5 Form Properties

Use 1 of the Form Properties below when calling:

```
open myform from "myform"
call a4gl_get_info("Form", "myform", ? ) returning ...
```

? will be one of:

Form Property	Return Type	Comment
Database	char	may be "formonly"
Delimiters	char	usually "[]"
ScreenRecordCount	int	
ScreenRecordName%	char	% is 1 of 1 .. ScreenRecordCount
FieldCount	int	
FieldName%	char	% is 1 of 1 .. FieldCount
AttributeCount	int	
CurrentField	int	
Width	int	
Height	int	
Field%	long?	
ScreenName%	char	
TableName%	char	
AliasName%	char	
FieldType%	int	
FieldSize%	int	
FieldBytes%	int	
FieldDets%	long	
Screens	long	

6.8.1.6 Statement Properties

Use 1 of the following Statement Properties when calling:

```
let l_query = "select * from customer"
prepare pquery from l_query
call a4gl_get_inf("Statement", "pquery", ? ) returning ...
```

? will be 1 of:

Statement Property	Return Type	Comment
NoColumns	int	
NoRows	int	
Name%	char	
Type%	char	
Scale%	int	
Nullable%	int	0 = false, 1 = true
Length%	int	

6.8.1.7 Window Properties

Use 1 of the following Windo Properties when calling

```
open window mywin at 2,3 with form "myform"
call a4gl_get_info("Window", "mywin", ? ) returning ...
```

Property	Return Type(s)	Comment
Height	int	
Width	int	
BeginX	int	
BeginY	int	
Border	int	0=no, 1=yes
Metrics	int,int,int,int	h,w,x,y

6.8.1.8 Connection Properties

The only property you can ask for is "Database". Parameter 2 is needed but ignored when you make the call:

```
let l_dbname = a4gl_get_info("Connection","","Database")
```

Aubit4GL will return the Database name of the current connection (if you have executed a 4GL database statement or a connect statement).

Note: There is a bug in versions of Aubit4GL versions up to 0.50.16 which prevents the above call from working. This is fixed in later versions.

6.8.1.9 Cursor Properties

The implementation of the "Cursor" set of calls to a4gl_get_info() is not yet done.

6.8.2 Comments

- The Type and Property parameters are case insensitive. e.g. "form" works as well as "Form", "database" is equivalent to "Database" etc.

6.8.3 Example

The following snipped is a working example of how the Properties with the trailing %s work. It depends on your having the traditional Informix stores database accessible:

```
database stores
main
        define l_sql char(64)
        define i, n int
        define l_namecol char(6)
        define la_name array[30] of char(64)
        let l_sql = "select * from customer"
        prepare p_sql from l_sql
        let n = a4gl_get_info("Statement","p_sql","NoColumns")
        for i = 1 to n
                let l_namecol = "Name", i using "<<"
                let l_namecol = l_namecol clipped
                let la_name[i] = a4gl_get_info("Statement", "p_sql", l_namecol )
                display i, ":", la_name[i]
        end for
end main
```

When you compile and run this, the output should be:

```
 1:customer_num
 2:fname
 3:lname
 4:company
 5:address1
 6:address2
 7:city
 8:state
 9:zipcode
10:phone
```

Note that we supplied the "Name%" as parameter 3 by constructing the values: "Name1", "Name2", ..., "Name10" within the FOR loop.

You can see that code like the above could be used in a Dynamic SQL application to discover the columns and their properties returned by an SQL query entered by the user at runtime. You can use these properties to label and format appropriately the rows returned by the cursor statement.

Chapter 7

Help system

7.1 Help message source file

Create your help files with a .msg extension using a text editor (e.g. vi)

A sample file:

```
.1
This is help message 1
.2
This is help message 2
```

7.2 Compiling help files

The syntax for compiling a help file (say myhelp.msg) into a binary help file (say myhelp.iem) is:
`amkmessage myhelp.msg myhelp.iem`
or
`amkmessage myhelp.msg > myhelp.iem`

Note that the syntax here is inconsistent with `fcompile` and `mcompile` in that you must supply the name (including suffix) of the target binary file. This is consistent with Informix's `mkmessage` program which has the same syntax.

7.3 help in programs

7.3.1 Within 4GL

```
CALL showhelp(3)
```

will display message number 3 from the current helpfile on the screen help line.

7.3.2 At runtime

The user presses the help key (default = CTRL-W) in any implemented command (Currently only menus have help support)

7.4 Decompiling

The command unmkmessage can be used to decompile an Informix compiled help file (usually with a .iem suffix) as follows:

```
unmkmessage myhelp.iem myhelp.msg
```

or

```
unmkmessage myhelp.iem > myhelp.msg
```

If you omit the 2nd filename, the unmkmessage program will output to the standard output stream (by default , your screen).

The unmkmessage program is useful when you lose or corrupt the source helpfile but still have the original binary.

7.5 Compatibility

The helpfile compiled by `amkmessage` is the same format as the IBM-Informix `mkmessage` program and the helpfiles will be compatible both source and binary.

7.6 mkmess

Note that amkmessage replaces the mkmess program used by earlier versions of Aubit 4GL. The 2 programs are incompatible. The older mkmess created binaries of a different format from the standard Informix .iem files.

Chapter 8

SQL Conversion

Aubit4GL allows you to connect to DBMSes (database management systems) from various vendors, as long the connection is via the SQL command language. Unfortunately, the syntax of the SQL language can differ considerably from one vendor to another, and often valid syntax for one DBMS fails when executed against some other DBMS. One way around this is to maintain different versions of your application, eg. one for use with Informix, another for running against Oracle, another for PostgreSQL, and so on. Another way is to replace each SQL command in your source code with a number of alternatives in a case statement, depending on the target database type. Either way, your code will be difficult to maintain and harder to read.

Aubit4GL resolves this by providing a module that lets you write code using just one version or "dialect" of SQL, and have this converted into the correct form for whatever database you connect to at run-time.

In order to do this, Aubit4GL needs to know the following:

- the source SQL dialect that your source code is written in
- the target SQL dialect expected by the currently connected DBMS
- rules on how to convert SQL commands between source and target forms.

8.1 Source SQL dialect

By default, the compiler assumes SQL is written using standard Informix syntax.

This can be changed by setting the environment variable `A4GL_SQLDIALECT`, or by setting the value of `SQLDIALECT` in the /etc/opt/aubit4gl/aubitrc file.

You can also change it at run-time using the SET SQL DIALECT command eg.
`SET SQL DIALECT TO ORACLE`

This will cause all subsequent statements to be treated as if they were written using Oracle syntax.

Note - the 4GL compiler is not guaranteed to handle commands using non-Informix syntax. If the compiler cannot understand a particular command, simply place it in a char variable (string), `PREPARE` it, and `EXECUTE` it.

8.2 Target SQL dialect

The database connection driver will inform Aubit4GL at run-time which dialect of SQL it speaks, so you do not have to configure this explicitly.

8.3 Configuration files

The syntax of an SQL command is converted from its source dialect to the DBMS' native dialect, by applying a number of transformations one after another on the SQL text.

For example, consider the steps taken to get the following Informix SQL statement to run correctly with PostgreSQL:

```
select last_name, first_name[1], (today-birthday)/365 age
from client
where last_name matches "M*"
```

1. replace double quotes with single quotes
2. replace `matches` with the regular expression operator `~`
3. use the function `substr()` instead of subscripting with []
4. replace the word today with `date(now())`
5. insert the word "AS" before the column alias age

The result is:

```
select last_name, substr(first_name,1,1), (date(now())-birthday)/365 AS age
from client
where last_name ~ '^M.*'
```

Special configuration files are used to indicate what conversions are needed.

They are located in the directory /opt/aubit4gl/etc/convertsql (this can be changed by setting the environment variable `A4GL_SQLCNVPATH` to an alternative location).

There is one file for each combination of source and target dialect, each file being named as source-target.cnv. For example, the rules for translating from Informix to PostgreSQL are in a file called INFORMIX-POSTGRESQL.cnv, in which the conversion rules for the above example are given as:

```
DOUBLE_TO_SINGLE_QUOTES
MATCHES_TO_REGEX
SUBSTRING_FUNCTION = substr
REPLACE today = date(now())
COLUMN_ALIAS_AS
```

8.4 Converting SQL scripts

Many 4GL programmers keep script files of SQL commands to be run through SQL command interpreters like isql, psql, etc., rather than via a 4GL program.

A command line utility, `convertsql` is available to convert these as well.

You may have to compile this program from source. Go to `/opt/aubit4g/tools/convertsql`, and follow the instructions in `README.txt`.

For example, to convert a file full of Informix SQL commands into SapDB compatible commands, you might execute:

```
convertsql INFORMIX SAPDB < mystuff.sql > mystuff2.sql
```

8.5 Conversion file syntax

The file contains a series of conversion directives, one to a line, with the following formats:

8.5.1 Simple directives

Simple directives take no arguments:

- DOUBLE_TO_SINGLE_QUOTES Change double quotes (") to single quotes (') around literal strings.
- MATCHES_TO_LIKE Change Informix_style 'matches' clause to one using 'like', and change * and ? to % and _ respectively. eg: matches 'X?Z*' -> like 'X_Z%'
- MATCHES_TO_REGEX Similar to 'matches-to-like' but uses the Postgres style regular expression syntax, eg: matches 'X?Z*' -> ~ '^X.Z.*'
- TABLE_ALIAS_AS Insert the word "as" before table alias names in a 'from' clause eg: from ..., table1 t1, ... -> from ..., table1 as t1, ...
- COLUMN_ALIAS_AS Insert the word "as" before column/expression alias names in a 'select' clause eg: select ..., sum(amount) amt, ...-> select ..., sum(amount) as amt, ...
- ANSI_UPDATE_SYNTAX Convert Informix-style "update ... set (..,..) = (..,..) " to the ANSI standard format "update ... set ...=..., ...=... " eg. update mytable set (col1,col2,col3) = ("01", "X", 104) where ...->update mytable set col1="01", col2="X", col3=104 where ...
- CONSTRAINT_NAME_AFTER Move the constraint name in a constraint command to after the constraint definition, eg: ... constraint c_name unique ->... unique constraint c_name
- CONSTRAINT_NAME_BEFORE Move the constraint name in a constraint command to before the constraint definition, eg: ... unique constraint c_name -> ... constraint c_name unique

8.5.2 Complex Directives

The following directive takes an argument (in the rules below, replace the word "string" with the appropriate values):

- SUBSTRING_FUNCTION = string Change Informix-style string subscripting to a function call, Replace 'string' with the name of the sql function. eg. where ... foo[3,5] = ... -> where ... substr(foo,3,3)

8.5.3 REPLACE directives

Search and replace is not case-sensitive. For legibility, lower case is used in the rules for search/replace strings to distinguish them from the keywords (in upper case).

You may leave the replacement string (after the = sign) blank. This will have the effect of removing the matched string from the converted output.

- REPLACE before = after Replace any occurrence of the string 'before' with 'after', eg.
  REPLACE rowid = oid
  REPLACE current year to second = sysdate
  REPLACE today = date(now())

- REPLACE_EXPR before = after Replace only if the 'before' text is found in an expression or where an expression is allowed, such as in a where clause or a select clause. eg.
  REPLACE_EXPR sysdate = current year to second
  REPLACE_EXPR today = date(now())

- REPLACE COMMAND before = after Replace, but only if the whole SQL statement matches the 'before' string eg.
  REPLACE_COMMAND set isolation to dirty read =

The example above has the effect of completely erasing the command.

Full list of available settings :

ADD_CASCADE

ADD_SESSION_TO_TEMP_TABLE

ANSI_UPDATE_SYNTAX

CHAR_TO_DATETIME

CHAR_TO_INTERVAL

COLUMN_ALIAS_AS

CONSTRAINT_NAME_AFTER

CONSTRAINT_NAME_BEFORE

DATETIME_EXTEND_FUNCTION

DOUBLE_TO_SINGLE_QUOTES

DTYPE_ALIAS

ESQL_AFTER_DELETE

ESQL_AFTER_INSERT

ESQL_AFTER_UPDATE

ESQL_UNLOAD

ESQL_UNLOAD_FULL_PATH

FAKE_IMMEDIATE

FULL_INSERT

IGNORE_CLOSE_ERROR

IGNORE_DTYPE_VARCHAR_MIN

IGNORE_OWNER

INSERT_ALIAS

INTERVAL_EXTEND_FUNCTION

LIMIT_LINE MATCHES_TO_GLOB

MATCHES_TO_LIKE

MATCHES_TO_REGEX

MONEY_AS_DECIMAL

MONEY_AS_MONEY

NO_DECLARE_INTO

NO_FETCH_WITHOUT_INTO

NO_ORDBY_INTO_TEMP

NO_OWNER_QUOTE

NO_PUT

NO_SELECT_WITHOUT_INTO

NO_SERIAL_START_VALUE

OMIT_INDEX_CLUSTER

OMIT_INDEX_ORDER

OMIT_NO_LOG

QUOTE_OWNER

RENAME_COLUMN_AS_ALTER_TABLE

RENAME_TABLE_AS_ALTER_TABLE

REPLACE

REPLACE_COMMAND

REPLACE_EXPR

REPLACE_SQLCONST

REPLACE_SQLFUNC

SELECT_INTO_TEMP_AS_CREATE_TEMP_AS

SELECT_INTO_TEMP_AS_DECLARE_GLOBAL

SIMPLE_GRANT_SELECT

SIMPLE_GRANT_UPDATE

SQL_CURRENT_FUNCTION

STRIP_ORDER_BY_INTO_TEMP

SUBSTRING_FUNCTION

SWAP_SQLCA62

TABLE_ALIAS_AS

TEMP_AS_DECLARE_GLOBAL TEMP_AS_TEMPORARY

USE_BINDING_FOR_PUT

USE_DATABASE_STMT

USE_INDICATOR

Chapter 9

Make

`make` is a command generator. It is used to automate the task of recompiling and relinking programs when you have altered a source file. Typically you create a file called Makefile or makefile (Makefile is preferred as it sorts higher in an ls listing of directory files) which contains information about which files depend on which others and lists the commands needed to create the object modules (.o files) and executable binaries.

Once you have your Makefile correctly written, whenever you want to recompile a program after changing a file, simply type:

```
make
```

and the minimum necessary compilation and linking will be done for you to produce the altered executable.

9.0.1 GNU make

This chapter gives some advice and examples for writing Makefiles for use with Aubit4GL. For documentation, ignore the O'Reilly book (which does not cover GNU make) but go the `www.gnu.org` website and read the online documentation there.

9.1 Makefiles

The following advice assumes that you are using GNU make (which has several constructs not available in other older versions of make).

9.1.0.1 Include File

Here is a sample set of definitions for an Aubit4GL Makefile:

```
# ---- Declare the following suffixes meaningful to make
.SUFFIXES: .afr .per
.SUFFIXES: .ao .4gl
.SUFFIXES: .iem .msg
# ---- Pattern rules for the above suffixes
```

```
%.afr : %.per   # equivalent to the old make form .per.afr:
(TAB)  aubit fcompile $<
%.ao : %.4gl
(TAB)  aubit 4glc -c $?
%.iem : %.msg
(TAB)  aubit amkmessage $< $@
```

These definitions should be put into a separate file (say `makedefs`) in the parent directory. You can then include the makedefs file in the Makefile itself with the statement:

```
include ../makedefs
```

The benefit of using include files in this way is that you avoid repetition of the included elements, and maintenance is reduced to a single file.

9.1.0.2 Make glossary:

$? = all the newer prerequisites (which need recompiling)

$@ = the current target (left of the : in the prereq line)

$< = the first of the newer prerequisitess. This is suitable when the command can only compile 1 file at a time (like aubit fcompile).

$^ = all the prereqs (not just the newer ones). Use this when you need to relink all the object modules.

$* = the stem (matching % in prereq line).

% = wildcard matches any sequence of zero+ chars. Note: the 2nd and subsequent % is the same sequence that the 1st % matched.

$?, $(?), and ${?} are all the same variable. If a variable has more than a 1 char identifier you must enclose the identifier in () or {}s

A modifier D, or F, can be used with $?, $@, $<, or $^ to return just the D(irectory part) or the F(ile part) of the filename.

e.g. if $? = `../lib/options.4gl` then

`$(?D)` = `../lib` and `$(?F)`= `options.4gl`

Note that these D and F modifiers are defined in make's built-in rules as:

```
?D=$(patsubst %/,%,$(dir $?))
?F=$(not-dir $?)
etc
```

The $(dir arg) and $(not-dir arg) macros are available for use with any variables whether user defined or builtin. Note that the $(?D) definition removes the trailing slash from the directory path (substituting %/ with %)

9.1.0.3 Makefile Example

```
#
GPATH = ../lib ../per
.PHONY: all
all: prog prog.iem prog.afr proga.afr prog.iem
srcfiles = prog0.4gl prog1.4gl prog2.4gl ../lib/options.4gl
objfiles = $(srcfiles:.4gl=.ao)
prog : $(objfiles)
(TAB) aubit 4glc -o $@ $^
# Note the subtle difference here $^ (all prereqs are needed)
#  $? would link only the newly compiled objects
```

The example file above is for a program consisting of 4 modules:

- prog0.4gl (containing the global ... end global statements)
- prog1.4gl (containing the main ... end main function and some general purpose functions)
- prog2.4gl (containing table specificated generated functions for Query, Add, Update, Delete, etc
- options.4gl for directing report output.

This structure was common with Fourgen generated programs.

9.1.1 Pattern Rules

Rules in Makefiles take the form:

```
target : prereq1 [[prereq2 ] ... ]
(TAB) command1
...
```

Note that the invisible tab is a crucial part of the syntax of Make. These sometimes get corrupted into spaces in ftp transfers - so be careful!. A make rule specifies that the target files depend on the listed prerequisite files and supplies the command that make should execute whenever a prerequisite file is newer (that is modified more recently than) the target file(s).

9.1.2 Make variables

In Makefiles like the above, we use variables `srcfiles` and `objfiles` to minimise the work of changing definitions. Note that the assignment to objfiles is done using a substitution expression (.ao replaces .4gl from the srcfiles list). If we add another library module to the srcfiles list (say `../lib/names.4gl`), no other change need be made to the Makefile.

Traditionally we have used uppercase for variable names in Makefiles. The GNU people now recommend that you use lowercase for better readablity.

9.1.3 GPATH and VPATH

Normally make will search only the current directory. If you want to force it to look elsewhere then you can set GPATH or VPATH to a list of search directories.

Directories listed in GPATH will we searched and the targets compiled into the remote directory.

Directories listed in VPATH will be searched but the targets compiled into the current directory.

In the example Makefile, options.ao will be compiled into ../lib/options.ao

9.1.4 .PHONY

Nearly all Makefiles have phoney targets: all, clean, install, and maybe others. GNU make allows you to declare these phoney targets (i.e. targets which are not real files to be built by commands). The benefit of doing this is the .PHONY declaration tells make to ignore any files called clean, install. etc. Omitting the .PHONY declaration might result in an accidentally created file called install, preventing make from executing the install commands.

9.1.5 Implicit rules

Note in the example that there is no specific rule for the help file and forms. These will be built by make using the make definitions we put into the include file. The targets: prog.iem, prog.afr, and proga.afr will be compiled using the %.iem : %.hlp and %.afr : %.per pattern rules in ../makedef.

9.1.6 Syntax

comments the hash symbol # comments out the rest of the line (i.e `make` ignores what follows the #).

quotes both single quotes ' and double quotes " are treated literally. Do not use them in Makefiles. In shell programs you use quotes to inhibit interpretation and the shell strips them from its input. `make` does not do anything special to quotes.

longlines break a long line by putting a backslash \ before the end of line. This will tell make to remove the backslash and the end of line, and interpret the result as a single line.

9.1.7 Debugging make

A botched Makefile can destroy your sourcefiles.

To help debug your Makefiles, use the -n and -p options.

-p will display all the rules (including the builtins) that make is using

-n will cause make not to actually execute the command but display them to the screen

Type the command:

```
make -np prog
```

will cause make to display all its definitions and rules, and to display all the commands it would run if you had typed the command: `make progAmake`

amake is an x4GL specific set of rules and tools for GNU "make"

Chapter 10

amake

10.0.1 Introduction

With Aubit 4gl compiler, compiling small program can be trivial:

```
aubit 4glpc *.4gl -o myprog
aubit 4glpc *.per
aubit amkmessage myhelp.msg myhelp.iem
```

Even with extra C code, it's still simple:

```
aubit 4glpc *.4gl myccode.c -o myprog -DAUBIT4GL
```

But, when you want to keep your make files compatible with Informix and 4Js compilers, have multiple program definitions in one directory, use pre-linked libraries, and be capable of compiling to P-code and C-code for each compiler, take care of installing and cleaning, it's not that simple any more.

10.0.2 Summary

When you need to create new make file to compile x4gl programs, you should use rules, headers and footer prototypes supplied with Aubit 4GL. Utility for running created make files, while not necessary, is also supplied, and can make your life a little easier.

For existing Informix 4gl and 4Js BDL/D4GL makefiles, I created a conversion system that will first create completely new set of make files from existing makefile (one per program) and then let you use it, in more or less same way we did so far, but erase most if not all of existing shortcomings. Old makefiles are preserved, so you can mix and match, if you really want to, but you won't need to.

10.0.3 Converting old makefiles

10.0.3.1 prepmake

run "prepmake" in the directory containing old make file, "makefile".

This will create file "makefile.prep" containing instruction needed for dumping program definitions to individual make files (*.mk). Note: this functionality depends on the fact that your existing makefiles use command "fgllink" or other 4gl compiler commands somewhere in each defined program target, and list all source files in dependencies. If for any reason this is not true for some makefile you want to process, look at the script, it should be easy to substitute this with some other present command.

Next, "prepmake" will first run "touch *.4gl" (to force all targets into thinking they need building) and then "make -f makefile.prep". This will create one make file for each program defined in makefile.prep, named as <program>.mk, using script "genmake". Each .mk files will contain definitions of include headers and footers, and names of source files needed to build that program, and nothing else. Like this:

10.0.3.2 example

```
include header.mki
PROG = P4E
GLOBALS.4gl = P4E.4gl
FILES.4gl = \
${GLOBALS.4gl} \
bankwind.4gl \
ckapwind.4gl \
ckhdwind.4gl \
secufunc.4gl \
vendwind.4gl
FILES.per = ${ALLFORMS.per}
include_footer.mki
```

10.0.3.3 amakeallo

`amakeallo` can be ued to rebuild all the .o object files in a Makefile.

10.0.3.4 amakeallf

`amakeallf` can be used to recompile all the .per form files in a Makefile.

Note: amake knows how to override `header.mki`, `footer.mki`, or both. You can also

override anything coming from header, and later, in footer, anything at all.

10.0.4 2. amake

Examples:

`aubit amake` # build default targets of all .mk files in ./

`aubit amake -k -all install` # install all programs, ignore errors

`aubit amake P11 aubit -k` # build aubit target defined in P11.mk, ignore errors

`aubit amake P11 aubit -defaultinclude#` build P11 target for Aubit compiler, use includes defined in P11

`aubit amake P11 -header myhead.mk` # default P11 target, use myhead.mk for header

`aubit amake --help` for full lost of flags and parameters.

10.0.5 Requests

Tell me if it's useful for you, if you need help, explanations, changes... If you make generally useful changes, I would like if you send them back to me. Latest version of these files will always available through Aubit 4gl CVS

10.0.6 Notes

- Most existing makefiles have no idea which file contains GLOBALS definitions; some compilers care, some don't. I assumed first source file listed in GLOBALS file, which can be wrong. If you step on this one, you'll need to find out manually which one is it actually. I guess it's more then possible to grep for "END GLOBALS" in "genmake" if we wanted to do that automatically.

- Some existing makefiles often don't have any references to form files, and even if they do, they have no idea which forms belong to which program. By default, I defined that each program needs all forms in current module. It would be wise to gradually replace this with actual forms needed. I guess that it should be possible to grep that from "genmake", since there we know all 4gl source files.

- You should consider this as technology demonstration. Some things are probably missing, or incorrect, in rules definitions and targets. But this is now so easy to fix, since it's all in one place that I did not worry too much. It compiled everything I tried. But I don't consider this finished code. It does what I needed, it may or may not do that same for you, but again, it's really easy to do anything in the way this is structured now. You should consult the "make" manual at http://www.gnu.org/manual/make-3.79.1/html_mono/make.html if you want to play with existing code.

- All "programs" that are nothing more then hard links, are ignored. This needs to be fixed in existing makefiles manually, unless someone can explain to me what's good about linking a program to a different name and then pretending it's something else. It won't work on Windows anyway, so if we want Windows compatibility, we cannot do it anyway.

- some of functionality depends on recent version of GNU make. If you don't have it, you'll need to download it from http://www.gnu.org. My version was 3.77. Current version as of time of writing was 3.79

- Most existing x4gl makefiles don't have any idea about help files. It should be possible to grep for this in "genmake".

- It's really easy to add functionality to do local check out, since now you can compile anywhere, even without any source files in local directory (amake/make will find them if they exist) This is closely related to the way that serious development should be organized using version control...

- Why one make file for one program? First, when more then one developer is working in same tree, it gives me the warm fussy feeling. Second, it makes target definitions cleaner, simpler, and easier to debug. Third, you can checkout your own make file to wherever you want, together with all sources needed for program. Or without them for that matter.

- Object libraries (.aox in Aubit, .42x in 4js dialect). I guess it should be possible to make attempt in automating this in "genmake", if we really want it. Related to this is an issue of how different 4gl compilers "strip" unused functions from executables. D4GL don't really care, since linking produces only a map file. i4gl does care, and Querix and Aubit, being C code translators, can easily strip executables.

- why is amake needed: actually, it's not, you can do "make -f 1.mk 2.mk params" or "make -f *.mk params" just fine, as long as you keep header and footer includes in each .mk file. It just makes things simpler, more flexible, and can replace headers on the fly.

10.0.7 Installation

(don't forget to convert back to UNIX file format if you are receiving this on Windows box; needless to say, scripts will need "chmod a+x")

These two should go somewhere in the path, but will probably be used only once:

prepmake - sh script to prepare original make file, created"makefile.prep"

genmake - sh script called from prepared makefile to create individual make files

Header will probably be most useful in your program directory, since it can contain module specific definitions, but one copy of general type should also probably be in /etc or /usr/incl:

header.mki - make file for including from each individual make file. It in turn includes a4gl.mk i4gl.mk q4gl.mk and d4gl.mk by default.

The Following files are supposed to be completely abstracted, so in /etc or /usr/include they go:

footer.mki - make targets definitions included from each individual makefile.

a4gl.mk - rules for compiling using Aubit 4gl compiler

i4gl.mk - rules for compiling using classic Informix 4gl compiler

d4gl.mk - rules for compiling using 4Js (Informix D4GL) 4gl compiler

q4gl.mk - rules for compiling using Querix 4gl compiler

And finally, this one should be in the path, probably in /bin:

amake - sh script used for executing make process, instead of the "make"command

10.0.8 Credits:

Thanks to Jonathan Leffler for Informix-4gl and 4Js rules, and general concept of how 4gl program should be processed by make. See www.informix.com/idn

10.0.9 #DEFINE

Note about using #DEFINE-style constructs, like C. There's nothing built into 4GL, but many people use the Unix "M4" command successfully. You could also use "cpp".

Stuart Kemp (stuart@cs.jcu.edu.au):

To use the C preprocessor (cpp) in conjunction with GNU make you might use a suffix of ".cpp" on the files you edit, and then build a Makefile containing:

```
.SUFFIXES: .4gi .4go .4gl .cpp .frm .per .cpp.4gl:
@echo Make $@ from $< $(CPPDEFS)
@$(CPP) $(CPPDEFS) $< > $@
.per.frm:
@echo Make $@ from $<
@form4gl -s $<
.4gl.4go:
@fglpc $<
```

Of course, the downside of this is that if you get an error-message when running your .4g[io] program, the line-number will be that in the .4gl file, not the .cpp file.

10.0.10 4GL Makefiles

There are no standard rules for how to organize Makefiles for 4gl. This note attempts to repair this deficiency for both Unix and NT systems.

10.0.10.1 Makefiles for Classic 4GL on Unix

Assuming that your version of MAKE understands the 'include' directive, a typical makefile will look rather like the file described earlier in thisdoccument. If your MAKE does not understand the 'include' directive, the simplest solution is to obtain a version of MAKE which does understand them.

One such MAKE is GNU Make, which is widely available on the Internet. See The GNU Project and the Free Software Foundation (FSF) for more information.

The rules file 'i4gl.mk' is located in some convenient directory. In the example, $HOME/etc is used, but a centralized location such as $AUBITDIR/incl, $INFORMIXDIR/etc or $FGLDIR/etc is a reasonable choice. Note that either the curly brackets or parentheses are required around the name of the environment variable in the makefile.

The macros list the components of the program, and the definitions of the lists avoid replicating names as much as possible, so that if a file is added, deleted or renamed, only one line in the makefile needs to be changed.

Note too that the current versions of i4gl.mk and d4gl.mk automatically provide definitions for the majority of the derived files, so the makefile itself does not have to define macros such as FILES.o or FILES.4ec. It must, however, define FILES.4gl for the I4GL source files, FILES.per for the form source files, and FILES.msg for the help source files, since these macros are used to define the other macros.

This makefile uses the 'standard' install script for Unix, and that means it can only install a single file at a time (an silly design decision, but one which was made so long ago that it cannot readily be changed). Consequently, we have to iterate over the list of form files. If there was more than one message file, we'd need to do the same for the message files.

The hard work in this makefile is the install and clean process. The actual compilation rules are minimal, occupying just six non-blank lines. There are some standard targets which are desirable in most makefiles. These include all to build everything that is needed by default, install to put the software in a location where it can be used, and clean to remove the debris from the development process.

As another pseudo-standard, if you are working with both Classic 4GL and Dynamic 4GL, or if you are using both p-code and c-code, it helps to standardize on some extra names. The makefiles illustrated here use:

- aubit Aubit 4gl c-code compilation
- i4gl-ccode Classic 4GL c-code compilation (I4GL)
- i4gl-pcode Classic 4GL p-code compilation (I4GL-RDS)
- d4gl-ccode Dynamic 4GL c-code compilation
- d4gl-pcode Dynamic 4GL p-code compilation
- i4gl Classic 4GL (both p-code and c-code)
- d4gl Dynamic 4GL (both p-code and c-code)
- querix Querix 4gl c-code compilation

These makefiles can also builds the custom I4GL p-code runner that is needed to run the program.

10.0.11 D4GL Makefiles on Unix

The rules for compiling D4GL are similar to the rules for compiling I4GL, but they use a different set of suffixes.

The first target in the makefile is 'default', and is what will be built if you simply type "make -f d4glonly.make". It is set up to build just the D4GL p-code program; to build the c-code program too, you have to specify "all" or "d4gl-ccode" on the command line.

This makefile builds a custom runner for D4GL because the code uses some C code. When you need a D4GL custom runner, you have to link with it too, so you have to build the custom runner before you try linking the program, and the dependencies ensure this happens automatically.

The rest of the makefile follows the pattern in the I4GL version, with the changes appropriate to handling D4GL instead of I4GL.

10.0.11.1 I4GL Makefiles on Unix

The actual rules for compiling Informix Classic 4GL are defined in the file i4gl.mk . There are a number of key things to note about them.

- The rules file does not reset the complete MAKE suffix list. Some versions of the file did, but this leads to problems when you try to add support for Dynamic 4GL as well; which file should be included first, and why, and so on. The down-side of being so accommodating is that if there is an intermediate ".c" file left over by a failed I4GL compilation, then that file will be compiled in preference to the ".4gl". To fix this, you have to nullify the suffix list and then reinstate the suffixes you want in the correct order (which means preferring the .4gl file to the .c file, and .ec files to .cfiles). However, it is difficult to write two separate files, i4gl.mk and d4gl.mk, which can be included in either order, and which don't repeat each others suffixes, if you also zero the suffix list in both files.
  I guess you could solve this if you defined I4GL.SUFFIXES and D4GL.SUFFIXES as macros, and had the line which re-instates the suffix rules specify both macros, even if one of them was actually empty (as it would be if you had not already processed the other rules file). A change for next month.

- The rules file does not define any targets, so that you can include it at the top of the makefile without altering the default target written in the makefile.

- The macro names are very consistent (arguably too consistent and not sufficiently mnemonic).

10.0.11.2 NMAKE

If you have Microsoft Visual Studio or Microsoft Visual C++ on your NT machine, you will have the NMAKE program available to you. You can use Makefiles patterned on the one shown below (from the D4GLDEMO program). Note that both the rules and the makefiles are much simpler on NT than on Unix because Classic 4GL is not available on NT, and neither is the Dynamic 4GL c-code compiler.

Some of the significant differences between MAKE on Unix and NMAKE on NT are:

- NMAKE does not accept ${MACRO}, but only $(MACRO).

- NMAKE does not accept a dot in macro names.

- NMAKE does not recognize 'null suffix' rules (for converting x.c into x, for example; it would only handle x.c to x.exe).

- Since there is no D4GL c-code compiler on NT, those rules in d4gl.mk are irrelevant.

- Since there is no I4GL c-code or p-code compiler on NT, the rules in i4gl.mk are irrelevant.

- There is no fglmkrun on NT.

- You have to be very careful about what you do with 'cd' commands. Typically,you have to do:
  cd with && $(MAKE) && cd .. POSIX.1 requires MAKE to accept both ${MACRO} and ${FILE.o}, unlike NMAKE.

- Since Unix versions of MAKE do accept the notations accepted by NMAKE, it would be possible, and possibly even sensible, to resign oneself to using the notation accepted by NMAKE in both the Unix and NT versions of the Classic 4GL and Dynamic 4GL makefiles and make rules. However, that also feels a bit like giving in to the school-yard bully, and that isn't really acceptable.

Prepared by: mailto:jleffler@informix.com

Last Updated: 1999-10-08

Edited by AF

10.0.12 Bug in ESQL/C rules:

Compiling ESQL/C code did not work because of macro name mismatches.

Specifically, there's a line that defines ESQL = ${ESQL_EC_ENV} ${ESQL_EC_CMD} ${ESQL_EC_FLAGS}
but the corresponding macros for compiling ESQL/C code use ${ESQL_EC} rather than ${ESQL}.
I concluded that I meant to define ESQL_EC, not ESQL.

For Aubit 4gl team,

Andrej Falout

Chapter 11

A4GL Utilities

11.1 adbschema

Generate a schema file representing tables and or procedures within a database. It can also produce sql scripts (or 4GL programs) for loading and unloading data to/from a database. This is useful when migrating from one RDBMS to another.

Usage :

```
adbschema [-noperms] [-fileschema] [-t tabname] [-s user] [-p user] [-r rolename] [-f procname]
-d dbname [-ss] [filename]
```

`-noperms` Do not include any GRANT/REVOKE

`-fileschema` Generate a schema suitable for the FILESCHEMA SQL Module

`-U` Generate unload statements

`-U4GL` Generate a 4GL program with unload statements

`-L` Generate load statements

`-L4GL` Generate a 4GL program with load statements

A typical example may be (assuming the database being migrated was called `customers`):

```
$ adbschema -q -noperms -d customers > customers.sql
$ convertsql INFORMIX POSTGRES < customers.sql > newdb.sql
$ adbschema -q -U4GL -d customers > unloadit.4gl
$ 4glpc unloadit.4gl -o unloadit
$ ./unloadit
$ adbschema -q -L4GL -d customers > loadit.4gl
```

(create database in new RDBMS and run the newdb.sql file to create the tables)

```
$ 4glpc loadit.4gl -o loadit
$ ./loadit
```

11.2 afinderr

Usage:

```
$ afinderr errornumber
```

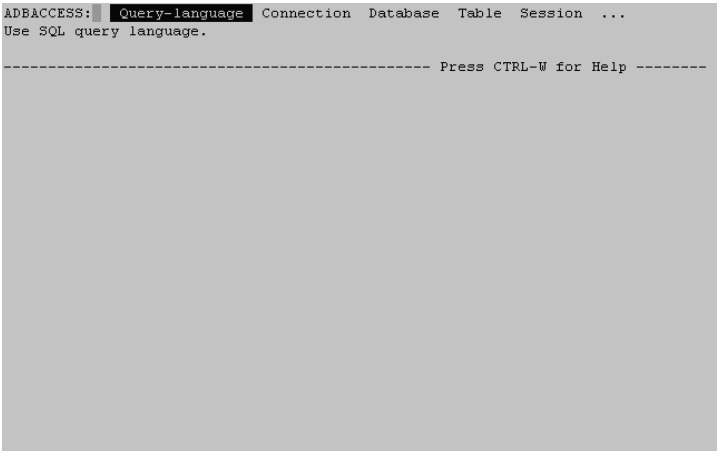
This will trawl through all of the message files in the `$AUBITDIR/etc` directory looking for any help messages associated with that help number. This is useful because the same error numbers could come from muliple places (eg. either Informix or Postgres) and hence may well have a different meaning.

11.3 asql

This is an workalike for Informix's `dbaccess` program. Several versions are required depending on the target database :

1. `asql_g.4ae` - Generic interface (For ODBC usage)
2. `asql_i.4ae` - Compiled using native Informix ESQL/C
3. `asql_p.4ae` - Compiled using native Postgres ecpg

When the program starts - you'll be presented with a menu :



```
NEW   : ESC    = Done editing      CTRL-A = Typeover/Insert    CTRL-R = Redraw
        CTRL-X = Delete character  CTRL-D = Delete rest of line

- Co:24/23 Ln:1/1 ------ test1@mike_2 ---------- Press CTRL-W for Help ------O-

select * from sometable

Database Opened
```

```
SQL:  New  Run  Modify  Use-editor  Output  Choose  Save  Info  Drop  Exit
Run the SQL commands

------------------------ test1@mike_2 ---------- Press CTRL-W for Help --------


   keyfield keydesc

          1 lock test 1

Q:0 1 - ( 1 Rows found )
```

The only major difference should be the Utilities menu - this provides access to some features which are present in the Informix `isql` tool which are not available in the `dbaccess` tool.

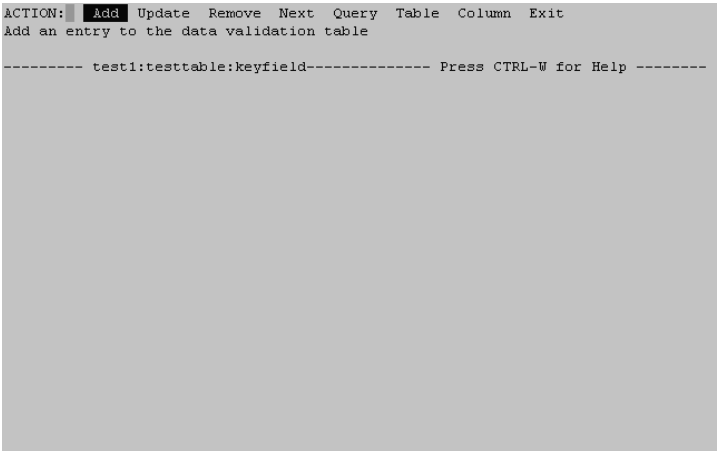
```
Utilities:  Form  Report  User-Menus  Exit
Form maintenance

------------------------ test1@mike_2 ---------- Press CTRL-W for Help --------

Q:0 1 - ( 1 Rows found )
```

11.3.1 runforms

This is a simple replacement for the sperform Informix utility which allows you to add, update and remove data from a table (or tables) via a simple form interface. `runforms` is used as the 'Run Form' option in the `asql` application.

11.4 aupscol

'aupscol' is a workalike for the Informix-4GL `upscol` utility. Using this you can specify default attributes and validation for use when forms are compiled.



11.5 P-Code

Aubit4GL includes an experimental PCode compiler. This will eventually allow you to compile your code into a portable bytecode. For now, only single module 4GL programs can be compiled as there is no facility to link 4GL modules. One interesting feature of the Aubit P-Code compiler is that it emulates the role of a very simplified C compiler. In this way it is still possible to use embedded C code within your 4GL program and have this work within the runtime environment (exceptions and limitations apply to the C code which can be embedded!)

Generic	4GL Specific	Description
c2pcode	c2pcode_fgl	Compiles a .c file
checker	checker_fgl	Dumps the contents of a compiled .c file
runner	runner_fgl	Runs the resulting file

11.6 configurator

The configurator allows you to view the various settings available within the Aubit4GL suite of programs. A brief summary is available in Appendix A.

11.7 convertsql

`convertsql` is a program which uses the SQL conversion rules used internally by the Aubit4GL compiler to convert the SQL of one RDBMS dialect to another. This is useful for converting

existing SQL scripts to run on a different server, for example, those generated by the `adbschema` program. The program always reads from the standard input, and writes to the standard output.

Usage:

```
convertsql source-sql-dialect target-sql-dialect
```

Note : Currently only 'Informix' is supported as a source dialect.

11.8 default_frm

`default_frm` will generate a default form for a table(s) specified on the command line.

Usage

```
default_frm -d dbname -t tabname [-t tabname ..] [-o outputfile]
```

If no output file is specified, then the output will be written to the standard output (ie normally the terminal)

Eg.

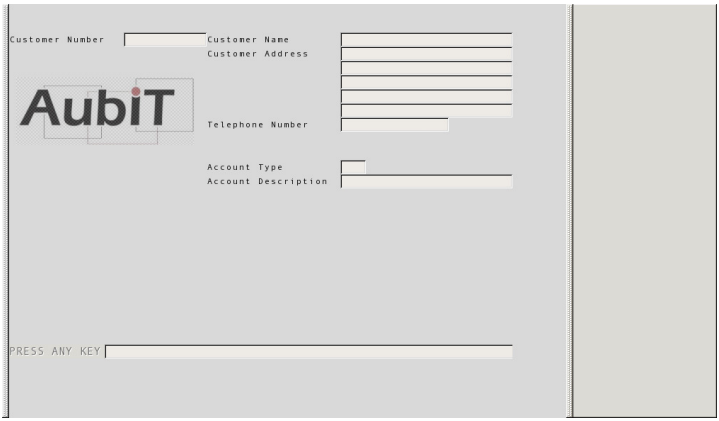
```
$ default_frm -d ordersnr -t customer
database ordersnr
screen
{
cust_no             [f000      ]
cust_name           [f001                    ]
cust_addr1          [f002                    ]
cust_addr2          [f003                    ]
cust_addr3          [f004                    ]
cust_addr4          [f005                    ]
cust_addr5          [f006                    ]
cust_tel            [f007           ]
account_type        [a]
}
end
tables
customer
attributes
f000 = customer.cust_no;
f001 = customer.cust_name;
f002 = customer.cust_addr1;
f003 = customer.cust_addr2;
f004 = customer.cust_addr3;
f005 = customer.cust_addr4;
f006 = customer.cust_addr5;
f007 = customer.cust_tel;
a = customer.account_type;
end
$
```

11.9 fshow

This is a very simple 4GL application which opens and displays the form specified on the command line. This is very useful for checking how a form will actually look from within a 4GL program (especially when using the GUI output).

Usage :

$ fshow formname



11.10 loadmap

This is a small 4GL application which can take the mapfiles generated by '4glpc –map' and load that information into a database. The sourcecode for this (tools/loadmap/loadmap.4gl) is meant to be a pro-forma for your own loadmap program.

11.11 mcompile

This is the menuhandler menu compiler. Menus created by this are nothing to do with standard 4GL menus, but are similar to those found in GUI applications. Support for these menus is temporarily suspended.

11.12 mkpackage

This program is for internal use. You can safely ignore it.

11.13 prepmake

A utillity script to convert makefiles to `amake` format

11.14 decompilers

Aubit4GL allows you to decompile most of the file formats which are compiled'(eg forms). The decompilers available are :

`unmkmessage` - message/help files

`mdecompile` - menu files

`fdecompile` - form files

11.15 Internally used applications

11.15.1 xgen

`xgen` is used interally as a replacement for the SUN RPC `rpcgen` program. This takes a '.x' description of data structures and generated the C code required to read and write those structures to disk. Internally, Aubit4GL makes use of .x files for describing forms, menus, and compiled P-Code.

The code generated by xgen is used by the generic packers to write the data in packed, memory packed, and gzip'd formats.

Chapter 12

Aubit4GL Extension libraries

12.1 channel

This library allows simple read/write access to files in a manner similar to that provided by some other 4GL vendors.

12.1.1 Dependencies

None

12.1.2 Function list

12.1.2.1 open_file

```
open_file(handle,filename,flag)
```

opens a file and associates 'handle' as a name for that file. flag is

'u' - use standard input/output

'r' - readonly

'w' - writeonly

'a' - append

12.1.2.2 open_pipe

```
open_pipe(handle,cmd,flag)
```

runs a command and allows reading/writing to that command via a pipe. flag is

'u' - input and output (*not implemented)

'w' - write only

'r' - read only

'a' - write only (*append is identical to 'write only' in this context)

12.1.2.3 set_delimiter

```
set_delimiter(handle,delimiter)
```

This sets the default field separator for a file

12.1.2.4 close

```
close(handle)
```

This closes the handle and associated file

12.1.2.5 fgl_read*

```
fgl_read(handle,nvars)
```

This reads from a file and returns each field as a separate return value. The number of values returned will therefore depend in the number of fields on each line of the file. This function is not part of the current implementation. As has been superceded by the 'read' function.

12.1.2.6 read

```
read(handle,variable)
```

or

```
read(handle,[variable list])
```

This reads the fields from the file (separated by the delimiter) and put that data into the variables passed to the read function.

eg. `read("file",[var1,var2,var3])`

12.1.2.7 write

```
write(handle,variable)
```

or

```
write(handle,[variable list])
```

eg. `write("file",[var1,var2,var3])`

This writes the data passed in to the specified file.

12.2 file

This is a library exposing various STDIO functions from the standard C library. Handles are all standard 4GL INTEGER's.

12.2.1 Dependencies

None

12.2.2 Function list

12.2.2.1 popen

Open a pipe for reading/write

```
popen(p_command,mode)
```

12.2.2.2 fopen

Open a file for reading or writing

```
fopen(filename,mode)
```

12.2.2.3 ftell

indicate the current position in a file

```
ftell(handle)
```

12.2.2.4 ferror

Tests if there is an error on a file handle

```
ferror(handle)
```

12.2.2.5 fseek

move the current position in a file

```
fseek(handle,n)
```

12.2.2.6 fseek_from_end

move the current position in a file, counting backwards from the end of the file

```
fseek_from_end(handle,n)
```

12.2.2.7 fsize

get the size of a file

```
fsize(handle)
```

12.2.2.8 fgets

read a string line from a file

```
fgets(handle)
```

12.2.2.9 feof

test if the position of the end of the file

```
feof(handle)
```

12.2.2.10 fclose

close the file associated with the handle

```
fclose(handle)
```

12.2.2.11 rewind

move the position back to the start of the file

```
rewind(handle)
```

12.3 memcached

This library allows access to memcached servers.

12.3.1 Dependencies

None. A specialized version of `libmemcache` (originally by Sean Chittenden) is included in the directory. Please see `memcache.c` and `memcache.h` for details

12.3.2 Function list

12.3.2.1 mc_new

```
mc_new()
```

12.3.2.2 mc_server_add

```
mc_server_add(lv_mc,lv_host,lv_port)
mc_server_add4(lv_mc,lv_host)
```

12.3.2.3 mc_add

```
mc_add(lv_mc, lv_key, lv_val, lv_bytes)
mc_add_str(lv_mc, lv_key, lv_val)
```

12.3.2.4 mc_replace

```
mc_replace(lv_mc, lv_key, lv_val, lv_bytes)
mc_replace_str(lv_mc, lv_key, lv_val)
```

12.3.2.5 mv_req_new

```
mc_req_new()
```

12.3.2.6 mv_req_add

```
mc_req_add(lv_req, lv_key)
```

12.3.2.7 mv_get

```
mc_get(lv_mc, lv_req)
```

12.3.2.8 mc_aget

```
mc_aget_str(lv_mc,lv_key)
mc_aget_rec(lv_mc,lv_key,lv_optr,lv_size)
```

12.3.2.9 mv_set

```
mc_set(lv_mc, lv_key, lv_val, lv_bytes)
mc_set_str(lv_mc, lv_key, lv_val)
```

12.3.2.10 mv_res_free

```
mc_res_free_on_delete(lv_res, lv_yesno)
mc_res_free(lv_req, lv_res)
```

12.3.2.11 mv_stats

```
mc_stats(lv_mc)
```

12.3.2.12 mv_delete

```
mc_delete(lv_mc, lv_key)
```

12.3.2.13 mc_incr

```
mc_incr(lv_mc, lv_key, lv_ival)
```

12.3.2.14 mc_decr

```
mc_decr(lv_mc, lv_key, lv_ival) mc_free(lv_mc)
```

12.4 pcre

This allows you to use perl style regular expressions within your 4GL program.

12.4.1 Dependancies

pcre - Perl Compatible Regular Expressions http://www.pcre.org/

12.4.2 Function list

12.4.2.1 pcre_text

Returns the matched portion of the string (up to 30 'portions' are stored)

```
pcre_text(i)
```

12.4.2.2 pcre_match

Indicate if the string 's' matches the regular expression 'p'

```
pcre_match(p,s)
```

Eg.

```
import package a4gl_pcre main
if pcre_match("cat|dog","There was an old cat") then
  display "Matches to ",pcre_text(1)
else
  display "No match"
end if
end main
```

12.5 pop

This module allows you to download and delete email from a pop3 server.

12.5.1 Dependancies

libspopc - `http://brouits.free.fr/libspopc/index.html`

12.5.2 Function list

12.5.2.1 popget

Get a portion of the header from a message, the 'From', To, subject,CC, date or size.

```
popget(lv_msg,which_info)
```

12.5.2.2 poperr

Returns the last error message from the POP3 server

```
poperr()
```

12.5.2.3 popbegin

```
popbegin(p_server,p_user,p_password)
```

begins a session, connecting to the server with the specified username and password

12.5.2.4 popnum

```
popnum()
```

12.5.2.5 popbytes

```
popbytes()
```

12.5.2.6 popmsgsize

```
popmsgsize(lv_msg)
```

12.5.2.7 popmsguid

```
popmsguid(lv_msg)
```

12.5.2.8 popgetmsg

```
popgetmsg(lv_msg)
```

12.5.2.9 popgethead

```
popgethead(lv_msg)
```

12.5.2.10 popcancel

```
popcancel()
```

12.5.2.11 popend

```
popend()
```

12.5.2.12 popdelmsg

```
popdelmsg(lv_msg)
```

12.6 smtp

This allows you to send email from your 4GL program. This module is also required if you wish to use 'REPORT TO EMAIL' from within your 4GL application.

12.6.1 Dependancies

A patched libsmtp - http://libsmtp.berlios.de.

12.6.2 Function list

12.6.2.1 set_errmsg

Displays the specified message and exits the program

```
set_errmsg(lv_msg)
```

12.6.2.2 clear_err

Clears down any active smtp error message/error number

```
clear_err()
```

12.6.2.3 set_server

Sets the name of the smtp server to use.

```
set_server(lv_server)
```

12.6.2.4 get_server

Gets the name of the smtp server to use. If no server has been specified via the set_server function, then the SMTP_SERVER environment variable will be used instead. If no SMTP_SERVER is specified, 'mail' is used.

```
get_server()
```

12.6.2.5 get_errmsg

Returns the last generated error message.

```
get_errmsg()
```

12.6.2.6 start_message

Start a new message session.

```
start_message(lv_sender,lv_subject)
```

12.6.2.7 add_recipient

Add a recipient to a message session.

```
add_recipient(lv_message,lv_to)
```

12.6.2.8 mime_type_new

Add a mime section to a message

```
mime_type_new(lv_message,lv_part,lv_mimetype)
mime_type_new_with_description(lv_message,lv_part,lv_mimetype,lv_description)
```

Normally - if you are using a mime email, you would add two initial sections a mixed part and a text part. You then add any files to the mixed part. Eg:

```
let lv_mixedpart = fgl_smtp::mime_type_new(lv_message,0,"multipart/mixed")
let lv_textpart =     fgl_smtp::mime_type_new(lv_message,lv_mixedpart,"text/plain")
if lv_rep_filename matches "*.pdf" or lv_hint="PDF" then
let   lv_pdfpart=fgl_smtp::mime_type_new_with_description(lv_message,lv_mixedpart,"application/pdf"
else
  let lv_reppart=fgl_smtp::mime_type_new_with_description(lv_message,lv_mixedpart,"text/html",lv_re
end if
```

12.6.2.9 connect

Connect to an smtp server with a given message to send, if no server is specified (ie its null or blank) then the server from `get_server` will be used.

```
connect(lv_message,lv_server,lv_port,lv_flags,lv_ismime)
```

12.6.2.10 disconnet

Closes the connection to the server and indicates that the email is complete and ready for transmision.

```
disconnect(lv_message)
```

12.6.2.11 send_to

Add additional recipients to an email message. the 'send_to' is identical to the 'add_recipient' function.

```
send_to(lv_message,lv_to)
send_to_cc(lv_message,lv_to)
send_to_bcc(lv_message,lv_to)
```

12.6.2.12 part_send_file

This is called to actually send the mime encoding of the file. the order in which these are used must match the order of the mime_type_new sections created previously.

```
part_send_file_html_listing(lv_message,lv_file,lv_last)
part_send_file(lv_message,lv_file,lv_last)
```

12.6.2.13 send_report

This is used by the 4GL library to send a report output (via the REPORT TO EMAIL) to the report recipients.

```
send_report(lv_hint,lv_rep_filename,lv_email_addr)
```

12.7 string

This module includes numerous string handling functions which may be useful from within a 4GL program.

12.7.1 Dependencies

None

12.7.2 Function list

12.7.2.1 split

Split a string into space separated fields

```
split(string,number_of_fields)
```

12.7.2.2 strstr

find the first location of a string within a string

```
strstr(haystack,needle)
```

12.7.2.3 strchr

find the first location of a character within a string

```
strchr(haystack,needle)
```

12.8 sxml

12.8.1 Dependencies

```
sxml - http://freshmeat.net/projects/sxml/
```

12.8.2 Function list

12.9 dynamic

12.9.1 Dependencies

None

12.9.2 Function list

This is a currently just list of all the Informix/4Js's Dynamic 4GL functions yet to be implemented...

Chapter 13

Aubit4GL Extensions

Aubit4GL fully implements the syntax of classic Informix 4GL v7.3. But further to that it has enhanced the language with many extra features.

13.1 Fake Comments {! ... !}

You can include A4GL extensions in your program code and still compile the source with Informix 4GL compilers by enclosing A4GL specific statements within the delimiters `{!` and `!}`. Aubit4GL will ignore the {! and !} delimiters and compile the code enclosed. Informix 4GL compiles will see the {! and !} as no different syntactically from { and } and will therefore treat enclosed code as a comment (and therefore not try to compile it). This allows you to write functions like the following:

```
function isaubit()
   {! return true !}
   return false
end function
```

13.2 Associative Arrays

This productivity enhancement will make complex Array manipulations easy and fast, and at the same time make your code easier to maintain and understand

13.3 Paused Screen Handling

This enhances usability over the slower connection lines, no matter which front-end implementation you deploy by selectively stopping updates to the screen. Using

SET PAUSE MODE ON

all screen updates are stopped, until a

SET PAUSE MODE OFF

is issued. This means that you can completely redraw the screen and then issue it to the user as a single screen rewrite, reducing cursor flicker as well as giving a much faster update.

13.4 ODBC Data access

ODBC compliance is a crucial feature for unprecedented connectivity and freedom of database options in the 4GL world.

13.5 Multiple Concurrent Connections

Based on the ODBC access concept, this feature will enable you to not only easily open several databases at the same time, and keep them open, but also to open several databases from several vendors from different servers, bringing together all database resources in corporate environments.

13.6 Application Constants

These are a small but effective contribution to creating error free programs that are easier to maintain and debug.

13.7 Map Files

These will for the first time enable you to have full overview of what your code is doing, how, and where. Indispensable for debugging and understanding unfamiliar code, and the behaviour of the compiler.

13.8 Variable IDs

No more hard-coded ID names! You can specify and reference all 4GL objects in the runtime! No more copy-and-paste code just to change ID names, resulting in higher productivity and code that is compact and easier to maintain.

13.9 Passing IDs

Passing IDs to functions is one of the implications of Variable IDs. It will allow you to name objects passed to functions, even in another module.

13.10 Embedded C code.

No more messing around with external C code, and no more complex make and link process. Just embed your C code inside your 4GL code, between the keywords CODE ... ENDCODE.

13.11 MOVE/SHOW/HIDE WINDOW

Enhanced windows manipulation resulting in more usable and flexible user interfaces.

13.12 WHENEVER SUCCESS/SQLSUCCESS

Will give you new options for conditional code execution, instead of always depending on error conditions.

13.13 Multilevel Menus

User interface enhancement that will make the coding and using applications faster and easier.

13.14 Extended DISPLAY ARRAY

Control providing many of features of INPUT ARRAY, and dynamically setting current and display lines of array. This will eliminate the need to use INPUT ARRAY logic where input is not needed, making the result safer and the code cleaner and easier to maintain.

13.15 Extended USING

Syntax provides more options for commonly used date formatting in reports and on the display, without the need to write additional code to handle this formatting, making especially report writing more productive.

13.16 Local functions

Defining a function to be local to the module opens possibilities fore some interesting and productive program structuring, and can also contribute to more easily maintainable and problem-free code.

13.17 get_info function

The get_info function will enable you to get almost all of the information about the state of the running program at runtime. It will allow you to write more flexible code than ever before, and achieve tasks that were simply not possible with other x4GL compilers.

13.18 Dynamic Screen Fields

These allow input fields to accept more data than will fit in the visible screen field size, making for more usable and flexible user interfaces.

13.19 Remote Function Calls

Will make x4GL applications for the first time enter the n-tier world. Running programs on the same or different machines, or even platforms, call each other to execute functions and return results. This can not only enhance typical 3-tier role separation, but also facilitate multi-processing on the level of the application, application partitioning on protocol level and enable weird things like accessing UNIX database from Windows PC that have no ODBC drivers for a specific platform....

13.20 SELECT/DELETE/UPDATE USING

By linking a record with a table and it's primary key, this extremely productive enhancement will automate and simplify multi-table data manipulation, the way it was always supposed to be, gaining productivity and maintainability in the ways you did not experience before.

13.21 ON ANY KEY

extremely useful with array manipulation, it will simplify user interaction logic in many places.

13.22 Compile Time Environment

This can override many library settings at compile time and will enable you to control compiler behaviour in ways not imaginable with other x4GL compilers

13.23 SET SESSION Option/SET CURSOR option

Thanks to the ODBC connectivity, it is possible to assign and change all attributes of database connection and defined database cursor at runtime, resulting in adjustable connection attributes at the same time exploring all the power of target ODBC driver and database from simple A4GL statements.

13.24 Application Partitioning

Thanks to user interface layer on one side, and ODBC layer on the other, and combined with RPC calling functionality, it is now possible to fully utilize all the resources of the enterprise environment, end-to-end, and deploy a4GL programs from one single computer, to hundreds of connected computers running different or same layers.

13.25 Y2K Runtime Translation

Two digit year support is implemented using run-time environment variable setting, enabling you to dynamically decide interpretation of year while preserving the code that was not written using 4 digit year functionality. Aubit 4GL is, of course, fully Y2K compliant.

13.26 Globbing

You can freely mix and use all IDs as module specific or global, allowing you do make distinction when naming ID's at runtime, thanks to 'Variable ID's' and the ability to pass ID's to functions as parameters. This functionality alone can save significant time in the coding process, and allow you to isolate ID related problems easily.

13.27 A4GL Wizard

13.27.1 Program Templates

These will allow the generation of full 4GL code for typical table oriented screens, just by specifying and compiling the template with a few simple definitions, much in the way that users used to use the Informix ISQL tool, but with full code generation and unprecedented flexibility, even to the point of direct inclusion in other 4GL programs.

13.28 PDF Reports

Built using PDFlib, allows you to produce reports in PDF format with fancy fonts.

13.29 GUI

Built using GTK+, this can allow normal 4GL programs to substitute a GUI version of the normal ASCII form based screens. Alternatively, you can exploit Aubit extensions to the classic language to create GTK widgets (e.g. cascading menus, pulldown lists, checkboxes, dialogues, etc.)

13.30 Packages

This is a feature borrowed from languages like Java, perl, and Python. It allows you to call functions from external libraries using normal CALL function() syntax.

13.31 IDE

13.31.1 Independent Development Environment

Written completely in 4GL, this application facilitates rapid development of any x4GL language application, while thanks to available source code remaining fully customizable using tools and language familiar to any 4GL language developer. FIXME: add JL's instructions to "Development Environment" page Please see appropriate sections of "A4GL enhancements to standard x4GL language for details of all features and syntax.

13.32 Logical Reports

These allow existing reports to be output as CSV, PDF or text files. These can be printed, saved to a file, etc - just like a normal 4GL report, and can also be automatically emailed to a recipient.

Chapter 14

Tricks, tips etc.

Chapter 15

Internationalisation

15.1 Auto-translation

Chapter 16

ACE reports

aace aace_4gl aace_perl

generate_aace

adecompile

Chapter 17

Aubit 4GL GUI

17.1 Plexus AD32 mode

An extension to A4GL is the ability to communicate using a Graphical User Interface. This is meant as a workalike for Plexus AD32. This is not meant for migrating normal 4GL applications to a GUI interface.

17.2 Aubit 4GL GUI mode

To compile a 4GL program with GTK and GUI support :
`aubit 4glpc -gtk filename.4gl -o filename`

(note: -gtk switch is now default for 4glpc)

This will generate a CUI/GUI switchable version. To use the GUI you must set the AUBITGUI environment variable:

`AUBITGUI=gtk export AUBITGUI`

Options for AUBITGUI are: text, curses, gtk, gui (not case sensitive).

You can then run it as normal (make sure you have the DISPLAY environment variable pointing to your X server).7.3 c

Notes:

- You must recompile any forms (these have changed!) The new forms will work for GUI and CUI modes whether you have compiled the 4GL with -gtk or not.
- Make sure that there is no form_x.h in lib/libincl (the makefile should remove this anyway)

Make sure that you're using the new libraries (ie if you've copied any to /usr/lib, /usr/local/lib etc. that these are updated).

17.2.1 Longer term

We'll need to add to assist.4gl to add in lots of useful functions - I've made a start - should be a few good examples to copy.

Client/Server mode - I've started to split the display bits from the rest of the library (look at lib/gui.c). This is used to do the redirecting from CUI to GUI modes. It should be possible to add to this to extend GUIs to non-GTK or remote displays.

VERY IMPORTANT : I've not tested lots of things and this is a very first draft.

Lots of things are not implemented yet - eg. attributes (colors, upshifts, formats etc) on input/display statements. I will ***need*** help to do all of these.

The way it works at the minute is really bad (uses #define to force a call to the GUI function instead of the CUI function.)

Eventually - I'd like to do a libtui (text mode), and get the calls routed via a variable (ui_mode). In that way you would compile the code in the same way (ie no -gtk), but have a command line/environment variable used to specify the runtime mode (Text or Graphical).

You might also want to set the environment variable 'NOCURSES' before running :
NOCURSES=1 export NOCURSES
./file

This will enable some of the output to be printed (There is some debugging stuff, GTK error messages and the output of "DISPLAY", and printf (if you use the embedded C code)) without turning on curses. (This isn't 100% effective yet.)

There are still a lot of things to work out (don't try CLEAR FORM/fieldname ...for example).

17.3 GUI Menus

There are two ways to use menus in GUI mode. The first is the traditional 4GL menu command in 4GL :

```
MENU ...
 COMMAND ...
 COMMAND ...
END MENU
```

This should work as before - but does not look very GUI. There is no support for drop down menus for example.

A GUI specific alternative is to use menu files.

Menu files have a couple of benefits :

- .They are the only way to get GUI looking drop down menus
- You can distribute different files, eg. in different languages (Only a very small benefit!)

Menu are loaded from this file using the 'SHOW MENU' 4GL command. eg.
`SHOW MENU my_menu USING my_menuhandler`

You should then have a `MENUHANDLER` function to deal with clicks on menu items:

```
MENUHANDLER my_menuhandler
DEFINE somevariables...
BEFORE SHOW MENU
ENABLE MENUITEM mn_1 #You can use MENUITEM or MENUITEMS here
ENABLE MENUITEMS mn_2,mn_3
DISABLE MENUITEM mn_1,mn_3
DISABLE MENUITEMS mn_1
ON mn_2
DISPLAY "Hello World"
ON mn_3
EXIT PROGRAM
END MENUHANDLER
```

By default menus are loaded from a file called `"menu.mnu"`, you can specify an alternate filename by using a `FROM` clause : eg.

```
SHOW MENU my_menu USING my_menuhandler FROM "myfile"
```

The .mnu will be automatically appended.

17.3.1 Menu File Format

Source menu files have the extension '.menu', which will be compiled to '.mnu' by the mcompile command.

FIXME: add to "using compilers"

$ mcompile filename

If no extension is specified .menu is assumed.

mcompile can also compile the menu file into C code which can be included in

the application directly (this means you don't need to distrubute the .mnu

file) using the -c option.

```
$ mcompile -c filename
```

This will generated a .c which can be linked into the application.

```
$ mcompile -c mymenu
```

will compile mymenu.menu into mymenu.c

```
$ mcompile mymenu
```

will compile mymenu.menu into mymenu.mnu

.menu_files

A menu file contains one or more MENUs. Each menu has an associated ID :

```
MENU file
....
END MENU
```

Within the menu you can place OPTIONs, these are specified as follows :

```
OPTION id "Caption"
```

You can also specify an image for an option in addition to the caption :

```
OPTION id Image="filename" "Caption"
```

Note : At present all images must be in .xpm format Although not fully implemented yet, you can also specify attributes for an option. Currently the only option allowed is 'RIGHT' [not implemented] which will right align the menu item (Often used for Help), although there may be others later...

```
OPTION id "Caption" ATTRIBUTES(RIGHT)
```

17.4 Simple GUI menu

A Typical file menu may look like this:

```
MENU file
OPTION mn_new Image="New.xpm" "New"
OPTION mn_open Image="Open.xpm" "Open"
OPTION mn_save "Save"
OPTION mn_saveas "Save As"
OPTION mn_exit "Exit"
END MENU
```

The above example will probably not appear as you'd imagine, the menu is displayed across the screen, what you'd normally have is a File menu, with the options listed as a drop down menu.

This is done using SUBMENUs. These have the same parameters as options : eg.

```
SUBMENU mn_file "_File" Image="file.xpm" .. ATTRIBUTE(RIGHT)
```

Options to submenus are listed between the `SUBMENU` and an `END SUBMENU` :

```
SUBMENU mn_zoom "Zoom"
OPTION mn_in "In"
OPTION mn_out "Out"
OPTION mn_fit "To Fit"
END SUBMENU
```

Additionally, because these may be reused, you can specify the ID of the SUBMENU:

```
SUBMENU mn_file "_File" USE file
```

In which case the compiler will substitute the options associated with a MENU with the specified ID. eg.

```
MENU file
OPTION mn_new "New"
OPTION mn_open "Open"
OPTION mn_save "Save"
OPTION mn_saveas "Save As"
OPTION mn_exit "Exit"
END MENU
MENU mymenu
SUBMENU mn_file "_File" USE file
END_MENU
```

Is the same as :

```
MENU mymenu
 SUBMENU mn_file "_File"
  OPTION mn_new "New"
  OPTION mn_open"Open"
  OPTION mn_save "Save"
  OPTION mn_saveas "Save As"
  OPTION mn_exit "Exit"
 END SUBMENU
END_MENU
```

You can also nest SUBMENUs

```
SUBMENU mn_edit "_Edit"
 OPTION mn_cut IMAGE="m1.xpm" "Cut"
 OPTION mn_copy Image="Copy.xpm" "Copy"
 OPTION mn_paste"Pastxxxxxxxxxxxxxxxxxxxxxxxxxe"
 SUBMENU mn_zoom "Zoom"
  OPTION mn_in "In"
  OPTION mn_out "In"
  OPTION mn_fit "To Fit"
 END SUBMENU
END_SUBMENU_____________________________________|
```

Again - these can be either direct (as in the above example) or SUBMENU .. USE.

A complete example :

```
MENU file
OPTION mn_new "New"
OPTION mn_open "Open"
OPTION mn_save "Save"
OPTION mn_saveas "Save As"
OPTION mn_exit "Exit"
END MENU
MENU mymenu
 SUBMENU mn_file "_File" USE file
  SUBMENU mn_edit "_Edit"
   OPTION mn_cut IMAGE="m1.xpm" "Cut"
   OPTION mn_copy Image="Copy.xpm" "Copy"
   OPTION mn_paste "Pastxxxxxxxxxxxxxxxxxxxxxxxxxe"
   SUBMENU mn_zoom "Zoom"
    OPTION mn_in "In"
    OPTION mn_out "Out"
    OPTION mn_fit "To Fit"
   END SUBMENU
  END SUBMENU
  SUBMENU mn_useful "Useful Stuff"
   OPTION mn_form "Open Window & Form"
   OPTION mn_sform "Open small Window & Form"
   OPTION mn_screen "Open Form on Screen"
   OPTION mn_lots "Open lots of windows"
   OPTION mn_loop "Loop windows"
   OPTION mn_closewin "Close windows"
  END SUBMENU
 OPTION mn_help "Help" ATTRIBUTES(RIGHT)
END_MENU
```

In this example - your program could 'SHOW MENU' mymenu or file.

17.4.1 Handling_menu_options

In code, options can be enabled or disabled using ENABLE MENUITEM id or DISABLE MENUITEM id

Note :

You can't use the 'SHOW OPTION "caption"' AND 'HIDE OPTION "caption"' ! These are for the traditional menu command.

17.5 GUI form files

New version of fcompile with minor changes to the original which can eventually be used for the text mode as well. It generates a slightly different output format which is currently incompatible with the old fcompile.

This includes extensions (which are present in the current fcompile in CVS) as well as a new one 'SCREEN TITLE'.

17.5.0.1 Extensions

`SCREEN TITLE`

fcompile can compile multiple screen sections into a single .per. Where more than one screen section is specified, the GTK GUI places each screen on a separate tab window (GtkNotebook). These will be labeled 'Screen n'. You can specify an alternate title with this extension : multi.per:

```
database formonly
screen title "Address" size 15 by 60
{
..
}
screen title "Contact" size 15 by 60
{..
}
screen title "Jobs" size 15 by 60
{..
}
```

17.5.1 WIDGET

`WIDGET` is a new parameter that specifies what should be place instead of an entry field (textbox). Currently this can contain "TEXT", "BUTTON", "CHECK" (checkbox), "LABEL" (text label - not editable), "PIXMAP" (picture - currently only xpm format handled), "COMBO" (combo box) , or "RADIO" (radio buttons).

More will be added when I get the chance!!

eg.

```
f001=formonly.fld1, WIDGET="CHECK";
```

17.5.2 CONFIG

When using a WIDGET, there are some specific things that may need setting - the CONFIG parameter is used to specify these. For all widgets you can specify a WIDTH and a HEIGHT (integers - in character spacing) If no width is specified - the size of the field on the form is used. If no height is specified a single character height will be used.

eg.

```
f001=formonly.fld1, WIDGET="BUTTON", CONFIG="WIDTH=5";
```

Some widgets require special config parameters, eg PIXMAP requires a FILENAME:

```
f001=formonly.fld1, WIDGET="PIXMAP", CONFIG="FILENAME='aubit.xpm'";
```

[When config parameters require strings, place them in single quotes.]

Some widgets have optional parameters :

```
f001=formonly.fld1, WIDGET="BUTTON", CONFIG="LABEL='OK'";
```

or

```
f001=formonly.fld1, WIDGET="BUTTON", CONFIG="IMAGE='okpic.xpm'";
WIDGET PIXMAP
CONFIG REQUIRED STR FILENAME filename (xpm format)
WIDGET BUTTON
CONFIG OPTIONAL STR LABEL label to use
CONFIG OPTIONAL STR IMAGE image to use (xpm format)
WIDGET RADIO
CONFIG REQUIRED INT NUM label to use
CONFIG REQUIRED STR Ln Label for button 'n'
CONFIG REQUIRED STR Vn Value for button 'n'
WIDGET ENTRY/TEXT/DEFAULT
CONFIG OPTIONAL INT MAXCHARS maximum field size
WIDGET LABEL
CONFIG REQUIRED STR CAPTION Caption for label
WIDGET CHECK
CONFIG OPTIONAL STR LABEL label for checkbox (may be clicked)
CONFIG OPTIONAL STR VALUE value for checkbox
WIDGET CHECK
CONFIG_NONE
```

17.6 gtk_form

Once you've compiled your form using the new fcompile, you can use gtk_form to show how it would look. There are no command line options on gtk_form, although gtk_init takes parameters (but I don't know what most of them do!).

There is also a GTKRC file which allows you to specify a scheme for example :

```
GTKRC=/usr/local/share/themes/Redmond95/gtk/gtkrc export GTKRC
```

You can select different themes by setting GTKRC under KDE (under GNOME I think this is automatic)

This program will not do anything useful once your form is displayed - but it should indicate when a field gains focus and when it is clicked or changed (depending on the widget type).

17.6.0.1 Examples_(in_test/gui/)

multi.per - An example of a multi screen form

radio.per - Radio buttons

widget.per - Many widgets on a single screen

17.7 GUI issues

FIXME: unsorted comments on GUI development:

Status:

So far I have a basic DISPLAY AT, DISPLAY .. TO ... and DISPLAY BY NAME. (90%)

Got HIDE WINDOW, SHOW WINDOW, MOVE WINDOW 100% complete

CURRENT WINDOW IS .. 100% complete (may need some attention later)

ENABLE/DISABLE fields.. 100% complete

Open Window 90% complete

Open Form 90% complete

Work on Menus and menuhandlers (90% complete)

Started on Input statement (40% complete)

Still have to deal with the modality issue, disabling, formhandlers, and the rest of the input & construct stuff..

Currently there is no support for 'SET PAUSE MODE ...' within the GTK stuff..

I'll_also_run_the_assumptions_by_you_:

- Opening a window in the old fashioned way will open a 'frame' in GTK on the 'screen' window.
- The menu command works pretty much the same as now (no multilevels, title to the left). Currently the only valid menu line is '1'
- All fields on a form will be disabled by default when the form is loaded (I may change this for PIXMAPs as they are dimmed when disabled and can't be activated anyway).
- Menu Handlers will load menus from a menu file, this allows for internationalisation and also allows descriptions of multilevel menus to be loaded at runtime. This uses an 'mcompile' command to compile into a runable menu (Details to follow).
- The only way to open a new X type window will be with the SHOW WINDOW xxx USING formhandler command. In this way - all current 4GL stuff should work and look the same as the text based one (with the additional widgets obviously)..
- Comments in forms and menus are displayed as tooltips rather than having a line of their own
- The DISPLAY AT is a bit dodgy - it works by putting a label at the specified position, if a label was started at that position it will be remove before the new one is created. if the text to display is "" then the label will be removed and no new label created. This may cause a problem with code like :
  DISPLAY "Hello World" at 1,1
  DISPLAY "World!" at 1,7
  Which would display "Hello World!" in the TUI mode, in GUI mode - you'd have two labels, one on top of the other, how it appears would depend on the type of font used, but it could be :
  Hello World!
  or
  HellWorld!
  (if the font is larger than my spacing)
  or
  Hello WWorld!
  (if the font is smaller)
  This will be a problem mainly with proportionaly spaced fonts. For courier and friends - we should be able to get the right spacings..

17.7.1 Colours in GUI

So far all colors and attributes (except border) are ignored. I need to investigate how to do this in GTK (any ideas ?)

17.7.2 Threads

So far I havn't needed to have any extra threads, although there are a couple of functions which may be called from 4GL, they will be something like :

```
FUNCTION gui_run_till_idle()
```

and

```
FUNCTION gui_run_till_terminate()
```

the 'till_idle" function will allow GTK to catch up with itself during complex programming. (At the minute all operations which do something to GTK, opening a window, displaying some text etc, run this after completing. the "till_terminate" function will run gtk_main() which is a loop that will stop the 4GL program terminating and should be the last function called in MAIN..END

MAIN if you use formhandlers or menuhanlders.

17.7.3 Progress

I'll try to get something posted up by the middle of the week. In the meantime - if anyone fancies helping - if you can brush up/read up on GTK it would be helpful, there will be a lot of 4GL functions that we will need to write to handle the nicer things (for example - I already have a 4GL function to set the window title which appears in the title bar...)

We'll need more for adding and maintaining list boxes etc.

Chapter 18

Extended Reports

18.1 PDF reports

18.1.1 Before you start

Aubit 4GL uses PDFLib to help generate the PDF output, you'll need a copy of this. NOTE : You must use a recent release of PDFLIB (available from http://www.pdflib.com).

You'll need to add '-DUSE_PDF_REPORTS' to the CFLAGS line at the top of lib/makefile.

Regenerate the files in lib (touch lib/pdf_reports.c;make) to include PDF generating capabilities.

18.2 Introduction

PDF reports are very similar to normal 4GL reports, but with added functionality. PDF reports are usually started with the

```
START REPORT repname TO "somefile.pdf"
```

This is because PDFs are read using Acrobat or some other pdf reader that requires a file.

To define a report as being a PDF report, you must use

```
PDFREPORT report_name(...)
```

instead of

```
REPORT report_name(...)
```

18.3 Output Section

The output section of a 4GL PDF report is slightly different to a normal report. It can have any of the following

```
LEFT MARGIN nval
RIGHT MARGIN nval
TOP MARGIN nval
BOTTOM MARGIN nval
PAGE LENGTH nval
PAGE WIDTH nval
FONT NAME "font"
FONT SIZE integer
PAPER SIZE IS A4
PAPER SIZE IS LETTER
PAPER SIZE IS LEGAL
REPORT TO "filename"
REPORT TO PIPE "progname"
```

nval can be any of the following :

```
n POINTS - PDF points 1/72 of an inch
n INCHES - Inches
n MM - metric mm
n
```

eg.

```
LEFT MARGIN 0.25 INCHES
RIGHT MARGIN 20 MM
PAGE LENGTH 60
COLUMN 10
```

When the units expression is omitted, n defaults to characters or lines (whichever is appropriate).

18.3.1 Fonts

The 4GL program will use the PDFLIB fonts. If the required fonts do not exist then the program will abort with a PDFLIB error.

NOTE : Case is sensitive for these font names!

Eg.

```
FONT NAME "Times-Roman"
```

or

```
FONT NAME "Helvetica"
```

18.3.2 Report Structure

The report structure will be identical to that of a normal 4GL report.

18.3.3 Extras

In order to generate 'nice' reports - there are a couple of extra features available.

18.3.3.1 Positioning

You can use the normal column and skip positioning mechanisms. You can use the nval values for column

eg

```
print column 1.1 inches,"Hello World"
```

but you have to use 'skip by' for nval movements :

Eg.

```
skip by 2 inches
```

Also you now have a 'skip to' which allows you to move to an absolute position within the current page (including backwards).

Eg.

```
skip to 2 inches
```

18.3.3.2 Using pdf_function()

This allows you to control certain aspects of the PDF report, changing fonts etc. The first argument is the operation type, this will be :

Argument Indirectly calls

```
set_font_name -> PDF_setfont
set_font_size -> PDF_setfont
set_parameter -> PDF_set_parameter set_value_->_PDF_set_value
```

Check the PDFlib manual for these

Eg.

```
call pdf_function("set_font_name","Times-Roman")
call pdf_function("set_font_size",30)
call_pdf_function("set_parameter","underline","true");
```

18.3.3.3 Images

It is also possible to include an image within the PDF report, this is done using the 'PRINT IMAGE' statement with a blob variable containing an image. The image must be a GIF,PNG, TIFF or JPEG and this type must be specified when displaying the image, this is done using the 'AS ...' keyword, ie "AS GIF", "ASTIFF", "AS PNG", "AS JPEG".

Finally - the image can be scaled when it is displayed. This can be either a single value (ie scaling x & y by the same value) or two (specifying the scaling for x & y separately)

```
|print image some_blob_var as png
print image some_blob_var as gif scaled by 0.5.7,0.8
```

18.3.4 Example program

Please see pfd_example.4gl in test/

18.4 Printing generated reports

What's really nice is that with most Linux distributions include the ability to print PDF / PS files direct to the printer!

From http://www.apsfilter.org/ :

> "Apsfilter supports PS (Postscript) printer and non-PS capable printer by using Ghostscript as PS emulator. So if you have a non-expensive color DeskJet printer, Apsfilter and Ghostscript enhance your printers capabilities, that you'll get a Color Postscript Printer in return for free ! "

This one is especially important for running Aubit on any kind of remote display:

"Printing on locally connected, network printer, as well as on Unix-, Windows-and AppleTalk remote printer is supported. "

From http://www.linuxprinting.org/howto/setup.html

> "apsfilter is a filter designed for use on a wide variety of Unices. It supports essentially all Ghostscript drivers. It, too, works with various strains of LPD, including stock BSD and LPRng. At the moment, this is probably the best third-party system around for non-PostScript printers"

Apsfilter V 6.0 Filetype Support lists PDF, and many other file types supported: http://www.apsfilter.org/filetype

Also see http://www.cups.org/ :

"A UNIX printing system (with sample drivers for HP, EPSON, and OKIDATA printers) based on the Internet Printing Protocol. CUPS is the basis for ESP Print Pro and is being considered as the standard printing system for a number of commercial and free UNIX operating systems. CUPS is provided under GNU GPLand LGPL. "

Chapter 19

Logical Reports

Logical reports take the print statements in an unmodified REPORT and log whats printed and the section in which its printed to a meta data file.

19.1 Invoking a logical report

The report 'function' is unchanged - but the calling procedure is enhanced to include :

```
START REPORT report-name TO CONVERTABLE
```

as well as the familiar TO PIPE/TO FILE etc.

This creates the meta data file (in /tmp) which can be processed later.

19.1.1 'Finishing' the report

The processing is done via the "FINISH REPORT" statement, "CONVERT REPORT" statement or via an external program "process_report".

The enhanced FINISH REPORT now accepts the following syntax :

```
FINISH REPORT report-name CONVERTING TO "filename" AS "type" [ USING "layout" ]
```

(You can also CONVERTING TO PRINTER, TO PIPE)

```
FINISH REPORT report-name CONVERTING TO EMAIL AS "type" [ USING "layout" ]
```

```
FINISH REPORT report-name CONVERTING TO MANY
```

19.1.2 Converting to "filename"

"type" can be any one of the conversions available on the system.

These are in $AUBITDIR/lib called libLOGREPPROC_*.[so/dll]

On an average system you may have :

libLOGREPPROC_CSV.so libLOGREPPROC_PDF.so libLOGREPPROC_TXT.so

This means you can process types of "CSV", "TXT" or "PDF" A special name of "SAVE" can also be used which copied the data verbatim from the meta data file into the filename specified. This file can then be used with the layout editor and/or the process_report program.

If USING "layout" is omitted a default layout will be used.

19.1.3 Default layouts

For PDF and TXT it is safe to setup a default layout.

These can be put in the $AUBITDIR/etc directory and have a `.lrf` extension. The filename is made up of combinations of program/module/report name and the width of the page (<=80 = narrow <=132 = normal >132 = wide). The search order is complex - but basically it depends on :

1. program_module_report_type.lrf
2. program_report_type.lrf
3. program_module_type.lrf
4. module_report_type.lrf
5. report_type.lrf
6. module_type.lrf
7. program_type.lrf

If none of these is found - then it looks for :

1. default_type_narrow.lrf
2. default_type_normal.lrf
3. default_type_wide.lrf

dependingon the width

Finally - it will use

1. default_type.lrf

(Where type is PDF, TXT or CSV for example)

To create one of these defaults - use layout_engine (for PDF and TXT, you can edit using any meta data file as an input)

19.1.4 Converting to many

This allows multiple conversions. The meta data file is not automatically deleted so it is possible to use the same meta data to generate a text file, CSV output and PDF if required.

To do this you need to use the CONVERT statement

CONVERT REPORT rep-name TO "filename" AS "type" USING "layout"

again - USING "layout" is omitted, one will be generated automatically..

Once you've done all your conversions, free report will delete the meta data.

Examples :

```
start report r1 to convertable

output to report r1 (1,2)

finish report r1 converting to "myfile1.pdf" AS "PDF" using "layout1"

start report r1 to convertable
```

```
output to report r1 (1,2)

finish report r1 converting to "myfile2.pdf" AS "PDF" # uses default layout

start report r1 to convertable

output to report r1 (1,2)

finish report r1 converting to many

convert report r1 to "orig.output" AS "SAVE"

convert report r1 to "myfile3.pdf" AS "PDF"

convert report r1 to "myfile4.txt" AS "TXT"

free report r1
```

19.2 Saved Meta Data

There are 3 things you can use with the meta data

19.2.1 The Report Viewer

This is a GTK2.0 application which displays the contents of the report in a tab'd window (one tab per page) - you can't print or anything, but its useful to see whats been put out in the meta data file and is used as the basis of the next app..

By default - it will only show you the first 10 and last 10 pages (if your report is only 5 pages long - you'll still only see 5 pages!) - this is basically to limit the impact of a very large report in terms of creating GTK widgets!

You can change this by changing the MAX_PAGE and MIN_PAGE in report_viewer/report_viewer.

Invoke using :

```
$ report_viewer filename
```

Where filename is the meta data file (ie the START REPORT TO "filename")

You will probably note that you can click on sections of the report and they change colour. These define the printed elements. When you click on an 'element' everything that the report considers to be printed in the same place in your 4GL (not based on lines/columns etc) is highlighted..

Also - there is a series of '>' going down the left hand side - these indicate the block in which those elements are printed. Again clicking on one of these highlights all lines printed within that block (i have not done anything about have a print ... ; in an after then have a print in an on every row etc)

There is some debugging stuff which is printed to stdout (ie from the window you ran the application from) which will eventually be removed...

19.2.2 The layout editor

This is another GTK2 application which embeds the report viewer and allows you to edit a logical report output.

ATM - there are only two coded report output types :

CSV and TXT

Although all of the code has been abstracted into shared libraries :

libLOGREP_???.so

You can't edit the TXT layout at all - so you get a 'no configurable options' for that.

For CSV mode - the libLOGREP creates a series of tables - one for each block which has seen something printed in the output... (Eg before group/after group/ page header/on every row)

You can then drag&drop information from the report viewer into these tables to generate the report layout. Double clicking a cell removes the contents of that cell..

At the minute you are limited to 10 cells across - this will be changed to use a spin button like the number of lines...

You can use the 'Default' menu option to create you a default layout which you can then play with.

Unfortunately - the layout is indicated by using the block and entry ID for the printed output - so you'll see things like "0/1", "1/4" in the layout editor - if you want to see what they represent, a single click will highlight that section on the report viewer...

You can load a layout using the menu option.. When you're happy - save the file using the menu options...

Invocation :

```
$ layout_engine type filename
```

Where type is TXT or CSV (more to be added later!) and filename is the original 4GL report output (just like for the report_viewer)

eg

```
$ layout_engine CSV /tmp/r1.out
```

You can't change the report you're editing or the type from within the layout engine - you'll need to start it again.

All load/saves within the layout editor refer to the layout file - not the meta data report file!!!

An extension .lrf (Logical Report Format) is used when it thinks its required...

19.2.3 The report processor

This a text mode application which takes a report meta data file and a report type and renders the report to the required output type with an optional layout file...

This is abstracted behind a shared library just like the report layout but its called libLOGREP-PROC_?.so

If no layout file is supplied then a default one is generated before the report is processed...

Invocation

```
$ process_report type filename or $ process_report type filename layoutfile
```

The output is currently stored in a temporary file (the name of which is displayed when the process completes)

19.2.4 Tips for CSV layouts

Copy to the same block type - the only exception might be for BEFORE GROUP OF to duplicate these details in an on every row..

19.3 Helper programs

process_report

report_viewer

layout_engine

Chapter 20

Debugging

Aubit4GL is a 'live' project. It is in pretty constant development, both adding new features and fixing any issues as they come along. It is important to understand that while considerable effort has been made to remove any bugs in the Aubit4GL, as with all code - some will remain. It is therefore essential that these bugs are reported back in the most efficient manner so that that can be fixed promptly.

20.1 Coredumps

To find the reason for core dumps, create debuggable files!

If 4glc (or fcompile etc) is core dumping - then recompile them to have debugging information included, to do this:

set the CFLAGS in incl/Makefile-common to have a `-g`, and compile recompile the relevant Aubit application.

If a compiled 4gl application is core dumping, then compile that with -g (4glpc -g ...) so that we have a debuggable 4gl executable.

Next, run the core-dumping application through gdb, when it dumps core do a `bt` in gdb..

```
aubit 4glc -g hello.4gl
```

```
gdb 4glc core
```

Now type `bt` inside gdb - that will give you a backtrace (with any luck).

20.2 Unexpected behaviour

If an application is failing in some way, the best thing to do is to create and examine (or ask those on the aubit4gl mailing lists to examine) a debug file.

This is created by setting

```
$ export DEBUG=ALL
```

You can then run your application and it should generate a file called 'debug.out'. This file can get huge very quickly, though only the last 100 or so lines will normally be needed to see whats wrong.

You may also find it useful to compile using the `-g` option and run it through the `gdb` debugger.

20.3 All other errors

When a 4GL module or form is compiled, the compiler will generate a .err file if the compilation is not successful.

20.4 compiler errors

20.5 Reporting bugs

For most cases, the simplest way to report a bug is to generate a test case. This is the minimal amount of code required to reproduce the bug. This may entail forms etc which should be included. Once a test case has been generated, either post it to one of the Aubit4GL mailing lists, or create an account on the Mantis Bug Tracking system at www.aubit.com/mantis and enter it there.

Chapter 21

Revisions

21.1 2006-8-1

- Further elaboration of builtin functions
- Folded get_info() documention into this manual
- Change of syntax (use . instead of ::) for extended library package calls
- More info about libraries, especially libchannel
- Put a Quick Reference section into the Language Chapter

21.2 2005-9-9

Just editing changes:

- Fix numerous spelling mistakes
- Fix some infelicities of English expression
- Fix punctuation, syntax, and some grammatical errors

21.3 2005-3-12

Extensive new material from Mike Aubury

- Quick Installation
- Elaboration of combinations of Informix/PostgreSQL with EC/C
- Troubleshooting
- Details of 4glpc and 4glc compiler flags
- Utilities:adbschema, adbaccess, asql, etc
- Extension Libraries: channel, file, etc
- Debugging
- Aubit 4GL GTK GUI development

21.4 2004-4-27

- Chapter 2: further information about PostgreSQL and in particular the gborg.postgresql.org project
- Chapter 5: include Mike's documentation on IMPORT PACKAGE packagename

21.5 2004-2-22

- Some tidying of chapters 1-3
- LyX preamble now sets up PDF properties: pdfinfo, pdfcatalog. You can navigate with Table of contents (bookmarks) on the left under Acroread now.
- HTML version now shows section numbering.

21.6 Problems

- Tables bug in latex2html is now fixed (thanks to Ross Moore of MacQuarie University)
- Stylesheets still not right (latex2html configuration problem?)

Appendix A

UNIX environment variables

The following list of environment variables was derived from the configurator program's description file.

A4GL_AUTOBANG=YES|NO UI/TUI/MENU
: Enable automatic ! for command entry(like dbaccess menus) for all applicable statements

A4GL_CINT COMPILE/RUNTIME
: Full path to CINT C-code interpreter, if installed, othewise 'no'. Used by 4glc compiler to run C compiled code after compilation.

A4GL_C_COMP COMPILE
: Name of the executable of C compiler to use. Note that 4glpc uses $CC

A4GL_EXE_EXT COMPILE
: Extension to use for executable files compiled by Aubit compiler. Aubit default extensions for compiled resources (forms,menus,help) and objects as used by Amake and Aubit compiler (see resources.c) Amake does NOT read this file (?-check) note that composite variables A4GL_FRM_EXT and A4GL_MNU_EXT exist only in/for Amake defaults:
A4GL_MNU_BASE_EXT=.mnu
A4GL_HLP_EXT=.hlp
A4GL_FRM_BASE_EXT=.afr
A4GL_XML_EXT=.xml
A4GL_PACKED_EXT=.dat
A4GL_OBJ_EXT=.ao
A4GL_LIB_EXT=.aox
A4GL_SOB_EXT=.aso
A4GL_SOL_EXT=.asx
A4GL_EXE_EXT=.4ae
To emulate Informix p-code extensions (for instance, to re-use legacy make files) you would use this settings; note that doing this is not recomended and that created files will still be in Aubit format, not Informix one:
A4GL_MNU_EXT=<no equivalent>
A4GL_HLP_EXT=.iem
A4GL_FRM_BASE_EXT=.frm
A4GL_XML_EXT=”
A4GL_PACKED_EXT=”
A4GL_OBJ_EXT=.4go

A4GL_LIB_EXT=<no (standard) equivalent>
A4GL_SOB_EXT=<no equivalent>
A4GL_SOL_EXT=<no equivalent>
A4GL_EXE_EXT=.4gi

A4GL_FORMTYPE FORMS/RUNTIME
Determine which runtime library to use for reading forms $AUBITDIR/lib/libFORM_?.so
Default forms driver to be loaded When used: run-time only
Options: (GENERIC), NOFORM, XDR
Generic implies that format specified with A4GL_PACKER will be used

A4GL_FRM_BASE_EXT RUNTIME/COMPILE/FORMS
Default form extension (for all packing types)

A4GL_HELPTYPE HELP/RUNTIME
Determine which runtime library to use for displaying help messages $AUBITDIR/lib/libHELP_?.so

A4GL_HLP_EXT HELP/RUNTIME/COMPILE
Specify the default extension for a help file

A4GL_INIFILE COMPILE/RUNTIME
Environment variable optionaly specifiying aubitrc file to use

A4GL_LEXDIALECT ESQL/COMPILE
Determine which ESQL/C dialect to use $AUBITDIR/lib/libESQL_?.so When A4GL_LEXTYPE=E
specify type of EC compiler to be used. Ignored if A4GL_LEXTYPE is not set to
EC When used: compile-time only
Options: (INFORMIX), POSTGRES, SAPDB, QUERIX

A4GL_LEXTYPE COMPILE
Determine what language to convert the 4GL code into $AUBITDIR/lib/libLEX_?.so
Default output language driver for 4gl compiler: When used: compile-time only
Options: (C), PERL, EC, CS
Note CS means C#
Note: EC (Embedded SQL C) can be Informix ESQL/C, SAP DB pre-compiler, Querix esqlc or PostgreSQL ecpg. Using EC will limit Aubit DB connectivity at run-time to that of used EC compiler, ignoring setting of A4GL_SQLTYPE

A4GL_LIB_EXT COMPILE
Extension to use for libraries created by Aubit compiler

A4GL_LINK_LIBS COMPILE
Libraries to link against when producing executables

A4GL_MENUTYPE MENU/COMPILE/RUNTIME
Determine library to use for menuhandlers (not normal 4GL menus) $AUBITDIR/lib/libMENU_?.so
Default menu driver to be loaded: When used: run-time only
Options: (NOMENU), XDR, GENERIC
Generic implies that format specified with A4GL_PACKER will be used

A4GL_MNU_BASE_EXT COMPILE/RUNTIME/MENU
Base extension for compiled menu files Base extension (without packer extension) to use when compiling/opening menu files

A4GL_MSGTYPE HELP/RUNTIME
Determine library for help message handling $AUBITDIR/lib/libMSG_?.so Default help message driver to be loaded: When used: run-time only
Options: (NATIVE), XML (??? XML? check this!)

A4GL_MV_CMD COMPILE
Command to ise to move files on the file system

A4GL_OBJ_EXT COMPILE extension to use when compiling 4GL modules to objects

A4GL_OMIT_NO_LOG

A4GL_PACKED_EXT COMPILE/RUNTIME
Determine file extension for packing

A4GL_PACKER MENU/FORMS/HELP/COMPILE/RUNTIME
Determine library for packing forms/menus/help etc $AUBITDIR/lib/libPACKER_?.so
You can select which packer to use
Options:(PACKED),XDR, XML, PERL
(PACKED) - default This is very similar to XDR in that data is written in a hopefully portable way (optionally non-portable if the required functions aren't available). This will probably give the smallest output files
XDR This is the same as doing it the old way
XML This stores and reads the data in an XML file. The reading is very limited and can basically only read the XML files that it generates - IT IS NOT A FULL BLOWN XML PARSER. It uses some odd constructs and isn't ideal - but you'll get the idea when you see the output. Size of created files is much larger then PACKED or XDR
PERL This generates a data structure which can be used inside a perl program - its pretty complicated stuff though using hashes for the data representation. What you do with it after you've generated it is up to you, because this is an output only library (ie it can't read back what its written).

A4GL_PDFTYPE REPORT/RUNTIME
Determine which library to use for extended reports $AUBITDIR/lib/libEXREPORT_?.so
Determine default driver for Extended Reporting When used: run-time only
Options: PDF, (NOPDF)

A4GL_RESERVEWORDS COMPILE –obsolete?– Reserved word handling Used to determine if traditionaly reserved words in 4GL language should be treated as reserved Procesing of reserved word is experimental. Set this to YES, if you want to disable this functionality. When set to NO, compiler will try to process most reserved words, instead of reporting the error.

A4GL_RM_CMD COMPILE
Command to use for deleting files on the file system

A4GL_SAPDB_ESQLC ESQL/COMPILE
Full path to SAP DB ESQL/C compiler full path to SAP-DB ESQL/C pre-compiler executable used when compiling EC output for SAP DB(does not have to be in the path)

A4GL_SQLTYPE SQL/RUNTIME/COMPILE Determine which library to use to connect to the database $AUBITDIR/libSQL_?.so Name of default SQL library plug-in to use. When used: run-time and compile-time
Options: (nosql) , <ODBC MANAGERS> iodbc unixodbc odbc32 (Windows only), <DIRECT ODBC> ifxodbc, pgodbc, sapodbc, sqliteodbc,
<NATIVE> esql esqpPG esqlSAP esqlQ sqlite sqliteS pg
<SPECIAL> FILESCHEMA
FILESCHEMA is to be used for compiling programs where either the database doesn't exist yet - or you can't get immediate access to it. This takes the 'database' as a filename (with a .schema extension) and uses that to collect the data used by compiler(s) Warning: this setting is ignored at run-rime when A4GL_LEXTYPE is set to 'EC'. At compile time, it is used by compilers regardless of A4GL_LEXTYPE setting

A4GL_UI UI/RUNTIME
Determine which plug-in to use for the user interface $AUBITDIR/lib/libUI_?.so Defines default UI (user intertface) driver plug-in to load When used: run-time only Options: (CONSOLE) [no deps.], HL_TUI [curses], GTK [GTK+], HL_GTK.

A4GL_USE_ALIAS_AS=YES|NO

A4GL_XML_EXT COMPILE/RUNTIME
extension to use with XML packer Used when when creating output (forms,menus) or opening resource files using XML packer Default: SEE ALSO: A4GL_ALWAYSCLOBBER=YES|NO

A4GL_ANSI_ERROR SQL/COMPILE
ANSI SQL 92 error checking mode When ANSI_ERROR is set to Yes, compiler will abort if non ANSI SQL 92 statement is found in source code (Static SQL only). If neither A4GL_ANSI_WARN or A4GL_ANSI_ERROR is set, no checking is performed.

A4GL_ANSI_WARN SQL/COMPILE
: ANSI SQL 92 warning checking mode When ANSI_WARN is set to Yes compiler will display a warning if it encounters static SQL statement not confitming to ANSI SQL 92 specification If neither A4GL_ANSI_WARN or A4GL_ANSI_ERROR is set, no checking is performed.

A4GL_ARR_DIR_MSG UI/TUI Display/Input array message 'There are no more rows in that direction'

AUBITDIR COMPILE/RUNTIME Specify the location of the aubit source tree or installation Default for source distribution:/opt/aubit/aubit4glsrc Default for binary distribution:/opt/aubit4gl Usually set using –prefix=/path to 'configure' script

AUBITETC COMPILE/RUNTIME
: Location of global Aubit configuration directory This internal variable points to default location of Aubit config files Default: /etc/opt/aubit4gl You should not need to change this.

AUBIT_Y2K RUNTIME
: Specify Y2K handling of dates:
+n (n<100) - set to nearest year using +n years from today as limit for future
-n (n>-100) - set to nearest year using -n from today as limit for past (note: -25 = +75) eg if year=1997 n=20 > 17 will be taken as historic anything <17 is future n=-20 <77 will be taken as future >77 is in the past
XX00 - always use century XX
999 - Do not add anything - dealing with AD 0-99
-999 - use current century

A4GL_AUTONULL COMPILE
: Auto initializing module and function local variables
=YES|NO
This setting is used at compile-time only. Numeric variables are initializet to 0, everything else to NULL To turn on, set to 'Y' (??? or is that YES ???)

A4GL_BACKGROUND UI/TUI
: Default background character (in hex) when creating a window (eg 2E for a '.') Application windows background colour xxxx is a HEX code of a colour attribute - eg 1400 (for 0x1400) for blue and reverse. Applies to all windows created when no attribute is specified (including the main screen)

A4GL_CLASSIC_I4GL_MONO UI/TUI
: Inhibit mapping of colours to attributes (like red->BOLD)
=YES|NO

A4GL_COLOR_TUI_BKG UI/TUI
: specify the default background color

A4GL_COLOR_TUI_BKG_DEF UI/TUI
: specify the default background color

A4GL_COLOR_TUI_BLACK UI/TUI
: Remap black screen colour to alternative

A4GL_COLOR_TUI_BLUE UI/TUI
: Remap blue screen color to alternative

A4GL_COLOR_TUI_CYAN UI/TUI
: Remap cyan screen color to alternative

A4GL_COLOR_TUI_FG UI/TUI
: specify the default fg color

A4GL_COLOR_TUI_FG_DEF UI/TUI
: specify the default fg color

A4GL_COLOR_TUI_GREEN UI/TUI
: Remap green screen color to alternative

A4GL_COLOR_TUI_MAGENTA UI/TUI
Remap magenta screen color to alternative

A4GL_COLOR_TUI_RED UI/TUI
Remap red screen color to alternative

A4GL_COLOR_TUI_WHITE UI/TUI
Remap white screen color to alternative

A4GL_COLOR_TUI_YELLOW UI/TUI
Remap yellow screen color to alternative

A4GL_COLUMNS UI/TUI
Specify the width of the screen See A4GL_LINES for description

A4GL_COMMENTS COMPILE
Add comments to the generated code

A4GL_COMMENT_LIKE_DISPLAY UI
Specify comments to be in current display color
=YES|NO

A4GL_COMMENT_LIKE_INPUT UI
Specify comments to be in current input color
=YES|NO

A4GL_CONSTANT2DEFINES COMPILE
Print on standard output a #define for all constants
=YES|NO (can be used to generate a .h file)

DBDATE RUNTIME
Specifies how dates will be formated

DBEDIT RUNTIME
Name of the editor to use for TEXT BLOB fields Applies to asql only?

A4GL_DBPATH RUNTIME/COMPILE
Path to look in for databases and resource files See 'DBPATH' for more information

DBPATH SQL/HELP/FORMS/MENU/RUNTIME/COMPILE
Path to look in for databases and resource files DBPATH variable containls list of directory(es) that will be searched for objects like compiled form, help and menu files, and SQLite databases. Use coloumn (:) as a delimiter between paths you want searched, (;) on Windows. Default: tools/ in Aubit source code root directory and tools/ in Aubit binaryinstallation directory. As opposed to most Aubit settings that are exclusive and order of there source (environment, aubitrc, built-in resources) decides which one will prevail, DBPATH and A4GL_DBPATH are cumulated from both variables, and added one to another in order depending on their source. So if you have path 1 in environment variable A4GL_DBPATH path 2 in environment variable DBPATH, path 3 in A4GL_DBPATH in aubitrc, path 4 in DBPATH in aubitrc, cumulated value will look like this: 1:2:3:4. Search for the file in DBPATH will then be performed from left to right, and first path found to contain file looked for will be used. NOTE: DBPATH to xxx/incl is for adbaccess form files Only SQLite databases are searched for using DBPATH. Resources file are:compiled forms/menus/help/p-code files

DBPRINT PRINT/RUNTIME/REPORT
Printing command Name of command to use to pass report output when executing reports defines as START REPORT ... TO PRINTER

A4GL_DEBUG DEBUG/COMPILE/RUNTIME
Log extensive information for tracing bugs in Aubit4gl code When you encounter programs that crash, use this for debugging - it will create file `debug.out` that can be very useful when you don't get a core dump, so you don't have file `core` to run `gdb` on. WARNING: do not set this under normal circumstances - all programs will create debug.out file when they run, files can be VERY large, and they will slow down program execution considerably. This setting applies to all Aubit compiler

executables (including all compilers) and to all 4gl programs compiled with the Aubit compiler.
FIXME: we should have separate settings for compilers and compiled programs, like A4GL_DEBUG_COMP and A4GL_DEBUG_PRG
FIXME: add note about priority numbers
Default=<not set>

A4GL_DEBUG_CFG DEBUG

A4GL_DEBUG_DECIMAL DEBUG

A4GL_DEBUG_LEVEL DEBUG/COMPILE/RUNTIME
Specify the detail in which debug messages will be logged

A4GL_DEFPRINTER PRINT

A4GL_DUMPCOMMENTS FORMS/COMPILE
Dump form file attributes when compiling form to stdout

A4GL_DUMPSTRINGS COMPILE
Dump all the strings in a 4GL to a file called strings.out
=YES|NO (normally set to 'ident') (see TRANSLATEFILE)

A4GL_DUMP_CORE DEBUG/RUNTIME
Action to perform when aubit/4gl programs crash
=YES|NO either print a sorry message (Internal Error...) , or dump core (seg fault)

A4GL_ERROR_MSG

A4GL_ESQL_UNLOAD ESQL/RUNTIME
=YES|NO

A4GL_EXTENDED_ERRORLOG DEBUG/RUNTIME
Error log handling Add module and line when writing to the error log from CALL errorlog(..)

A4GL_EXTENDED_GRAPHICS FORMS/UI/TUI
enable the use of extended graphics from form files (+<>^v for cross and tee's) If set to Y allows forms to contain the additional graphics characters <,>,^,v, and + to be used for tee's and an intersection.So the following :
\gp--v--q\g
\g| | |\g
\g>--+--<\g
\g| | |\g
\gb--^--d\g Will draw a box with an intersecting horizonal and vertical line. Note - you'll need to set this before you compile the form as well as when you run program that will use form file compiled this way

A4GL_FAKELEXTYPE PCODE/COMPILE
Compile C code resulting from 4gl compilation to P-code

A4GL_FAKE_IMMEDIATE

A4GL_FIELD_CONSTR_EXPR UI/TUI
Message to display when a fields value cannot be used for a construct statement

A4GL_FIELD_ERROR_MSG UI/TUI
Message to display when a fields value is invalid (eg non numeric in numeric field)

A4GL_FIELD_INCL_MSG UI/TUI
Message to display when a value in a field is not in the include list

A4GL_FIELD_PICTURE_MSG UI/TUI
Message to display when a pressed which is invalid for picture fields

A4GL_FIELD_REQD_MSG UI/TUI
Message to display when a field requires a value to be entered

A4GL_FIXUPDATE=YES|NO

A4GL_FORMAT_OVERFLOW RUNTIME
Determines what happens when a decimal number is too large to fit [ROUND,REFORMAT] =ROUND|REFORMAT

A4GL_GTKGUI UI/RUNTIME GTK+ —obsolete?—

GTKRC UI/RUNTIME GTK+
resources file to use when running in GTK+ GUI mode –probaly obsolete, GTK libs use this themselves?–

A4GL_GTK_INC_PATH UI/COMPILE Path to includes needed ghen compiling GTK gui enabled code —should be obsolete— Full path to GTK+ includes (header) files, used when ...? FIXME: why do we need this?

A4GL_GUIPORT UI/RUNTIME —obsolete?—

A4GL_HIDE_MENU MENU/UI/TUI
Remove menu when finished with it, default is to leave it displayed =YES|NO

A4GL_DIM_INACTIVE_MENU MENU/UI/TUI
Leave menu displayed - but as DIM rather than NORMAL to show its inactive =YES|NO

HOME COMPILE/RUNTIME
System environement vatialbe pointing to current user's home directory Used to find user-scpecific copy of Aubit configuration file (aubitrc) if any

A4GL_INCLINES DEBUG/COMPILE
Adds originating line number to genrated source code
=YES|NO Adds originating line number of each created target language statement coresponding to 4gl source code, to created target language source code, which is useful for debugging. e.g.: #line 2 '../tools/test/test_build.4gl'

INFORMIXDIR ESQL/COMPILE
Location of Informix ESQL/C installation Used when compiling EC ouptput using Informix ESQL/C compiler

A4GL_INIT_COL_REFRESH UI/TUI
Reinitialise curses colors on exit Used when curses colours must be reinitialized when returning to Screen mode (terminal specific)
=YES|NO

A4GL_INPARRAY_FULL_MSG UI/TUI Message to display when input array becomes full

A4GL_KEEP_QUALIFIER=YES|NO

A4GL_KEYFILE DEBUG/UI/RUNTIME
Read keystokes from a file and replay them Mechanism for doing automated testing A4GL_KEYFILE=(some filename in DBPATH) SEE ALSO: A4GL_KEYDELAY

A4GL_KEYDELAY DEBUG/UI/RUNTIME
Speed to replay keystokes Mechanism for doing automated testing

A4GL_KEYDELAY=(time in usec 1000000 = 1 second, defaults to 0.1s) SEE ALSO: A4GL_KEYFILE

A4GL_NEEDALLKEYS DEBUG/UI/RUNTIME
Keyfile handling. Specifies an error if more key stokes are requested than appear in the keystoke file (otherwise -return to keyboard input) SEE ALSO: A4GL_KEYFILE

A4GL_KEYLOG DEBUG/UI/RUNTIME
Log all keystokes to the specified file

A4GL_LANGUAGE

A4GL_LINES UI/TUI
: Number of rows on the screen. Terminal size This should make programs work with a normal (not xterm) terminal session. Defaults:
A4GL_COLUMNS=80
A4GL_LINES=24 FIXME: is this really A4GL_ variable - terminal will set LINES/COLUMNS, not A4GL_LINES/A4GL_COLUMNS SEE ALSO: A4GL_COLUMNS

A4GL_LOGNAME DEBUG/RUNTIME

MAKE

A4GL_MAP4GL=YES|NO

A4GL_MARK_SCOPE

A4GL_MONEY_AS_DECIMAL=YES|NO

A4GL_MONEY_AS_MONEY=YES|NO

A4GL_MONO UI/TUI
: Force monochrome output
=YES|NO

A4GL_NOCFILE=YES|NO

A4GL_NOCLOBBER=YES|NO

A4GL_NO_INVIS_ATTR UI/TUI
: Disable usage of A_INVIS in curses - attempt alternative method for concealment
=YES|NO

A4GL_PAGER

A4GL_PAUSE_MSG REPORT/RUNTIME
: Message to show when executing PAUSE statement in REPORT

A4GL_PGKEYSMOVE UI
: Defines the use of the PgUp and PgDn keys as the same as NEXT KEY or for ON KEY (PGDN)
=YES|NO

POSTGRESDIR ESQL/COMPILE
: Base directory of PostgreSQL installation. Used when looking for includes or libraries to link with, when compiling usign PostgreSQL ESQL compiler

A4GL_PRINTPROGRESS

A4GL_PRINTSCRFILE DEBUG/UI/TUI
: Specify a file to dump screen to (start with a | to pipe to a command)

A4GL_PRINTSCRKEY DEBUG/UI/TUI
: Specify a key to automatically dump the screen with (goes to PRINTSCRFILE)

A4GL_RPCTYPE RUNTIME
: Determine which library to use for remote procedure calls $AUBITDIR/lib/libRPC_?.so
Determine default RPC (Remote Procedure Call) driver to load When used: run-time only
Options: SUNRPC, (NORPC), XMLRPC
Note: XMLRPC is client only at the moment

A4GL_SCROLLBACKTO1 UI/TUI
: Display array handling
=YES|NO

A4GL_SCROLLTOEND UI/TUI
: Display array handling
=YES|NO In display array scroll back to first line if PgUp is used rather than to just first page

A4GL_SIMPLE_GRAPHICS UI/TUI
: Force usage of simple graphics for borders
=YES|NO if set to YES then +,|,- will be used to draw graphics characters instead of proper borders (if available)

A4GL_SQLCNVPATH RUNTIME/SQL
: Specifies the location of the conversion details for SQL grammars CONFIG FILE BASED CONVERSIONS convert_sql() now uses configuration files. These are by default located in /opt/aubit4gl/etc/convertsql/, but that can be changed with A4GL_SQLCNVPATH.

A4GL_SQLCONVERT COMPILE/RUNTIME/SQL
: Autoconvert SQL from sources files dialect to runtime dialect. Conversion of SQL statements in 4GL code, to the SQL dialect of target RDBMS. Conversion is only done if you set A4GL_SQLCONVERT=YES and only if the dialect used by the program differs from that used by the DBMS interface.

A4GL_SQLDIALECT COMPILE/RUNTIME/SQL
: SQL Dialect used for the source file. Declares the SQL dialect of SQL code in 4GL source code. an 4GL directive to change the default SQL dialect at runtime is: SET SQL DIALECT TO ORACLE by default the system assumes the 4GL application is using Informix SQL syntax, but this can be changed by setting, for example:A4GL_SQLEXEC SQL

A4GL_SQLPWD SQL/COMPILE/RUNTIME
: Database access password See A4GL_SQLUID for description

A4GL_SQLUID SQL/COMPILE/RUNTIME
: Database access user name FIXME: is not odbc.ini supposed to have default login name and password? Defines username and password for accessing database server via ODBC: needed for DATABASE and DEFINE LIKE statements at compile time, and procedural DATABASE statement ar run-time. You can use OPEN SESSION and supply login information at run-time, but NOT at compile time:
Default=<no default value> WARNING!! BE CAREFULL NOT TO HAVE A TAB OR OTHER SPECIAL CHARACTRS IN THE VALUE OF THIS VARIABLES !!!!!!!!!!

A4GL_SQL_CURRENT_FUNCTION SQL

A4GL_SYSTEM

A4GL_SYSTEMDIR

A4GL_SYSUSER

A4GL_TEMPDIR

A4GL_TRANSLATEFILE COMPILE
: Specifies the location of a translation file. This is used for transforming 4GL strings via a message file (see DUMPSTRINGS)

A4GL_TRANSMODE

A4GL_TRIMDUMP DEBUG/UI/TUI
: Trim the results of a dump screen to a specified screen size (eg 24x80) =24x80|25x80|24x132|25x132

A4GL_TRIMUSINGFMT RUNTIME
: Trim trailing spaces from a using string variable before applying it

A4GL_USEPAGEKEYS UI
: Does odd processing with PgUp PgDn keys on keyboard

A4GL_USE_BINDING_FOR_PUT SQL
: =YES|NO

A4GL_USE_DATABASE_STMT SQL
: =YES|NO

A4GL_USE_FORM_SIZE FORMS/UI/RUNTIME
: Aubit used to honouring the size y by x in the form, this has been removed. If you require to specify the size, it can still be used by setting A4GL_USE_FORM_SIZE=Y (using this is an Aubit extension - and not default informix behaviour!)
=YES|NO

A4GL_USE_INDICATOR ESQL/COMPILE
: Use indicator variables in ESQL/C generated code
=YES|NO

VISUAL RUNTIME
: Name of the editor for BLOB fields (?)

A4GL_YYDEBUG DEBUG/COMPILE
: Aubit parser debugging

A4GL_EXDTYPE RUNTIME
: External data types support to be loaded $AUBITDIR/lib/libEXDTYPE_?.so Currently only MPZ (large integers) are supported FIME: not sure if this is needed - looks like this is loaded on request: see example testmpz.4gl into the tools/test directory.

A4GL_NULL_DECIMAL_IF_BAD RUNTIME
: Null a decimal value rather than set it to 0 if its invalid
=YES|NO The standard informix behaviour seems to be to set the value to 0 for decimals but sets dates to NULL. This is inconsistent and so this default behaviour is switchable via this configuration setting

A4GL_BEEPONERROR RUNTIME
: Indicates that a beep should be emitted by the ERROR statement
=YES|NO

A4GL_FLASHONERROR RUNTIME
: Indicates that a screen flash should be emitted by the ERROR statement
=YES|NO Not all terminals are capable of emitting a screen flash. If a screen flash is not possible then the terminal bell is rung instead.

A4GL_REFRESH_AFTER_MOVE UI/TUI
: Issue a screen update after a cursor movement
=YES|NO This is a screen drawing optimisation function. Normally a screen update is not required but there may be some instances where the screen cursor does not move to the right place if this isn't set. If you're not too worried about where the screen cursor is, or your application doesn't suffer from this problem then set this to N

A4GL_FIRSTCOL_ONERR UI/TUI
: Move to the beginning of a field after an error
=YES|NO Can only be set if CLR_FIELD_ON_ERROR=N See CLR_FIELD_ON_ERROR

A4GL_CLR_FIELD_ON_ERROR UI/TUI
: Clears a field after an error
=YES|NO If this is set them FIRSTCOL_ONERR will never be triggered See FIRSTCOL_ONERR

A4GL_NO_REFRESH_AFTER_SYSTEM UI
: Issue a screen refresh after any sysem command
=YES|NO In Informix 4GL, the screen is not refreshed after every system command but only after a new screen instruction is issued. This means that if you are running a lot of system commands, Aubit4GL's screen may appear to flicker between line mode and screen mode. Set this to N to inhibit the automatic screen refresh.

A4GL_NO_ARRAY_EXPAND COMPILE
: Remove the array expansion code
=YES|NO This is solely for backward compatibilty with older Aubit4GL versions. It should be set to N in all other cases..

RM_COMMENTS_FIRST COMPILE
: remove any comments before compiling the 4GL code
=YES|NO This defaults to Yes, if you have problems with compilation - it may be that this code is getting confused. Try setting to N, or setting DUMP_LAST

GDB_ATTACH RUNTIME Attach GDB
to the process when a Segmentation Fault occurs
=YES|NO This is useful for tracing back problems during runtime execution The first command to execute in gdb would normally be a 'bt' which should give something like :
#0 0x402095a9 in ___wait4 () from /lib/libc.so.6
#1 0x40271ad8 in ___DTOR_END___ () from /lib/libc.so.6
#2 0x401ad506 in system () from /lib/libc.so.6
#3 0x40038858 in A4GL_core_dump () at fglwrap.c:911
#4 <signal handler called>
#5 0x8048bbd in aclfgl_xxx (_nargs=0) at ./x1.c:95
#6 0x8048a6d in main (argc=1, argv=0xbffff1d4) at ./x1.c:58
#7 0x40180baf in ___libc_start_main () from /lib/libc.so.6
Ignore everything up to the <signal handler called>, and 'frame 5' (in this case) should show the offending line.

DUMP_LAST COMPILE
output the results of the last remove comments
=YES|NO This will produce a file 'last' which contains the file with the comments removed. This is used to check the operation of the RM_COMMENTS_FIRST code