# The Aubit4GL Manual

Editor John O'Gorman

13 December 2010

# Contents

**1 Features** **21**

1.1 4GL: 21

1.2 Aubit 4GL 21

1.3 Aubit4GL Benefits 22

1.3.1 GNU, GPL, OpenSource 22

1.3.2 Commercial Support 22

1.3.3 Productive 23

1.3.4 Fast 23

1.3.5 Compatible 23

1.3.6 Engine Independent 23

1.4 Aubit4GL Extensions 24

1.5 What's New 24

**2 Install** **27**

2.1 Platforms 27

2.2 Choices 27

2.2.1 GTK 28

2.2.2 PDFLib 29

2.2.3 Which . . . 29
2.2.4 Whither . . . 29
2.3 But first . . . 30
2.3.1 Binary . . . 30
2.3.2 Source . . . 31
2.4 Download . . . 33
2.4.1 Filenames . . . 33
2.4.2 Tarballs . . . 34
2.4.2.1 CVS . . . 34
2.5 Build . . . 35
2.5.1 configure . . . 35
2.5.2 Binary . . . 36
2.5.3 Source . . . 36
2.5.3.1 Missing Software . . . 37
2.6 Connect . . . 38
2.6.1 AUBITDIR . . . 38
2.6.2 PATH . . . 38
2.6.3 ldconfig . . . 38
2.7 aubitrc . . . 39
2.7.1 Binary . . . 40
2.7.2 Source . . . 40
2.7.3 Plugins . . . 40
2.7.3.1 Informix . . . 41
2.7.3.2 PostgreSQL . . . 41
2.7.3.3 MySQL . . . 41
2.7.3.4 SQLite3 . . . 41
2.7.3.5 Others . . . 41

2.7.4 A4GL_SQLACL . . . . . . . . . . . . . . . . . . . . . . . 41

2.8 DECIMAL format . . . . . . . . . . . . . . . . . . . . . . . 42

2.9 Remote connection to pg8: . . . . . . . . . . . . . . . . . . . 42

2.10 Check . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44

2.10.1 Commands . . . . . . . . . . . . . . . . . . . . . . 44

2.10.2 Try to compile a simple 4gl hello . . . . . . . . . . . . 46

2.10.3 With database . . . . . . . . . . . . . . . . . . . . 47

2.10.4 Test Programs . . . . . . . . . . . . . . . . . . . . 47

2.10.5 For Informix . . . . . . . . . . . . . . . . . . . . . 48

2.10.5.0.1 Cient SDK . . . . . . . . . . . . . . 48

2.10.5.0.2 Check the SDK . . . . . . . . . . . 48

2.10.5.0.3 Set Up Aubit . . . . . . . . . . . . 48

2.10.5.0.4 Try to compile a simple 4gl . . . . 48

2.10.5.0.5 Try to run it . . . . . . . . . . . . . 49

**3 Set up** **51**

3.1 ODBC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51

3.1.1 ODBC config files . . . . . . . . . . . . . . . . . . . 52

3.1.1.1 Sample odbcinst.ini . . . . . . . . . . . . . . . . 52

3.1.1.2 ODBC Datasources . . . . . . . . . . . . . . . . 53

3.1.1.3 Informix ODBC Drivers . . . . . . . . . . . . . 54

3.1.1.3.1 Informix Driver Manager . . . . . . 55

3.1.1.4 PostgreSQL Drivers . . . . . . . . . . . . . . . 55

3.1.1.5 SAPDB Drivers . . . . . . . . . . . . . . . . . 55

3.1.1.6 ODBC Warning . . . . . . . . . . . . . . . . . 56

3.1.1.7 Native . . . . . . . . . . . . . . . . . . . . . . 56

3.2 Databases . . . . . . . . . . . . . . . . . . . . . . . . . . . 57

3.2.1 Informix . . . . . . . . . . . . . . . . . . . . . . . . . 57
3.2.2 PostgreSQL . . . . . . . . . . . . . . . . . . . . . . . 57
3.2.2.1 Overview . . . . . . . . . . . . . . . . . . . . 57
3.2.2.2 Documentation . . . . . . . . . . . . . . . . . 57
3.2.2.3 Installation . . . . . . . . . . . . . . . . . . . 58
3.2.2.4 Instances . . . . . . . . . . . . . . . . . . . . 59
3.2.2.4.1 initdb . . . . . . . . . . . . . . . . . 59
3.2.2.4.2 pg_ctl start . . . . . . . . . . . . . 60
3.2.2.4.3 pg_ctl stop . . . . . . . . . . . . . . 60
3.2.2.4.4 createdb . . . . . . . . . . . . . . . 61
3.2.2.5 Environment . . . . . . . . . . . . . . . . . . 61
3.2.2.6 Maintenance . . . . . . . . . . . . . . . . . . 63
3.2.2.6.1 vacuumdb . . . . . . . . . . . . . . 63
3.2.2.6.2 pg_dump . . . . . . . . . . . . . . . 63
3.2.2.7 Commands . . . . . . . . . . . . . . . . . . . 63
3.2.2.8 psql . . . . . . . . . . . . . . . . . . . . . . . 64
3.2.2.9 Stored Procedures . . . . . . . . . . . . . . . 65
3.2.2.9.1 PL/pgSQL matches function . . . . 66
3.2.2.9.2 $$ quoting . . . . . . . . . . . . . . 67
3.2.2.9.3 E escaping . . . . . . . . . . . . . . 67
3.2.2.9.4 PL/pgSQL install . . . . . . . . . . 67
3.2.3 MySQL . . . . . . . . . . . . . . . . . . . . . . . . . 68
3.2.4 SQLite3 . . . . . . . . . . . . . . . . . . . . . . . . . 69
3.2.5 SQLServer . . . . . . . . . . . . . . . . . . . . . . . 70

**4 Problems** **73**

4.1 Curses . . . 73

4.1.1 Wide Characters . . . 73

4.1.2 Encodings . . . 73

4.1.3 LENGTH . . . 76

4.2 Engines . . . 76

**5 Modules** **81**

5.0.1 Choices . . . 81

**6 Aubit4GL Compilers** **87**

6.1 A4GL compilers . . . 87

6.2 4glpc . . . 88

6.2.1 Usage . . . 88

6.3 4glc . . . 92

6.4 Compiling forms . . . 92

6.5 Compiling help files . . . 92

6.6 Compiling menu files . . . 93

**7 4GL Language** **95**

7.1 Introduction . . . 95

7.2 Summary: . . . 96

7.3 Short Intro to x4GL . . . 97

7.3.1 4GL Programs . . . 97

7.3.1.1 Structure of a program . . . 97

7.3.1.2 DATABASE section . . . 98

7.3.1.3 GLOBALS section . . . 98

7.3.1.4 Functions . . . 98

7.3.1.5 MAIN block . . . . . . . . . . . . . . . . . . 99

7.3.1.6 DEFINE section . . . . . . . . . . . . . . . . . 99

7.3.1.7 Arrays Syntax: . . . . . . . . . . . . . . . . . 100

7.3.1.8 Records . . . . . . . . . . . . . . . . . . . . 100

7.3.1.8.1 Syntax . . . . . . . . . . . . . . . . 101

7.3.1.9 Associative Arrays . . . . . . . . . . . . . . . 102

7.3.1.9.1 Performance Note . . . . . . . . . . 102

7.3.1.10 Constants . . . . . . . . . . . . . . . . . . . 103

7.3.1.11 DEFINE NEW TYPE . . . . . . . . . . . . . 103

7.3.1.12 Packages . . . . . . . . . . . . . . . . . . . 104

7.4 Quick Reference . . . . . . . . . . . . . . . . . . . . . . 105

7.4.1 Data Types . . . . . . . . . . . . . . . . . . . . . . 105

7.4.2 Constants . . . . . . . . . . . . . . . . . . . . . . . 107

7.4.3 Global Variables . . . . . . . . . . . . . . . . . . . 107

7.4.4 Syntax Conventions . . . . . . . . . . . . . . . . . 107

7.4.5 Operators . . . . . . . . . . . . . . . . . . . . . . . 108

7.4.6 Aubit4GL Operators . . . . . . . . . . . . . . . . . 109

7.4.7 Attribute Constants . . . . . . . . . . . . . . . . . 109

7.4.8 Key Constants . . . . . . . . . . . . . . . . . . . . 109

7.4.9 Table Privileges . . . . . . . . . . . . . . . . . . . 110

7.4.10 Comments . . . . . . . . . . . . . . . . . . . . . . 110

7.4.11 4GL Statement Syntax . . . . . . . . . . . . . . . 110

7.4.12 Report Syntax . . . . . . . . . . . . . . . . . . . . 123

7.4.13 Report Statement Syntax . . . . . . . . . . . . . . 123

7.4.14 Report Expressions . . . . . . . . . . . . . . . . . . 123

7.4.15 PDF Report Syntax . . . . . . . . . . . . . . . . . 124

7.4.15.1 PDF Report Expressions . . . . . . . . . . . 125

7.4.15.2 PDF Statements . . . . . . . . . . . . . . . . 125
7.4.15.3 PDF_FUNCTION arglists . . . . . . . . . . . 126
7.5 Builtin Functions . . . . . . . . . . . . . . . . . . . . . . . . 126
7.5.1 Standard 4GL Builtin Functions . . . . . . . . . . . . 126
7.5.2 Standard 4GL Operators . . . . . . . . . . . . . . . . 127
7.5.3 D4GL Builtin Functions . . . . . . . . . . . . . . . . . 129
7.5.4 Aubit Builtin Functions . . . . . . . . . . . . . . . . . 130
7.5.5 a4gl_get_info() . . . . . . . . . . . . . . . . . . . . . 130
7.5.5.1 Connection . . . . . . . . . . . . . . . . . . . 131
7.5.5.2 Form . . . . . . . . . . . . . . . . . . . . . . 131
7.5.5.3 Statement . . . . . . . . . . . . . . . . . . . 132
7.5.5.4 Window . . . . . . . . . . . . . . . . . . . . 132
7.5.6 aclfgl_ Builtins . . . . . . . . . . . . . . . . . . . . 133
7.5.6.1 Procedures . . . . . . . . . . . . . . . . . . . 133
7.5.6.2 Functions . . . . . . . . . . . . . . . . . . . 133
7.6 Form Syntax . . . . . . . . . . . . . . . . . . . . . . . . . 134
7.6.1 Tag Description . . . . . . . . . . . . . . . . . . . . 135
7.6.2 Aubit 4GL GUI Attributes . . . . . . . . . . . . . . . 136
7.7 VDC Forms . . . . . . . . . . . . . . . . . . . . . . . . . 136
7.8 Callbacks . . . . . . . . . . . . . . . . . . . . . . . . . . 137

**8 Help system** **139**
8.1 Help message source file . . . . . . . . . . . . . . . . . . 139
8.2 Compiling help files . . . . . . . . . . . . . . . . . . . . 139
8.3 help in programs . . . . . . . . . . . . . . . . . . . . . . 140
8.3.1 Within 4GL . . . . . . . . . . . . . . . . . . . . . . 140
8.3.2 At runtime . . . . . . . . . . . . . . . . . . . . . . . 140
8.4 Decompiling . . . . . . . . . . . . . . . . . . . . . . . . . 140
8.5 Compatibility . . . . . . . . . . . . . . . . . . . . . . . . 140
8.6 mkmess . . . . . . . . . . . . . . . . . . . . . . . . . . . 141

**9 SQL Conversion 143**

9.1 Source SQL dialect . . . 144

9.2 Target SQL dialect . . . 144

9.3 Configuration files . . . 144

9.4 Converting SQL . . . 145

9.5 Conversion Syntax . . . 146

9.5.1 Simple directives . . . 146

9.5.2 Complex Directives . . . 147

9.5.3 REPLACE directives . . . 147

**10 Make 151**

10.0.4 GNU make . . . 151

10.1 Makefiles . . . 152

10.1.0.1 Include File . . . 152

10.1.0.2 Make glossary: . . . 152

10.1.0.3 Makefile Example . . . 153

10.1.1 Pattern Rules . . . 154

10.1.2 Make variables . . . 154

10.1.3 GPATH and VPATH . . . 155

10.1.4 .PHONY . . . 155

10.1.5 Implicit rules . . . 155

10.1.6 Syntax . . . 155

10.1.7 Debugging make . . . 156

**11 amake 157**

11.1 Introduction . . . 157

11.2 Summary . . . 158

11.3 Converting old makefiles . . . . . . . . . . . . . . . . . . . . . . 158
11.3.1 prepmake . . . . . . . . . . . . . . . . . . . . . . 158
11.3.2 example . . . . . . . . . . . . . . . . . . . . . . . 158
11.3.3 amakeallo . . . . . . . . . . . . . . . . . . . . . . 159
11.3.4 amakeallf . . . . . . . . . . . . . . . . . . . . . . 159
11.4 2. amake . . . . . . . . . . . . . . . . . . . . . . . . . 159
11.4.1 Requests . . . . . . . . . . . . . . . . . . . . . . . 160
11.4.2 Notes . . . . . . . . . . . . . . . . . . . . . . . . . 160
11.4.3 Installation . . . . . . . . . . . . . . . . . . . . . 161
11.4.4 Credits: . . . . . . . . . . . . . . . . . . . . . . . 162
11.4.5 #DEFINE . . . . . . . . . . . . . . . . . . . . . . 163
11.4.6 4GL Makefiles . . . . . . . . . . . . . . . . . . . . 163
11.4.6.1 Makefiles for Classic 4GL on Unix . . . . . . 163
11.4.7 D4GL Makefiles on Unix . . . . . . . . . . . . . . . 165
11.4.7.1 I4GL Makefiles on Unix . . . . . . . . . . . . 165
11.4.7.2 NMAKE . . . . . . . . . . . . . . . . . . . . 166
11.4.8 Bug in ESQL/C rules: . . . . . . . . . . . . . . . . 167

**12 A4GL Utilities** **169**
12.1 adbschema . . . . . . . . . . . . . . . . . . . . . . . . 169
12.2 afinderr . . . . . . . . . . . . . . . . . . . . . . . . . 170
12.3 asql . . . . . . . . . . . . . . . . . . . . . . . . . . . 170
12.3.1 runforms . . . . . . . . . . . . . . . . . . . . . . . 172
12.4 aupscol . . . . . . . . . . . . . . . . . . . . . . . . . 173
12.5 P-Code Dropped . . . . . . . . . . . . . . . . . . . . . 173
12.6 configurator . . . . . . . . . . . . . . . . . . . . . . . 173
12.7 convertsql . . . . . . . . . . . . . . . . . . . . . . . . 174

12.8 default_frm . . . . . . . . . . . . . . . . . . . . . . . . . 174
12.9 fcompile . . . . . . . . . . . . . . . . . . . . . . . . . . . 175
12.9.1 Builtin Forms . . . . . . . . . . . . . . . . . . . . . 175
12.10 fshow . . . . . . . . . . . . . . . . . . . . . . . . . . . 176
12.11 loadmap . . . . . . . . . . . . . . . . . . . . . . . . . . 177
12.12 mkpackage . . . . . . . . . . . . . . . . . . . . . . . . 177
12.13 prepmake . . . . . . . . . . . . . . . . . . . . . . . . . 177
12.14 decompilers . . . . . . . . . . . . . . . . . . . . . . . 177
12.15 Internal Apps . . . . . . . . . . . . . . . . . . . . . . 178
12.15.1 xgen . . . . . . . . . . . . . . . . . . . . . . . . . . 178

**13 Packages 179**

13.1 Packages . . . . . . . . . . . . . . . . . . . . . . . . . . 179
13.2 channel . . . . . . . . . . . . . . . . . . . . . . . . . . . 181
13.2.1 Dependencies . . . . . . . . . . . . . . . . . . . . . 181
13.2.2 Synopsis . . . . . . . . . . . . . . . . . . . . . . . . 182
13.3 file . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 182
13.3.1 Dependencies . . . . . . . . . . . . . . . . . . . . . 182
13.3.2 Synopsis . . . . . . . . . . . . . . . . . . . . . . . . 183
13.4 html . . . . . . . . . . . . . . . . . . . . . . . . . . . . 183
13.4.1 Dependencies . . . . . . . . . . . . . . . . . . . . . 184
13.4.2 Synopsis . . . . . . . . . . . . . . . . . . . . . . . . 184
13.4.3 Example . . . . . . . . . . . . . . . . . . . . . . . . 187
13.5 memcached . . . . . . . . . . . . . . . . . . . . . . . . . 189
13.5.1 Dependencies . . . . . . . . . . . . . . . . . . . . . 189
13.5.2 Synopsis . . . . . . . . . . . . . . . . . . . . . . . . 189
13.6 pcre . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 190

13.6.1 Dependancies . . . . . . . . . . . . . . . . . . . . . . . 191
13.6.2 Synopsis . . . . . . . . . . . . . . . . . . . . . . . . . . 191
13.7 pop . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 191
13.7.1 Dependancies . . . . . . . . . . . . . . . . . . . . . . . 192
13.7.2 Synopsis . . . . . . . . . . . . . . . . . . . . . . . . . . 192
13.8 smtp . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 194
13.8.1 Dependancies . . . . . . . . . . . . . . . . . . . . . . . 194
13.8.2 Synopsis . . . . . . . . . . . . . . . . . . . . . . . . . . 194
13.9 string . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 195
13.9.1 Dependencies . . . . . . . . . . . . . . . . . . . . . . . 195
13.9.2 Synopsis . . . . . . . . . . . . . . . . . . . . . . . . . . 196
13.10sxml . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 196
13.10.1Dependencies . . . . . . . . . . . . . . . . . . . . . . . 196
13.10.2Synopsis . . . . . . . . . . . . . . . . . . . . . . . . . . 197
13.11dynamic . . . . . . . . . . . . . . . . . . . . . . . . . . . . 197
13.11.1Dependencies . . . . . . . . . . . . . . . . . . . . . . . 197
13.11.2Function list . . . . . . . . . . . . . . . . . . . . . . . . 198

**14 Extensions 199**
14.1 Fake Comments {! ... !} . . . . . . . . . . . . . . . . . . . 199
14.2 Associative Arrays . . . . . . . . . . . . . . . . . . . . . . 200
14.3 Paused Screen Handling . . . . . . . . . . . . . . . . . . . 200
14.4 Slices . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 201
14.5 TODO statement . . . . . . . . . . . . . . . . . . . . . . . . 201
14.6 ODBC Data access . . . . . . . . . . . . . . . . . . . . . . 202
14.7 Concurrent Connections . . . . . . . . . . . . . . . . . . . 202
14.8 Constants . . . . . . . . . . . . . . . . . . . . . . . . . . . . 202

14.9 Callback Functions . . . . . . . . . . . . . . . . . . . . . . 203
    14.9.1 CONSTRUCT VIA . . . . . . . . . . . . . . . . . . . . 203
        14.9.1.1 VIA Example . . . . . . . . . . . . . . . . . 204
    14.9.2 SORT ... USING sortfunc . . . . . . . . . . . . . . . . 206
        14.9.2.1 Example code . . . . . . . . . . . . . . . . . 207
        14.9.2.2 Example 2 . . . . . . . . . . . . . . . . . . 208
    14.9.3 LOAD ... USING FILTER fname ... . . . . . . . . . . . 209
14.10Error Hooks . . . . . . . . . . . . . . . . . . . . . . . . . 211
    14.10.1 A4GL_ERRHOOK . . . . . . . . . . . . . . . . . . . . 211
    14.10.2 errlog() . . . . . . . . . . . . . . . . . . . . . . . 212
    14.10.3 Example . . . . . . . . . . . . . . . . . . . . . . . 212
    14.10.4 sample.4gl . . . . . . . . . . . . . . . . . . . . . . 213
14.11Map Files . . . . . . . . . . . . . . . . . . . . . . . . . . 215
14.12New Types . . . . . . . . . . . . . . . . . . . . . . . . . . 215
14.13Variable IDs . . . . . . . . . . . . . . . . . . . . . . . . . 216
14.14Passing IDs . . . . . . . . . . . . . . . . . . . . . . . . . 216
14.15Embedded C code. . . . . . . . . . . . . . . . . . . . . . . 216
14.16MOVE WINDOW . . . . . . . . . . . . . . . . . . . . . . . 218
14.17WHENEVER . . . . . . . . . . . . . . . . . . . . . . . . . 218
14.18Multilevel Menus . . . . . . . . . . . . . . . . . . . . . . 218
14.19Extended DISPLAY . . . . . . . . . . . . . . . . . . . . . 218
14.20Extended USING . . . . . . . . . . . . . . . . . . . . . . 218
14.21Local functions . . . . . . . . . . . . . . . . . . . . . . . 219
14.22get_info function . . . . . . . . . . . . . . . . . . . . . . 219
14.23a4gl_get_info() . . . . . . . . . . . . . . . . . . . . . . . 219
14.24get_error_details() . . . . . . . . . . . . . . . . . . . . . 220
14.25Dynamic Form Fields . . . . . . . . . . . . . . . . . . . . 220

14.26 Remote Functions . . . 221
14.27 LINKED TO . . . 221
14.28 ON ANY KEY etc . . . 222
14.29 Compile Time Environment . . . 222
14.30 SCHEMA v DATABASE . . . 223
14.31 SESSIONS . . . 224
14.32 Application Partitioning . . . 224
14.33 Y2K Runtime Translation . . . 225
14.34 Globbing . . . 225
14.35 A4GL Wizard . . . 225
14.35.1 Program Templates . . . 225
14.36 PDF Reports . . . 225
14.37 GUI . . . 226
14.38 Packages . . . 226
14.39 a4gl IDE . . . 226
14.39.1 Independent Development Environment . . . 226
14.40 Logical Reports . . . 226

**15 ACE reports** **227**

15.0.1 generate_aace . . . 228
15.1 aace . . . 228
15.2 aace_4gl . . . 228
15.2.1 -C Compatibility . . . 229
15.2.2 -I Insert Cursor . . . 229
15.2.3 -B Batch Size . . . 229

**16 New Display Clients 231**

16.1 New GUI Front Ends . . . . . . . . . . . . . . . . . . . . . . . . 231

16.1.1 History . . . . . . . . . . . . . . . . . . . . . . . . 231

16.1.1.1 TUI . . . . . . . . . . . . . . . . . . . . . . . . 231

16.1.1.2 GTK . . . . . . . . . . . . . . . . . . . . . . . . 231

16.1.1.3 HL_TUI, HL_GTK . . . . . . . . . . . . . . . . 232

16.1.1.4 Graphical Front Ends . . . . . . . . . . . . . . 232

16.2 VDC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 233

16.3 Requirements . . . . . . . . . . . . . . . . . . . . . . . . . . . 233

16.4 Ventas Display Client . . . . . . . . . . . . . . . . . . . . . . 234

16.4.1 Linux . . . . . . . . . . . . . . . . . . . . . . . . . 234

16.4.1.1 Source . . . . . . . . . . . . . . . . . . . . . . 234

16.4.1.2 Binary . . . . . . . . . . . . . . . . . . . . . . 234

16.4.2 Windows . . . . . . . . . . . . . . . . . . . . . . . 234

16.4.2.1 Source . . . . . . . . . . . . . . . . . . . . . . 234

16.4.2.2 Binary . . . . . . . . . . . . . . . . . . . . . . 235

16.5 Proxy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 235

16.5.0.3 Authentication . . . . . . . . . . . . . . . . . . 236

16.6 Form Layouts . . . . . . . . . . . . . . . . . . . . . . . . . . . 236

16.6.1 SCREEN . . . . . . . . . . . . . . . . . . . . . . . . 236

16.6.2 LAYOUT . . . . . . . . . . . . . . . . . . . . . . . . 236

16.6.2.1 GRID . . . . . . . . . . . . . . . . . . . . . . . 237

16.6.2.2 TABLE . . . . . . . . . . . . . . . . . . . . . . 237

16.6.2.3 HBOX, VBOX . . . . . . . . . . . . . . . . . . 238

16.6.2.4 FOLDER . . . . . . . . . . . . . . . . . . . . . 238

16.6.2.5 Container Syntax . . . . . . . . . . . . . . . . 240

16.6.3 Field Widgets . . . . . . . . . . . . . . . . . . . . . 240

16.6.3.1 Widget Syntax . . . . . 241
16.7 Settings/Environment Variables . . . . . 242
16.7.1 Debugging . . . . . 242
16.8 Special functions . . . . . 243
16.9 Ventas GUI Client . . . . . 243
16.9.1 Startup . . . . . 244
16.10 Look & Feel . . . . . 244
16.10.1 Toolbars . . . . . 244
16.10.2 Images . . . . . 245
16.10.3 Application Launcher . . . . . 245
16.10.3.1 STYLES . . . . . 245
16.10.3.2 Attributes . . . . . 246
16.10.3.3 Modifiers . . . . . 247
16.10.3.4 Actions . . . . . 247
16.10.3.5 Attributes . . . . . 248
16.10.4 SSH client mode . . . . . 248
16.11 Other GUI clients . . . . . 248
16.12 Protocol . . . . . 249
16.12.1 Testing . . . . . 249
16.12.2 DTDs . . . . . 251

**17 PDF Reports 253**

17.1 Before you start . . . . . 253
17.2 Introduction . . . . . 253
17.3 Output Section . . . . . 254
17.3.1 Fonts . . . . . 255
17.3.2 Report Structure . . . . . 255

17.3.3 Extras . . . . . . . . . . . . . . . . . . . . . . . . . 256
17.3.3.1 Positioning . . . . . . . . . . . . . . . . . . . . 256
17.3.3.2 Images . . . . . . . . . . . . . . . . . . . . . . 256
17.3.4 Example program . . . . . . . . . . . . . . . . . . . 257
17.4 Barcodes . . . . . . . . . . . . . . . . . . . . . . . . . . 257
17.4.1 Barcodes 128 . . . . . . . . . . . . . . . . . . . . . 260
17.4.1.1 Tilde Method . . . . . . . . . . . . . . . . . . . 261
17.4.1.2 Type 128B . . . . . . . . . . . . . . . . . . . . 262
17.5 Printing . . . . . . . . . . . . . . . . . . . . . . . . . . . 263
17.5.1 PDF . . . . . . . . . . . . . . . . . . . . . . . . . . 264
17.5.2 ASCII, ISO8895, etc . . . . . . . . . . . . . . . . . . 264
17.5.3 CUPS . . . . . . . . . . . . . . . . . . . . . . . . . 264
17.5.3.1 Print Queue Set Up . . . . . . . . . . . . . . . 265
17.5.3.2 cpi and lpi . . . . . . . . . . . . . . . . . . . . 266
17.5.3.3 CUPS Documentation . . . . . . . . . . . . . . 267
17.5.4 Print Queue Problems . . . . . . . . . . . . . . . . . 267
17.5.4.1 a2ps . . . . . . . . . . . . . . . . . . . . . . . . 268

**18 Logical Reports 271**
18.1 Invoking a logical report . . . . . . . . . . . . . . . . . . 271
18.1.1 FINISHing the report . . . . . . . . . . . . . . . . . 271
18.1.2 Converting to "filename" . . . . . . . . . . . . . . . . 272
18.1.3 Default layouts . . . . . . . . . . . . . . . . . . . . . 272
18.1.4 Converting to many . . . . . . . . . . . . . . . . . . 273
18.2 Saved Meta Data . . . . . . . . . . . . . . . . . . . . . . 274
18.2.1 The Report Viewer . . . . . . . . . . . . . . . . . . . 274
18.2.2 The layout editor . . . . . . . . . . . . . . . . . . . 275
18.2.3 The report processor . . . . . . . . . . . . . . . . . . 276
18.2.4 Tips for CSV layouts . . . . . . . . . . . . . . . . . . 276
18.3 Helper programs . . . . . . . . . . . . . . . . . . . . . . . 277

**19 Debugging 279**

19.1 Coredumps . . . . . . . . . . . . . . . . . . . . . . . . . 279

19.2 Unexpected behaviour . . . . . . . . . . . . . . . . . . . 280

19.3 All other errors . . . . . . . . . . . . . . . . . . . . . . . 280

19.4 compiler errors . . . . . . . . . . . . . . . . . . . . . . . 280

19.5 Reporting bugs . . . . . . . . . . . . . . . . . . . . . . . 280

**20 Web Services 281**

20.1 4GL Web Tools . . . . . . . . . . . . . . . . . . . . . . . 281

20.1.1 Client . . . . . . . . . . . . . . . . . . . . . . . 282

20.1.2 Server . . . . . . . . . . . . . . . . . . . . . . . 282

20.2 WSDL and SOAP . . . . . . . . . . . . . . . . . . . . . . 282

20.3 gSOAP . . . . . . . . . . . . . . . . . . . . . . . . . . . 283

20.3.1 Warning . . . . . . . . . . . . . . . . . . . . . . 284

20.4 wsdl2fgl . . . . . . . . . . . . . . . . . . . . . . . . . . . 284

20.4.1 Client Example . . . . . . . . . . . . . . . . . . . 285

20.4.2 Web Server . . . . . . . . . . . . . . . . . . . . . 286

20.4.3 Limitations . . . . . . . . . . . . . . . . . . . . . 291

20.4.3.1 Single Threaded . . . . . . . . . . . . . . . 291

20.4.3.2 Limited Datatypes . . . . . . . . . . . . . . 291

20.4.3.3 Unsupported Services . . . . . . . . . . . . 292

**21 Revisions 293**

21.1 2010-8-23 . . . . . . . . . . . . . . . . . . . . . . . . . . 293

21.2 2006-8-1 . . . . . . . . . . . . . . . . . . . . . . . . . . . 293

21.3 2005-9-9 . . . . . . . . . . . . . . . . . . . . . . . . . . . 294

21.4 2005-3-12 . . . . . . . . . . . . . . . . . . . . . . . . . . 294

21.5 2004-4-27 . . . . . . . . . . . . . . . . . . . . . . . . . . 294

21.6 2004-2-22 . . . . . . . . . . . . . . . . . . . . . . . . . . 295

21.7 Problems . . . . . . . . . . . . . . . . . . . . . . . . . . 295

**22 Environment Variables** **297**

22.0.1 Version 1.2 . . . . . . 315

**23 This Manual** **329**

23.1 LyX . . . . . . 329

23.2 TeX . . . . . . 330

23.3 LaTeX . . . . . . 330

23.4 PDF and HTML . . . . . . 331

23.4.1 PDF . . . . . . 331

23.4.2 HTML . . . . . . 332

23.4.2.1 Export LyX to TeX . . . . . . 333

23.4.2.2 Convert TeX to HTML . . . . . . 333

23.4.3 Makefile . . . . . . 334

# 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 most to it. A number of other contributors have included Andrej Falout, John O'Gorman, Sergio Ferreira, Walter Haslbeck. Recently, other companies have joined the Aubit4GL brotherhood with the intention of contributing to the development of independent graphical frontends.

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, mysql, SQLite3, 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 Commercial Support

Commercial support is available from Aubit Computing Ltd if you want it. Aubit Computing Ltd is Mike Aubury's company. Aubit has developed some important new debugging tools: fgllint and fglcalltree which assist professional developers to find bugs in any 4GL source code. These tools are not GPLed but can be purchased from Aubit Computing Ltd.

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 also available from:

- Moredata `http://www.moredata.eu` for Portuguese and Spanish speakers
- Ventas `http://www.ventas.de` for German speakers. Ventas are developing the new VDC graphic frontend for Aubit4GL

### 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 not 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 TUI, 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.

## 1.5 What's New

Since version 1.10, we have added a lot of things to Aubit4GL. These include:

- A new TODO ... END TODO construction similar to CASE ... END CASE but which iterates until all its WHEN branches are satisfied
- A new SORT arrayvar USING sortfunc statement which will sort an array for you using your supplied callback function sortfunc to order successive elements.
- Extensions to the CONSTRUCT, SORT, and LOAD statements to allow use of a callback function
- DEFINE NEW TYPE id definition
- New event catchers for INPUT/DISPLAY/CONSTRUCT statements e.g.
    - ON CHANGE field

  - ON ANY KEY
  - ON IDLE | INTERVAL n HOURS | MINUTES | SECONDS

- The ability to use error hooks (i.e. calls to a user supplied error function)
- Extensions to the PDFREPORT to support
  - More 4GL statements for setting font properties, etc
  - Support for more barcode formats
- Myriad more environment vars to control lots of things
- INPUT/DISPLAY ARRAY arrname SLICE [field1 THRU fieldn]
- A COPYOF operator and its complement COPYBACK which allow efficient bulk passing of function parameters using memcpy() rather than tedious pushing and popping to and from the 4GL function call stack. This is mainly for the benefit of the new SORT statement.
- Some tools (fglproto, wsdl2fgl) to assist in using Aubit4GL to supply and/or use web services
- A new XML interface to allow independent Visual Display applications to communicate with Aubit4GL backends.
- IGNORE ERROR( n [, ..]) for 4glstatement
- USE IGNORE ERROR FOR 4glstatement
- REMAP ERROR -n=-m
- Many new Aubit4GL builtin functions.

# Chapter 2

# Install

## 2.1 Platforms

You can install Aubit4GL on recent versions of

- Linux (e.g Redhat, openSUSE, Ubuntu, and the like)
- Unix operating systems
- Apple Mac OSX
- Microsoft Windows with or without Cygwin.

## 2.2 Choices

Whereas Informix4GL works only with Informix databases, Aubit4GL can work with

- Native database connections also for PostgreSQL, SAPDB, Mysql, or SQLite3
- Vendor ODBC packages for Informix, PostgreSQL, MySQL, and SQL-ite3

- Independent ODBC manager libraries for database connection: unix-ODBC, iODBC (or Windows ODBC)
- PDF library for fancy reports using PDFLite from `www.pdf.org`
- GTK2+ Library (for grapical frontend)
- Jabber IM library (for instant messaging) using the iksemel package
- 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 )
- different dialects of SQL: Informix or Postgres
- XML for working with independent Visual Display Clients

They will be discovered and linked by the autoconfig `configure` script when you install Aubit4GL. This versatility forces you to have to make a number of choices. It is rather like buying an icecream in the US. You don't get anything till you have made a series of choices...

### 2.2.1 GTK

Aubit4GL 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 Aubit4GL, you need the GNOME GTK2 (Graphical Tool Kit) development library available at installation and run time. The GTK capabilities of Aubit4GL are now being surplanted by new Display Clients documented elsewhere in this manual. Given that there will be no further development of the GTK interface, you are advised not to use it.

## 2.2.2 PDFLib

An extension to the 4GL language allows Abit4GL to exploit a PDF (Portable Data Format) library to produce fancy reports. This is an optional feature and if unavailable when you build the Aubit4GL 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`

Get the PDF-Lite version which is free for non-commercial use.

When you have downloaded it and installed, you will need to make the directory which contains the file: `pdflib.so` available to the linker program `ld`. Either put the path to pdflib.so in a file
`/etc/ld.so.conf.d/aubit.conf` and as root run the command:
`ldconfig -v`
or add the directory to the environment variable LD_LIBRARY_PATH.

## 2.2.3 Which

Aubit4GL is available in two forms:

- Binary tarballs
- Source tarballs or CVS

## 2.2.4 Whither

Traditionally Linux systems expected you to install extra software in /opt or /usr/local. Times and practices have changed. It is better to create a separate disc partition (or slice in Unix parlance) for software that comes from outside the OS distribution. For Aubit4GL you need at least 300MB. Mount this partition on one of the following recommended directories. When you upgrade or reinstal the OS, you can unmount the partition, do the upgrade or reinstall, then mount the partition again and you will not have lost any of the Aubit4GL files.

We recommend that you download Aubit4GL to one of the following (in the author's order of preference):

- /local/opt
- /local
- /app

Avoid the traditional /usr and /opt as these are populated by Linux distributions and can get lost and overwritten on update or re-installation. Many opensource applications install by default into /usr/local. If you going to follow the advice above, and install into /local, run configure as follows:

```
./configure --prefix=/local
```

## 2.3 But first

### 2.3.1 Binary

Before you install Aubit4GL you may need to have installed other software

- ncurses 1.78 or later (and maybe ncurses-wide if you are using UTF-8 or other multibyte characters). Get this from your OS distribution.
- pdflib (if you intend to expoit PDFREPORTs). You will get this from pdflib.
- GTK2 (if you want to use the (discontinued) alternative graphic frontend). Get this from your OS distribution.
- ESQLC or ODBC for your database engine(s) unless using PostgreSQL
- Ventas Display Client (if you want to use a graphic frontend). Get this from ventas.de
- gSOAP (if you want to write web services). Get this from ???
- perl (if you want to use aace_perl, etc). Get this from your OS distribution

### 2.3.2 Source

The above will likely suffice for the binary installation.

But if you want to download Aubit4GL source and build it on your platform, because it is built in C, you will need the following as well:

- GNU C compiler: gcc and its prerequisites e.g.: GNU make
- autoconf tools
- bison, flex, m4
- glibc2
- gdb, valgrind
- cvs, svn, and their prerequisites
- curl

You should get any development packages associated with these prerequisites. (They usually have a suffix -dev or -devel). These will provide among other things the necessary .h files. If you are a software developer you have probably installed these already.

How you get these and install them varies with different distributions. If your OS distribution has packages for your version of the OS, then use those in preference to downloading and compiling from source.

- openSUSE.
  - Use GUI Yast2 to search and install
    or from the command line:
  - zypper search packagename
    then
  - zypper install package
- Ubuntu (or other Debian derived systems)
  - Use GUI aptitude or synaptic and search
    or from the command line:

  - apt-cache search packagename
    then
  - apt-get install package

The above will all install any prerequisite packages in addition to your chosen application.

For applications like pdflib which do not come packaged with the OS distribution, installation will vary but you will expect to install most of them by

- pointing a browser at the site (e.g. www.pdflib.org)
- find the download file you want
- right-click on the file to download it.
- mv the file to your chosen directory (e.g. /local/opt/pdflib)
- cd to the above directory
- extract the files from the download tarball. e.g.:
  tar xvfz appname-1.17-34.tar.gz
- cd to the directory extracted. e.g.
  cd appname-1.17-34
- find any README and/or INSTALL file and follow the instructions. They are usually something like
  - ./configure
  - make
  - make test
  - make install (as root)
  - configure as necessary PATH, LD_LIBRARY_PATH etc

## 2.4 Download

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

There are several possible sources for download

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

### 2.4.1 Filenames

Download files are tarballs. We name them as follows:

**Binary** aubit4glbin-i686-pc-linux-gnu-1.2-14.tgz

**Source** aubit4glsrc.1.2.14.tar.gz

The 1.2-14 is the version of Aubit4GL. The other stuff i686-pc-linux-gnu identifies the CPU architexture, OS, and libraries where the binary was built.

Binaries need to have been made on a system similar to yours. The above example works for openSUSE and other linux distros. You may need to look in subdirectories (e.g. ubunt/hardy, win32, etc) to find a match for your setup.

There is one common source file for all target distributions, hence the simpler names.

### 2.4.2 Tarballs

Tar means tape archive. Tarballs are created by bundling a whole directory structure into a single file for storage on tape or any other medium. They are usually compressed with gzip and given the suffix .tar.gz or .tgz

Before downloading save copies of critical config files (if you have them already). e.g.: $AUBITDIR/etc/aubitrc

Download is as described in the PDFLIB section above: point a browser at the filename and right-click. Experts can do it from the command line using the wget or curl commands.

Extract from the tarball:

```
export AUBITDIR=/local/opt/aubit4glbin
cd $AUBITDIR
tar xvfz /path/to/tarball
```

#### 2.4.2.1 CVS

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

1. cd $SRCDIR
2. Set an environment variable CVSROOT to:
   `:pserver:anonymous@cvs.sourceforge.net:/cvsroot/aubit4gl`
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`

## 2.5 Build

### 2.5.1 configure

After unpacking a tarball or updating an existing source application from CVS, you can change to the install directory and run the configure script.

Part of the autoconf tools suite of tools, the configure program is crucial to installing many applications and seeks out existing installed products and does what is necessary to link them to your target application. For both binary and source installations you should run autoconf to check that Aubit4GL will find all its sister libraries. Some useful command line examples:

```
./configure --help
./configure --prefix=/local
./configure ... > config.out
./confgure --with-pdf=/local
```

Whenever you run ./configure it writes to config.log. This is a very verbose (more than 7000 lines) log of all its actions and it can give you a precise indication of what it is doing to find each component.

When you redirect output to config.out, you get a 500 lines or so resume of lost and found elements. e.g.:

```
checking for main in -liksemel... yes
checking if we can use IksEmEl... yes
...
checking for main in -lsxml... no
checking sxml.h usability... no
checking sxml.h presence... no
checking for sxml.h... no
checking if we can use SXML... no
```

Use these file to determine if Aubit4GL is finding all the installed elements you expect it to find.

### 2.5.2 Binary

Nothing to do!

Mike has already done the hard work for you. If you are lucky and yours is the same distribution as his then the system libraries will have the same names. Otherwise, you will either have to create synonyms for them or abandon the binary and install the source instead.

### 2.5.3 Source

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

- cd /local/opt/aubit4glsrc
- Read the file README.txt
- Run the configure command:
  `./configure --prefix=/local/opt/aubit4glsrc`
  - This will search for all the prerequisites and options and build Makefiles appropriately
  - 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 the above failed where you expected it to succeed, figure out why, fix it, and try 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
  `Aubit4GL compiled successfully`

#### 2.5.3.1 Missing Software

Run the configure script to see which of these you have (or don't have) in config.log:

```
    ./configure --prefix=/local/opt/aubit4gl > config.out
```

Look in config.out or in config.log (for more verbosity) for what configure found and missed. Don't be too concerned if there are many things missed. e.g. you need at most one ODBC (on Linux: unix-odbc or iodbc). Similarly you will likely not need an ESQLC application for postgreSQL or SQLite3 or MySQL.

When config.out or config.log shows that you are missing something that you want, e.g. Jabber you will see what configure is looking for (in the case of Jabber: iksemel). Use your OS packet management software to search for and install the missing application and then run ./configure again.

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 iksemel run the the following:

```
rpm -qa | grep -i iksemel
iksemel-1.3-0
iksemel-devel-1.3-0
```

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

```
locate pdf
```

## 2.6 Connect

Now that you have Aubit4GL built on your system, whether form source or binary, you need to tell the operating system where to find everything. This amounts to

### 2.6.1 AUBITDIR

set AUBITDIR to the directory where installed Aubit4GL e.g.:

```
export AUBITDIR=/local/opt/aubit4glbin
```

### 2.6.2 PATH

set PATH to include the $AUBITDIR/bin e.g.:

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

### 2.6.3 ldconfig

Tell the linker `ld` where to find the Aubit4GL library files. This is done in one of two ways:

Either create a file: `aubit.conf` in `/etc/ld.so.conf`

Sample contents:

```
/local/opt/aubit4glbin/lib
/local/opt/aubit4glbin/plugins-1.2_14
/local/opt/PDFlib-Lite-7.03/libs/pdflib/.lib
```

and run the command (as root):

```
ldconfig -v
```

or add the above directories to LD_LIBRARY_PATH

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

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$AUBITDIR/plugins-1.2_14

mypdflib=/local/opt/PDFlib-Lite-7.03/libs/pdflib/.lib

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

Of course, replace my example above with values appropriate to your installation.

## 2.7 aubitrc

All Aubit4GL compilers read their configuration files in this order:

1. /etc/opt/aubit4gl/aubitrc (only if you run: make install)
2. $AUBITDIR/etc/aubitrc
3. ~/.aubit4gl/aubitrc
4. ./.aubitrc
5. $A4GL_INIFILE

Values in later files override those already set in earlier files. The aubitrc file in $AUBITDIR/etc is based on aubitrc.in and is edited to appropriate values by the configure program. To see what your current settings are run the command:

```
aubit-config -a
```

- `aubit-config` A4GL_SQLTYPE will show you what A4GL_SQLTYPE is currently set as
- `aubit-config -a` with show you the value of all Aubit4GL settings
- `aubit-config -ae` will show all the above as export VARIABLE=value (which is useful for generating aubitrc files)

You can edit these files with a text edit program such as vi(m) or (x)emacs or you can choose to use one of the programs that Aubit4GL supplies:

- quickguide.4ae
- configurator

quickguide is new with version 1.20 and is now the preferred method to set the environment.

### 2.7.1 Binary

The file $AUBITDIR/etc/aubitrc will be the one set up when Mike Aubury created the binary tarball. It will almost always be wrong for your setup. Change it. Whenever you install a binary make sure you have saved your aubitrc beforehand (e.g. in /local/etc/aubitrc or as oaubitrc in $AUBITDIR/etc) and then selectively reassert your entries from your saved aubitrc into the newly installed one. Run the `diff` command on the old and new and look for differences in settings. (The new one my have extra settings which you should add to the old aubitrc.

### 2.7.2 Source

Your new aubitrc will be built by merging a file aubitrc.in with settings the configure found. It should be close to what you want apart from things where you have a multiple choice (e.g. postgresql or sqlite3)

### 2.7.3 Plugins

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 plugins will be loaded by the compiler and/or 4GL program. At this stage you need only concern yourself with SQLTYPE and LEXTYPE. The defaults for all the other plugins will be appropriate.

#### 2.7.3.1 Informix

```
export A4GL_SQLTYPE=esql
export A4GL_LEXTYPE=EC
```

#### 2.7.3.2 PostgreSQL

```
export A4GL_SQLTYPE=pg8
export A4GL_LEXTYPE=C
```

#### 2.7.3.3 MySQL

```
export A4GL_SQLTYPE=mysql
export A4GL_LEXTYPE=C
```

#### 2.7.3.4 SQLite3

```
export A4GL_SQLTYPE=sqlite3
export A4GL_LEXTYPE=C
export DBPATH=$DBPATH:/path/to/sqlite3database
```

#### 2.7.3.5 Others

The others will need ODBC and esqlc

```
export A4GL_SQLTYPE=iodbc
export A4GL_LEXTYPE=EC
```

### 2.7.4 A4GL_SQLACL

To configure all your access to databases you should populate an ACL file. By default this is:

`$HOME/.aubit4gl.acl`

But you can set up an alternative one and set environment variable
`export A4GL_SQLACL=/path/to/file`
instead.

The ACL file should contain lines with 3 fields:

`dbname:username:password`

The password should be in plain text but will be automatically encrypted (albeit with a somewhat weak encryption) the first time the file is read. If there is an entry in this file with the database name at the top of the 4gl file, Aubit4GL should use that username and password for the connection.

You can also assign username and password to the SQLUID and SQLPWD environment variables.

For PostgreSQL, if it is just the port you want to change - you could set PG_PORT instead.

For PostgreSQL you can set PG_HOST to a hostname. e.g.:
`export PG_HOST=myhost`

## 2.8 DECIMAL format

To use continental formatting for decimal and numeric data: e.g. 200,12 where English uses 200.12, set the following:

```
    A4GL_ALLOWCOMMAINDECIMAL=Y
    A4GL_NUMERIC=,.
    LANG=it_IT.UTF-8
```

The example is Italian. Set LANG appropriately for your country.

Similarly you can set:

A4GL_DB_NUMERIC

A4GL_SCANF_NUMERIC

See the chapter on Environment Variables.

## 2.9 Remote connection to pg8:

If you're using the PG8 connector - then you can use any of :

```
PG_DBPATH
PG_PORT
PG_HOST
```

Syntax :

```
export PG_DBPATH=dbname[@server][:port]
export PG_PORT=1234
export PG_HOST=tstserv
```

If PostgreSQL is running on a remote machine, make sure you configure it with TCP/IP enabled. (If using ECPG, you'll need to check how to change the target host for that, but its normally PG_DBPATH )

- Specify the username and password by setting environment variables:
  A4GL_SQLUID
  and
- A4GL_SQLPWD.

These can be read from the aubitrc file if you prefer (by default `$HOME/.aubit4gl/aubitrc`

If you want a different username/password depending on the database you're connecting to, then you can use the Aubit4GL ACL file. The filename can be set explicitly in `$A4GL_SQLACL` setting. If that variable isn't set, it will be read from `$HOME/.aubit4gl.acl` and if that doesn't exist, it will try `/etc/aubit4gl.acl`

(On windows, it reads `$AUBITDIR/etc/aubit4gl.acl`, if that doesn't exist, it tries `c:/aubit4gl/aubit4gl.acl`)

The ACL file has the following format :

```
dbname:username:password
```

Put the password in plain text - the first time the file is read, the password will be encrypted. (This isn't a particularly secure encryption, but it is better than nothing!)

## 2.10 Check

### 2.10.1 Commands

The following commands should be created in $AUBITDIR/bin:

```
4glc
4GL_metrics.cgi
4glpc
a4gl
a4gl.4ae
a4glc
a4glpc
aace
aace_4gl
aace_perl
aace_runner
adbaccess
adbload2
adbschema
adecompile
afinderr
amake
amkmessage
aperform
asql_g.4ae
asql_i.4ae
asql_p.4ae
aubit
aubit-config
aupscol
c2pcode
c2pcode_fgl
checker
checker_fgl
configurator
convertsql
default_frm
```

```
    ecpg_wrap
    fcompile
    fdecompile
    fshow
    generate_aace
    genmake
    ide1.4ae
    layout_engine
    link_fgl
    loadmap
    mcompile
    mdecompile
    odbctest-iodbc3
    odbctest-unixODBC
    prepmake
    process_report
    quick_check_logrep
    report_viewer
    runforms.4ae
    runner
    runner_fgl
    runner_fgl_wrapper
    shtool
    sql_parse
    unmkmessage
```

The aace_perl program will not work unless you

```
    make install
```

or

copy the files `report.pm` and `using.pm` from
`$AUBITDIR/compilers/ace/perl_runner`
to where perl will find it.

To find where perl looks, run the command:

```
    perl -e 'print "@INC";'
```

This will output a space separated list of directories. If you want each directory on a separate line, assign \n to the perl print output separator variable `$"` as follows:

```
perl -e '$"="\n"; print "@INC";'
```

Depending on the version of perl the output might look like this:

```
/usr/lib/perl5/5.10.0/i586-linux-thread-multi
/usr/lib/perl5/5.10.0
/usr/lib/perl5/site_perl/5.10.0/i586-linux-thread-multi
/usr/lib/perl5/site_perl/5.10.0
/usr/lib/perl5/vendor_perl/5.10.0/i586-linux-thread-multi
/usr/lib/perl5/vendor_perl/5.10.0
/usr/lib/perl5/vendor_perl
.
```

To install, copy the file into one of these directories. e.g.:

```
su
cd $AUBITDIR/compilers/ace/perl_runner
cp report.pm using.pm /usr/lib/perl5/site_perl/5.10.0
```

### 2.10.2 Try to compile a simple 4gl hello

```
MAIN
  DISPLAY "Hello World"
END MAIN
```

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

Try to run it:

```
./simple
```

### 2.10.3 With database

Set up a database. This example for PostgreSQL:

```
export A4GL_SQLTYPE=pg8
export A4GL_LEXTYPE=C
createdb mydb
DATABASE mydb
MAIN
  DISPLAY "Hello World"
END MAIN
```

```
4glpc simpledb.4gl -o simpledb
```

Try to run it:

```
./simpledb
```

This assumes that you have an instance of PostgreSQL running. If not, read the Databases chapter, set up the instance, start it, and try the program again.

### 2.10.4 Test Programs

If both the above tests worked, congratulations!

For a more vigorous test:

```
cd $AUBITDIR/tools/test
make
./hello.4ae
./hello2.4ae
./hello_db.4ae
```

If all these programs work then you have almost certainly successfully installed at the basic Aubit4GL functionality.

### 2.10.5 For Informix

**2.10.5.0.1 Cient SDK** To use Aubit4GL with Informix engines, you need the Informix Client Software Kit which is free but only available if you have a validly licensed Informix engine.

Configure the CSDK (normally the $INFORMIXDIR/etc/sqlhosts etc)

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

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

**2.10.5.0.3 Set Up Aubit** Check that you have a `$AUBITDIR/lib/plugins-1.2_14/libSQL_esql.so`

```
export AUBITDIR=/local/opt/aubit4glsrc
export A4GL_SQLTYPE=esql
export A4GL_LEXTYPE=EC
```

**2.10.5.0.4 Try to compile a simple 4gl**

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

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

**2.10.5.0.5 Try to run it** Try to run that `simple.4ae`

# Chapter 3

# Set up

This chapter deals with setting up not Aubit4GL but the things that Aubit4GL uses such as ODBC, database engines, etc.

## 3.1 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). Not really needed now.
- sapodbc direct ODBC with SAPDB (a free opensource Database Engine up till version 7.3)

### 3.1.1 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.

#### 3.1.1.1 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.

#### 3.1.1.2 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.

#### 3.1.1.3 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` |

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

#### 3.1.1.4 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.

#### 3.1.1.5 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`

#### 3.1.1.6 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.

#### 3.1.1.7 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 Aubit4GL 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!

| Aubit4GL Native Connections | | | |
|---|---|---|---|
| SQLTYPE | RDBMS | Compiler | Comment |
| esql | Informix | esqlc | |
| esqlPG | PostgreSQL | ecpg | |
| pg | PostgreSQL | ecpg | obsoleted by pg8 |
| esqlSAP | SAPDB | cpc | |
| esqlQ | Querix | esqlQ | |

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

## 3.2 Databases

You will, in general, need to have a database engine available for Aubit4GL.

If you already have one installed (e.g. Informix, PostgreSQL, MySQL, SQLite3, or an ODBC application) skip this chapter.

### 3.2.1 Informix

We do not supply documentation about installing or upgrading Informix. It is proprietary software and IBM who supply it give ample documentation.

But remember that you will need to obtain and install the Informix CSDK (Client Software Development Kit) in order to get the esql package.

### 3.2.2 PostgreSQL

Most people using Aubit4GL will use it in conjunction with PostgreSQL and most of the current development effort in Aubit4GL is with PostgreSQL.

Aubit4GL now does not need PostgreSQL's esqlc package. Instead use the pg8 module.

#### 3.2.2.1 Overview

PostgreSQL is a fully ANSI and ISO compliant Relational Database Management System (RDBMS) which arose from a project called Ingres at the University of California, Berkeley under the direction of Professor Michael Stonebraker in the 1980s. Ingres was spun off to become a commercial product. Michael Stonebraker sold his developments in Object Relational technology to Informix and the University renamed its free Ingres as Postgres and continued development. The name change to PostgreSQL reflects its conformance with the official ANSI-ISO SQL3 public standard.

#### 3.2.2.2 Documentation

The official website for PostgreSQL documentation is:
`http://www.postgresql.org/docs/8.3`

#### 3.2.2.3 Installation

PostgreSQL is installed from RPM packages supplied with the Linux Distribution. OpenSUSE provides about 30 Postgres packages of which the following are essential:

- postgresql
- postgresql-server
- postgresql-devel
- postgresql-libs

These packages are best and most easily installed using the SUSE Yast2 program.

Take the route: Software -> Software Management. Enter: *postgres* in the search field and check the boxes for the above packages (and any others you want to install).

Alternatively you may install from the shell command line:

```
yast2 -i postgresql
yast2 -i postgresql-server
...
```

The simple package name is sufficient. Yast2 will find the full path and filename of the package.

You can also use the zypper equivalent of yast2:

```
zypper install postresql-server
...
```

Once installed, the RPM system will create a user `postgres` intended to be the superuser for PostgreSQL databases. (In principle the database superuser should be a different user from root).

For Debian derived distributions of Linux such as Ubuntu the equivalent installer is

```
apt-get install postgresql
...
```

#### 3.2.2.4 Instances

PostgreSQL shares the same notion as Informix of a database instance being a collection of databases with common resources (known as a *cluster* in PostgreSQL parlance).

**3.2.2.4.1 initdb** Instances are created with the command `initdb`. For example:

```
initdb --pgdata=/local/data/dev
```

In the example, dev is our name for the development instance of the database. You create a directory like /local/data/dev in our example and pass it to the option: `--pgdata`

By default, postgresql will build an instance in `/var`. This is vulnerable to being lost when you upgrade or reinstall the operating system. It is much better to create disk area outside the normal operating system directories. Hence the use of the `--pgdata=/local/data` in our example.

`initdb` populates its given directory with all the system files, common catalogue tables, and a database called `template1` which is copied when any user databases are subsequently created in the instance.

To connect to the instance created, as a user you set the environment to include:

- `export PGDATA=/local/data/dev` (or whatever is the directory where the instance has been created)
- `export PGPORT=5432` (or 5433 or 5434) as the TCP/IP port for frontends to connect to the database.

For more details, use the man or info commands: e.g. `info initdb`

The backends need PGDATA to identify the directories where the database files are.

The backends and frontends need PGPORT (or command line options) to identify which instance of the program: /usr/bin/postgres to communicate with.

To test that all is well, try the command:

```
env | grep PG
```

and you should see PGDATA, PGPORT, PGDATABASE, and PGDATESTYLE with appropriate values.

**3.2.2.4.2 pg_ctl start** Having created a database instance, you must start the postgres engine using the commands:

```
    pg_ctl start -l /local/data/dev.log
```

The -l option determines where the instance will write its log.

Unlike Informix's threaded architecture, PostgreSQL is single-threaded and a new `postgres` backend process is spawned as each new connection is made to the engine instance.

Another check you can make to see if postgres instances are running is the command:

```
    netstat -pa | grep postgres
```

This will return a line of output showing UNIX domain sockets with an endpoint something like:

```
    /tmp/.s.PGSQL.5432
```

where 5432 is the TCP/IP port for the socket.

**3.2.2.4.3 pg_ctl stop** The pg_ctl command is also used to stop the postgres engine:

```
    pg_ctl stop
```

There are in fact 3 stop options for pg_ctl: smart (the default), fast, and immediate

- `pg_ctl -m s[smart]` waits for all users to disconnect before shutting down
- `pg_ctl -m f[ast]` rolls back all current transactions then shuts down
- `pg_ctl -m i[mmediate]` immediately shuts down and will force recovery on restart

Calls to `pg_ctl {start|stop}` can be used at startup and shutdown via the usual `/etc/init.d` scripts.

PostgreSQL will not allow the pg_ctl command to be run by any user other than the Postgres super-user.

#### 3.2.2.4.4 createdb

Having created an instance, and having started postgres with pg_ctl start, you may now create a database. There are 2 ways to do this

- `createdb mydb`  
  `or`
- Run `psql` or `adbaccess` and use the SQL statement:  
  `create database mydb`

The shell command `createdb` is just a wrapper for the SQL statement. In both cases, the database name defaults to the user's name. We override this by setting the environment variable: PGDATABASE e.g.

`export PGDATABASE=mydb`.

### 3.2.2.5 Environment

Set up the following environment variables for PostgreSQL:

```
export PGDATABASE=mydb
export PGDATA=/local/data/$INSTANCE
export PGPORT=5432 # or 5433 or 5434
export PGDATESTYLE='SQL, dmy'
```

The character encoding used by the PostgreSQL engine will default to the Linux LOCALE which is UTF-8. This seems to work fine and we can leave it that way unless you run into trouble with your LANG environment and special smbols like the British pound sign. UTF-8 is likely to become a universal practice in the future.

By default, PostgreSQL tries to connect to a database with the user's login name. The variable PGDATABASE overrides this. When you run commands like: `psql` it will connect to `your database` by default.

By default, PostgreSQL places its data in the `/var` partition. We regard this as undesirable as it is vulnerable to loss or corruption when the OS is reinstalled or upgraded. We install it in the partition:`/local`. To ensure this, set the environment variable PGDATA.

By default, PostgreSQL listens on port 5432 for frontend programs seeking to connect. When we run multiple instances of PostgreSQL, we need to have a separate port assigned to each instance. If, for example, you have 3 instances: for development, training, and production, you could make the following arbitrary assignments:

- `dev:  export PGPORT=5432`
- `tng:  export PGPORT=5433`
- `prd:  export PGPORT=5434`

The frontends (e.g. psql, pg_ctl, createdb, etc.) all rely on the environment variable PGPORT if it is set. PGPORT can be overidden by command line options, otherwise the default is to use port 5432.

We need to set PGDATESTYLE to SQL, dmy to get your style of date format. The default is American style middle-endian dates (mdy). You can configure European style date within SQL also using the syntax: SET datestyle TO "SQL, dmy";

Set the following environment variable for Aubit4GL:

```
export A4GL_LEXTYPE=C
```

export A4GL_SQLTYPE=pg8

The above environment variables may be set up in

- `/etc/profile.local` as the default for all users
- `~/.profile` in each user's home directory (in /export/username)
- in other directories dotted from `~/.profile` (using the shell . command)

#### 3.2.2.6 Maintenance

**3.2.2.6.1 vacuumdb** The `vacuumdb` command is PostgreSQL's equivalent to the Informix `UPDATE STATISTICS` command. You can run it from the command line or within psql or adbaccess as an SQL statement. Its Synopsis:

```
vacuumdb [--full] [--analyze]
```

The equivalent SQL syntax is: `VACUUM FULL ANALYZE`

with the FULL and ANALYZE optional.

For documentation on vacuumdb, use info or man:

```
info vacuumdb
```

**3.2.2.6.2 pg_dump** The PostgreSQL command: `pg_dump` is used in place of Informix's dbschema and dbexport commands.

Use `pg_dump -s` for a schema of the whole database.

Use `pg_dump -st access` for a schema of the access table.

Use `pg_dump` for a complete export of the schema and data. The database will be placed in a single file of SQL statements which are sufficient to reload the data using the commands: `psql` or `adbaccess`

#### 3.2.2.7 Commands

The installation of PostgreSQL results in the following commands being installed in `/usr/bin`:

```
clusterdb
createdb
createlang
createuser
dropdb
droplang
dropuser
pg_dump
pg_dumpall
pg_restore
psql
reindexdb
vacuumdb
initdb
ipcclean
pg_controldata
pg_ctl
pg_resetxlog
postgres
postmaster
rcpostgresql
--- contributed --
oid2name
pgbench
vacuumlo
```

Many of the above are simply shell wrappers around the same command in SQL. Some (e.g. `postmaster` and `postgres`) are meant to be invoked only by other commands (e.g.: `pg_ctl`). Use the commands `info` or `man` to learn more.

#### 3.2.2.8 psql

`psql` is the PostgreSQL SQL interpreter. Unlike Informix's `dbaccess` it is not a form mode menu program but instead uses the Linux readline library to buffer a historied command line interface very similar to the Unix shells.

Use psql to execute SQL statements.

- type `\h` for help with psql's SQL syntax
- type `\?` for help with psql's own internal commands

To use psql to execute an SQL file:

```
psql -f filename [mydb]
```

You can echo SQL statement(s) into psql as follows:

```
echo "select * from agent;" | psql
```

This is faster for one-line statements as psql exits immediately after execution, but be careful to protect special characters (like ;) from the shell by quoting.

Another useful argument for psql is -l which lists the available databases and exits immediately.

Aubit4GL provides `adbaccess` as a workalike for Informix's dbaccess. This is fully featured when connected to an Informix database but still lacks some capabilities with PostgreSQL as a backend. You will likely prefer `adbaccess` for tables with rows of many columns (such as the table: access).

We recommend that you not use PostgreSQL's esql package. Aubit4GL's pg8 library provides all the necessary connectivity with the PostgreSQL engine.

#### 3.2.2.9 Stored Procedures

PostgreSQL supports stored procedure languages including its own native PL/pgSQL. This is similar to Informix's SPL but has quite different syntax.

We need to use PL/pgSQL to perform the translation of variables containing CONSTRUCTed clauses from Informix MATCHES statements into PostgreSQL regular expression (RE) statements (using the ~ operator).

The algorithm of conversion is:

- start the re with a ^ – RE BOL (beginning of line)

- replace ? with . – RE for any single char
- replace * with .* – RE for any single char followed by zero or more others
- replace . with \. – RE for a literal .
- otherwise just copy the char
- At the end if the final char is not * add a $ (RE EOL end of line)

##### 3.2.2.9.1 PL/pgSQL matches function

```
CREATE FUNCTION matches_to_regexp(str text, esc text)
  RETURNS text
AS $$
DECLARE
   lv_rval text;
   lv_c char;
   lv_cnt int4;
BEGIN
   lv_cnt:=0;
   lv_rval:='^';
   for lv_cnt in 1..length(str) loop
       lv_c:=substr(str,lv_cnt,1);
       if lv_c='?' then
           lv_rval:=lv_rval||'.';
       elsif lv_c='*' then
           lv_rval:= lv_rval||'.*';
       elsif lv_c='.' then
           lv_rval:=lv_rval||E'\.';
       else
           lv_rval:=lv_rval||lv_c;
       end if
   end loop;
   if substr(str,length(str),1) !=
         '*' then
       lv_rval := lv_rval || '$';
   end if;
```

```
        return lv_rval;
    END;
    $$
        LANGUAGE plpgsql;
```

**3.2.2.9.2 $$ quoting** In the above code, $$ is a quoting mechanism which allows the usual single and double quotes to be used inside the stored function without interference with the syntax of the function declaration. The $$ can also be replaced with an identifier between the $ symbols: e.g. `$fred$` and differently identified quotes can nest within each other.

**3.2.2.9.3 E escaping** In the above stored procedure the E'\.' expression tells pg to treat the \. literally and not interpret the \ as an escape mechanism.

The PL/pgSQL syntax also allows `CREATE OR REPLACE FUNCTION f()` during development to obviate the need for a `DROP FUNCTION f()` when modifying and testing.

**3.2.2.9.4 PL/pgSQL install** You may need to run the following SQL statements before the above will be accepted:

```
CREATE LANGUAGE plpgsql;
CREATE FUNCTION plpgsql_call_handler()
RETURNS opaque AS '/usr/lib/postgresql/plpgsql.so',
'plpgsql_call_handler' LANGUAGE c;
ALTER FUNCTION public.plpgsql_call_handler()
OWNER TO informix;
```

A good technique is to add the above stored procedure statements to the `template1` database so that in the event of dropping and reinstalling your database the `matches_to_regex()` function will be automatically built again by the statement: `CREATE DATABASE mydb;`

### 3.2.3 MySQL

Install various Mysql components e.g. on Ubuntu

```
apt-get install mysql-common \
   libmysqlclient15-dev \
   mysql-server mysql-client
```

You should now have mysql running.

MySQL setup example:

#set up the MySQL user userpt18z97 but first of all set up a root passwort in mysql for safety

#start MySQL as root, we assume you run MySQL only locally for the time being

As root:

```
mysql
mysql>update user
   set password=PASSWORD("x5a4p4i7")
   where User='root';
```

From now on you start MySQL as root with "mysql -p"

Create your initial database

```
mysql>create database mydb;
```

Tip: if you want to be able to switch databases in your programs create one database emptydb which has all the tables you use in mydb but without data. This is then used in the .per and the .4gl at compile time and you only switch later to the real database mydb with data in your 4gl application. Naturally emptydb must reflect the exact db-schema of mydb at all times.

Grant whatever privileges you want the user to have, below is the maximum (not advisable in production)

```
mysql>grant all privileges on mydb.*
  to 'userpt18z97'@'localhost'
  identified by 'p5z1m7u9'
  with grant option;
```

All further operations on the db (e.g.: `create table`...) should be done only as user `userpt18z97`

...

#Spanish keyboard with Euro sign for example, 7-bits chars (ncurses) unicode would be a different story

```
export LANG=es_ES@euro
export A4GL_SQLTYPE=mysql
export A4GL_SQLUID=userpt18z97
```

#can be different from logon password, of course

```
export A4GL_SQLPWD=p5z1m7u9
export A4GL_UI=TUI
export A4GL_DBNAME=mydb
export DBDATE=dmy4.
```

# for example with putty settings example Spanish keyboard : Terminal/keyboard "linux",

# putty contd: ...Window"80"x"25",Translation "ISO-8859-15:1999 (Latin-9, "euro")", etc...

```
export TERM=linux
```

Our thanks to Karl Rumpf (klrumpf@gmail.com) for this advice.

### 3.2.4 SQLite3

```
export A4GL_SQLTYPE=sqlite3
export A4GL_LEXTYPE=C
```

DBNAME needs to contain the full path to where your SQLite3 database is. e.g.:

```
export DBPATH=/home/john/data
```

When the 4glc compiler encounters the statement:

```
DATABASE mydb
```

It will search the DBPATH directories for a database file: `mydb`

SQLite3 is unusual in that it does not have an SQL statement to create a database. You create a database by supplying the database path and name on the calling line. e.g.:

```
sqlite3 /home/john/data/mydb
```

If the mydb file does not exist, it will be created. Note that SQLite3 needs the full path to the database file unless it is in the current directory. Aubit4GL syntax does not accept a path in the DATABASE statement or the CONNECT TO statement, hence the need to supply the path in the DBPATH variable. Once Aubit4GL has found your full pathname for the database in $DBPATH, it will subsequently use that full path in all its communication with the SQLite3 library.

### 3.2.5 SQLServer

Bernard Moreton (itman@tnauk.org.uk) has this advice for setting up Aubit4GL to work with SQLServer.

SQLServer is a Microsoft database engine available only on Windows.

To access SQLServer, from a Linux machine you need to install

- freeTDS
- ODBC

I run Aubit4GL against a (pre-installed) MS SQLServer database. This is a matter of 'needs must', and I would not necessarily recommend that database engine as first choice from cold.

On MS Windows, there is little to do. Users run a batch file to set the environment and start the core menu module:

```
@echo off
set AUBITDIR=y:\a4gl
set A4GL_HOME=y:\a4gl
set PATH=%AUBITDIR%\bin;%AUBITDIR%\lib;%AUBITDIR%\local-bin;%PATH%
set A4GL_SQLTYPE=odbc32
set A4GL_DBDATE=DMY4/
set A4GL_UI=TUI
set LOCAL_BIN=y:\a4gl\local-bin
set A4GL_CLASSPATH=y:\a4gl\local-bin
set ALLOWUSINGEXT=Y
set A4GL_ALWAYS_EXTENDED_FETCH=Y
set A4GL_STATUSASCOL=Y
set ALLOWDYNAMIC=Y
%AUBITDIR%\local-bin\tnauk.exe
exit
```

Permissions for users are all set up through the normal MS database system.

On W32, Aubit is used only for reporting purposes, since the database is the core of a proprietary membership system; but running from Linux, where all the development is done, I also have a number of modules that write to the database under controlled conditions.

On Linux, freeTDS is required; and sqlconvert/INFORMIX_SQLSERVER.cnv is already present in the standard build. Access permissions are set up by the normal .aubit4gl.acl

# Chapter 4

# Problems

## 4.1 Curses

If you want Aubit4GL 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.

### 4.1.1 Wide Characters

Where in the past a character meant 1 byte (using either the lower 7 bits for 127 possible characters, or later 8 bits for 255 possible characters). The mapping of the numbers to characters is called an encoding. Increasingly Linux distributions and postgreSQL are configured by default to use the UTF-8 encoding. Aubit4GL can work with UTF-8 which is a multibyte encoding but only if a special wide version of curses is installed.

### 4.1.2 Encodings

The following encodings are now common:

**Unicode** multibyte character - ie - a single character needs more that one byte to store it. Chinese, Japanese, and Korean - need lots more than 255 possible characters. The Java programming language uses Unicode as its encoding (both internal and external).

**UTF-8** unicode using 8 bit characters - this uses the top bit to say its multibyte. It can store ascii 0-127, then anything over that needs more than one byte. The number of bytes actually used can be up to 4 - but can also be 1 (for <=127) or 2 etc.

**iso8859-1** http://en.wikipedia.org/wiki/8859-1 Common single byte 8-bit character encoding which uses the characters between ASCII 128 and 255 to store characters common in western alphabets.

**iso8859-15** similar to 8859-1 - but with some values changed.

The C library has functions like `isprint()` which determine if a character is printable. The `curses` library uses this internally for determining whether we can display a character. If LANG is not set then the default character set is the ASCII characters 1-127 (the first 31 are mostly control codes which are not printable).

This means that if LANG is not set the 0xa3 (a UK pound) won't be printable and so you cannot use it. Likewise for accented vowels used in French, or Umlauts over o's and u's in German etc.. If you set LANG=en_GB, it sets the character set for `isprint()` to the iso8859-1 and 0xa3 becomes printable. Note: this is not unicode or multibyte etc. Its just plain old ASCII.

On modern Linux distributions of Ubuntu, OpenSUSE, etc in the UK, LANG is often set to en_GB.UTF-8 (meaning British but using UTF-8 character encoding). With UTF-8, the high bit is used to mark it as a multibyte character, so 0xa3 is no longer available for the pound ("£") - we need a new multibyte character instead. In UTF-8 it is in fact 2 bytes: c2a3.

If, however, we want to use the Latin (western) character set, we need a non-UTF-8 locale in LANG and we can use `UI=TUI`

If we want to use UTF-8, we need a UTF-8 locale (e.g. en_GB.utf-8) and we need to use `UI=TUI_wide`. TUI_wide though will only be compiled up if it detected the wide ncurses which is unfortunately not always compiled by default in Linux distributions.

To get Aubit4GL to work with multibyte character sets such as UTF-8 you may have to

- install the wide character ncurses library
- compile Aubit4GL from source so that when you `./configure` it, it will compile and generate a `libUI_TUI_wide.so` in the plugins directory.
- export A4GL_UI=TUI_wide
- set your LOCALE so that `isprint()` returns true for the higher characters..

If you are using ISO8859-1 or similar, (i.e. not multibyte), you can check for printable characters with a C program something like :

```
#include <locale.h>
#include <ctype.h>
#include <stdio.h>
main()
{
  int a;
  setlocale (LC_ALL, "");
  for (a=0;a<255;a++)
  {
    int isp;
    if (isprint(a))
    {
       printf("%3d %c\n",a,a);
    }
    else
    {
       printf("%3d Not printable\n",a);
    }
  }
}
```

You can try with different LANGs, but if the character is not printable, the ncurses form library will not allow you to use it.

### 4.1.3 LENGTH

Regardless of whether you have multibyte characters, 8-bit characters, or simple ASCII, the 4GL `length(str)` function returns the byte-count of the string which will be more than the character count if there are multibyte characters in the string. Also the function `substr(i)` may point into part of a multibyte character and produce unexpected output (or an unprintable character).

## 4.2 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.

Aubit4GL 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 free and opensource engine up to version 7.4 with threaded architecture. The engine was SAP's tried and true commercial product and was solid and very fast. Unfortunately, MySQL acquired the rights to develop the next version of SAPDB (to be renamed MAXDB) and the interfaces are no longer free (GPL but not LGPL licensed). SAPDB has dropped below the radar now. It will probably still work if you have the old library files however - but we no longer test for it or include it in the binary releases.

- SQLite3 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 with your distribution or failing that from `www.sqlite.org`

- MySQL has been absorbed first by SUN MicroSystems then when Oracle bought out SUN, by Oracle. MySQL had a policy of free if you are free, commercial if you are commercial. Now that it is owned by Oracle who, of course, have their own proprietary database engine, who knows what will be the fate of MySQL? In the meantime, Aubit4GL works with MySQL.
- Any other database engine with an ODBC interface including PREPARE and SCROLL CURSOR statements.

# Chapter 5

# Modules

### 5.0.1 Choices

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 | libEXREPORT_??? |
| 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) | libPACKER_??? |
| A4GL_RPCTYPE | Specify the Remote Procedure Call handler | libRPC_??? |
| A4GL_SQLTYPE | Specify the SQL handler | libSQL_??? |
| A4GL_SQLDIALECT | Specify the SQL dialect to use | libSQLPARSE_??? |
| A4GL_UI | output module to use to display the program | libUI_??? |

Aubit4GL's libraries are created in the directory $AUBITDIR/plugins-1.2_14. With each new version of Aubit4GL a new plugins directory is created. This allows you to revert to earlier versions more easily.

Most of them have filenames of the form `libXXX_YYY.so` (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.

This is my list. Yours will probably be different! :

- `libaubit4gl.so` `libbarcode.so` `libchannel.so`
- `libA4GL_:`  
  `file.so HTML.so string.so`  
  These are miscellaneous extra libraries.
- `libDATA_:`  
  `menu_list.so module.so`  
  `report.so struct_form.so`  
  These are internal libraries for reading data files.
- `libESQL_`  
  `INFORMIX.so 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`
- `libEXDTYPE_mpz.so`  
  Example extended datatype library (implements the GNU mpz datatype).
- `libEXREPORT_:`  
  `NOPDF.so PDF.so`  
  Extended report handling. `libEXREPORT_PDF.so` relies on having pdflib installed. It will not be generated otherwise. PDF reports are experimental.
- `libFORM_:`  
  `GENERIC.so NOFORM.so 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)...

- `libHELP_std.so`
  Always set to std - can be ignored
- `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...
- libLOGREP_:
  `CSV.so TXT.so`
  Logical report handling. Converts logical reports to different formats
- `libMENU_NOMENU.so`
  GUI Menu handling. Obsoleted (probably).
- `libMSG_NATIVE.so`
  `Ignore`
- `libPACKER_:`
  `MEMPACKED.so PACKED.so`
  `PERL.so XDR.so XML.so`
  This specifies the packer to use for reading and writing data files. The library is specified via the `A4GL_PACKER` variable. Use MEMPACKER for embedding forms into a program. Don't use 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.
- `libRPC_NORPC.so libRPC_XDR.so`
  Specifies which RPC protocol to use - advanced stuff - still experimental.
- `libSQL_:`
  `esql.so esql_s.so FILESCHEMA.so`
  `ifxodbc.so nosql.so`
  `mysql.so pg8.so pgodbc.so`
  `sqlite3.so 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

- `libUI_:`
  `CONSOLE.so lHL_TUIN.so HL_TUI.so`
  `TUI.so TUI_s.so HL_GTK.so`
  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. Use this for programs to be called by cron, at, etc where they will run without a terminal (real or virtual) connected to stdout.

  TUI - The traditional output for dumb terminals (real or emulated)

  TUI-WIDE - for wide characters (e.g. UTF-8, Unicode, etc)

  HL_TUI - Was to be the next version of TUI, abstracted to help make other HL_.. modules. There will no further development of this.

  HL_TUIN - Ignore

  HL_GTK - GTK version. This will not be developed further and will be superceded by the new generation of Visual Display Clients.

- `libXDRPACKER_:`
  `menu_list.so module.so`
  `report.so struct_form.so`
  This is a helper module when `FORMTYPE` etc are set to XDR. These contain the actual XDR routines.

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

# Chapter 6

# Aubit4GL Compilers

## 6.1 A4GL compilers

Aubit4GL 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
- `amkmessage` which creates a binary help file from source

On Linux/Unix systems these programs may be 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.

## 6.2 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.

`4glc` has a synonym: `a4glc` which you can use when you want to exploit the shell history ! mechanism. e.g.
`!a4`
will cause the most recent command starting with `a4` to be re-executed. (This is likely to be `a4glc modname.4gl`). If you try to use
`!4g`
instead, the shell with rerun the 4th command line from the history file.

### 6.2.1 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 libraries will be found) |
| -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 include files can be found) |
| -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 verbosity) |
| -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, WRITE, or PERL |
| -td TYPE or –lexdialect TYPE | Specify the output language dialect for ESQL/C generation (INFORMIX or POSTGRES) |
| -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-Generate(Default) |
| –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 |

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=POSTGRES, 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.

## 6.3 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 wrapper instead.

## 6.4 Compiling forms

```
$ fcompile file.per
```

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

## 6.5 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.

## 6.6 Compiling menu files

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

Chapter 7

# 4GL Language

## 7.1 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.

## 7.2 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

## 7.3 Short Intro to x4GL

- 4GL Programs
- Structure of a program
- DATABASE section
- GLOBALS section
- Functions
- MAIN block
- DEFINE section
- 4GL Commands

### 7.3.1 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.

#### 7.3.1.1 Structure of a program

database section

globals section

function/report/main block

.

.

.

.

function/report/main block

#### 7.3.1.2 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.

#### 7.3.1.3 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.

#### 7.3.1.4 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`

#### 7.3.1.5 MAIN block

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

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

#### 7.3.1.6 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).

#### 7.3.1.7 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!

#### 7.3.1.8 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.

#### 7.3.1.8.1 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"
```

Records may also be nested and used in conjunction with arrays. The following are all therefore valid:

```
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
```

#### 7.3.1.9 Associative Arrays

These are an Aubit4GL extension.

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.

**7.3.1.9.1 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
```

#### 7.3.1.10 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).

#### 7.3.1.11 DEFINE NEW TYPE

As from version 1.2 of Aubit4GL you can define new datatypes with syntax:

```
DEFINE NEW TYPE mytype typestatement
```

e.g.

```
DEFINE NEW TYPE contract RECORD
     level smallint,
     denomination char(8)
   END RECORD
DEFINE mycontract contract
LET mycontract.level = 3
LET mycontract.denomination = "Spades"
```

The new type definition saves the programmer the tedium of replicating a RECORD .. END RECORD definition for instances such as passing records to functions or reports.

#### 7.3.1.12 Packages

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

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

or

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

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

You can use a . as a synonym for :: in the above library calls.

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)

## 7.4 Quick Reference

The material in this section should be the same as the separate document a4glqref but is typeset differently.

### 7.4.1 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.

### 7.4.2 Constants

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

### 7.4.3 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.

### 7.4.4 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"`)

### 7.4.5 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

### 7.4.6 Aubit4GL Operators

```
[NOT] IN ( {expr [ ,...]
   |selectstatement})
[NOT] EXISTS ( selectstatement )
```

### 7.4.7 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

### 7.4.8 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
```

### 7.4.9 Table Privileges

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

### 7.4.10 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. |

### 7.4.11 4GL Statement Syntax

```
ALLOCATE ARRAY name[size] #[ and ] literal here
ALLOCATE ARRAY dynname[rows, cols]
ALLOCATE ARRAY dynname[panes, rows, cols]
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([COPYOF][arg][,...])
      [RETURNING arglist]
    CASE [(expr)]
      WHEN {expr | booleanexpr}
        statement
        ...
        [EXIT CASE]
      ...
      [OTHERWISE]
        ...
        [EXIT CASE]
        ...
      END CASE
    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]
   [,...]


           VIA viafunc


   {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 convrepname TO
     {"filename"|PIPE program|PRINTER|[E]MAIL}
     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 (colname)
  DEFINE name ASSOCIATE [CHAR](n)
     with ARRAY[m] OF datatype
  DEFINE NEW TYPE id datatype


DELETE FROM tabname
   [WHERE {condition|CURRENT OF cursor}]
DELETE USING linkid
```

```
DISABLE FORM name
DISPLAY {BY NAME varlist
  | varlist TO {fieldlist|screenrec[[n]].*}
       [,...]
     | AT screenrow,screencol]}
   [ ON KEY (keylist)
      statement
      ...
      [EXIT DISPLAY]
   ...
   END DISPLAY]
DISPLAY ARRAY id
   [SLICE (field1 {THROUGH|THRU} field2)]
   TO screenarray.*
   [ ATTRIBUTE[S]( attlist |

           [INSERT|DELETE ROW =[=] {TRUE|FALSE}]
           [UNBUFFERED]
           [CURRENT ROW =[=] expr
           [[MAX]COUNT =[=] n]
           [REVERSE|colour
       )]
      {[ON CHANGE field]|
       [ON {IDLE|INTERVAL} n {HOURS|MINUTES|SECONDS}]|
       [ON ACTION id]|
       [ON ANY KEY]|

   [ON KEY (keylist)]
      statement
      ...
      [EXIT 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
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 REPORT
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([[COPYOF] arg] [,...]})
   [ 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]

   IF boolexpr THEN
     statement
     ...
     [ELIF|ELSIF
       statement
       ...]
      ...
     [ELSE
       statement
       ...]
     END IF]
   IGNORE ERRORS (n [, ...]) FOR 4glstatement
   IMPORT PACKAGE name


INITIALIZE varlist
  {LIKE collist| TO NULL}
```

```
INPUT ARRAY array
  SLICE(field1 THROUGH|THRU field2)
  [WITHOUT DEFAULTS]
  FROM screenarray.* [HELP n]


           [ATTRIBUTE[S](
             [INSERT|DELETE ROW =[=] {TRUE|FALSE}]
             [UNBUFFERED]
             [CURRENT ROW =[=] expr
             [[MAX]COUNT =[=] n]
             [REVERSE|colour
           ] [, ...])


    [{BEFORE {ROW|INSERT|DELETE|FIELD list}
    [,...]
    |AFTER {ROW|INSERT|DELETE|FIELD list|
       INPUT}[,...]
     |ON KEY (keylist)}

          [ON CHANGE field
          [ON {IDLE|INTERVAL} n {HOURS|MINUTES|SECONDS}]|
          [ON ACTION id]|
          [ON ANY KEY]


     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)]
```

```
   LOAD FROM filename
     [USING FILTER filtfunc]
   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
    MESSAGEBOX message


MOVE WINDOW
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)
PREPARE id from "charexpr"
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
RETURN  exprlist
REVOKE { tabpriv ON tabname
   | CONNECT | RESOURCE | DBA}
   FROM {PUBLIC | userlist
ROLLBACK WORK
ROLLFORWARD DATABASE name
RUN command [RETURNING n
   |WITHOUT WAITING]

  RUN program WAITING FOR expr
     [WITH {MESSAGE|ERROR}] expr
  RUN program {WAIT|EXIT}
  RUN CLIENT progname
  SCHEMA dbname

SCROLL {fieldlist| screenrec.*}[,...]
   {UP|DOWN}[BY n]
SELECT sellist [INTO varlist] FROM collist
   [joinclause] [fromclause]
   [groupclause [havingclause]]
   [orderclause]
```

```
SELECT USING linkid

SET BUFFERED LOG
SET CONSTRAINTS ALL IMMEDIATE
SET CURSOR
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
SET LOG
SET PAUSE MODE OFF
SET PAUSE MODE ON
SET SESSION TO name

SET SQL DIALECT TO "{INFORMIX|ORACLE|...}"

SHOW OPTION "optname"
SHOW WINDOW name
SLEEP n

SORT array USING sortfunc [limit n]

SQL [WITH[OUT] CONVERSIONS]
   sqlstatement [,...] END SQL

START EXTERNAL FUNCTION

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

[WITH [TOP|BOTTOM|LEFT|RIGHT MARGIN n ...  | PAGE
LENGTH n]
```

```
   STOP ALL EXTERNAL


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

   CONVERTIBLE

}]
       |

   [[WITH TOP|BOTTOM|LEFT|RIGHT MARGIN]...  |PAGE
   LENGTH n]


   TERMINATE REPORT


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
```

### 7.4.12 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
```

### 7.4.13 Report Statement Syntax

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

### 7.4.14 Report Expressions

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

```
LINENO
PAGENO
TIME
WORDWRAP
```

### 7.4.15 PDF Report Syntax

PDF reports are an Aubit 4GL extension.

- r, g, b are values between 0.0 and 1.0 for red, green, blue
- TOP variants of LINE TO etc, put origin for x.y at top left. (PDF usually has origin at bottom left.)

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}]
  [LEFT MARGIN nval}
  [RIGHT MARGIN nval]
  [TOP MARGIN nval]
  [BOTTOM MARGIN nval]
  [PAGE LENGTH  nval]
[ORDER [EXTERNAL] BY sortlist]
FORMAT
  { EVERY ROW
   | {[FIRST] PAGE HEADER
      |PAGE TRAILER
      |ON EVERY ROW
      |ON LAST ROW
```

```
              |{BEFORE|AFTER} GROUP OF argvar}
                statement| pdfstatement
                ...
                [...]}
       END PDFREPORT
```

#### 7.4.15.1 PDF Report Expressions

```
       COLUMN nval
       reportexpression
```

#### 7.4.15.2 PDF Statements

```
       PRINT IMAGE blobvar AS
          "{GIF|PNG|TIFF|JPEG}"
          [SCALED by x.n, y.n}
       PRINT IMAGE "filename"
       SKIP {BY|TO} nval
       BOOKMARK expr [,expr] [ RETURNING index]
       PRINT BARCODE [NO TEXT] [AT x, y] [WIDTH w HEIGHT h]
       SET BARCODE TYPE {"2"|"5"|"8"|"13"|"25"|"39"|"QR"}
       SET BARCODE TYPE {"128A"|"128B"|"128C"}
       FILL
       FILL STROKE
       LINE TO [TOP] y, x
       MOVE TO [TOP] y, x
       SET COLOR r, g, b #all 0-1
       SET FILL COLOR r, g, b
       SET FONT NAME name
       SET FONT SIZE n
       SET PARAMETER name, value
       SET STROKE COLOR r, g, b
       STROKE
       CALL PDF_FUNCTION(arglist)
```

#### 7.4.15.3 PDF_FUNCTION arglists

There are many libpdf functions. For a full list look at the PDFlib documentation. Here is an example:

```
"set_parameter", "{underline|...}", "{true|false|...}"
```

Note: Font names are case sensitive.

## 7.5 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.

### 7.5.1 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 |

### 7.5.2 Standard 4GL Operators

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

| Operator | Comment |
|---|---|
| ascii(n) | returns a char,<br>e.g. ascii(64) returns 'A' |
| date(s) | returns a date |
| date | returns a string<br>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 |

### 7.5.3 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()` | |

### 7.5.4 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)` | |

### 7.5.5 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.

#### 7.5.5.1 Connection

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

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

#### 7.5.5.2 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 |

#### 7.5.5.3 Statement

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

#### 7.5.5.4 Window

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

### 7.5.6 aclfgl_ Builtins

#### 7.5.6.1 Procedures

These do not return values. Prefix them with aclfgl_
e.g. call aclfgl_client_set()

| Procedure (prefixed: aclfgl_) |
|---|
| call_in_shared( lib, func) |
| client_execute( string) |
| client_set() |
| delete_file(filename) |
| dump_screen(string, h,w,y,x) |
| expand_env_vars_in_cmd_line() |
| flush_inp() |
| flush_ui() |
| sendfile_to_ui(filename) |
| send_to_ui(string) |
| set_color(c, r,g,b) |
| setenv(varname, value) |
| set_pdf_encoding(encoding) |

#### 7.5.6.2 Functions

These all return values. Prefix with aclfgl_
e.g. LET random=aclfgl_random(13)

| Function (prefixed: aclfgl_) | Returns |
|---|---|
| file_exists(filename) | true\|false |
| getclientfile() | various? |
| get_connection_username() | name |
| get_construct_element(tab,col,t,l) | value |
| get_curr_height() | h |
| get_curr_width() | w |
| getcwd() | dirname |
| get_pdf_encoding() | encoding |
| get_sql_requirement | string |
| get_stack_trace() | trace |
| get_user() | name |
| parse_csv(line) | several? |
| random(n) | random%n |
| read_pipe(command) | output |
| replace_in_string(str, repl, with) | string |
| replace_start(string, with, start) | string |
| split_on_delimiter( line) | lines |
| tea_string_decypher(code) | string |
| tea_string_encypher(string) | code |
| text_as_str(text) | string |
| trim_nl(string) | string |

get_error_details() returns

1. lineno INTEGER
2. ModuleName CHAR(255)
3. Message CHAR(255)
4. State INTEGER

See under Aubit4GL Extensions

## 7.6 Form Syntax

```
DATABASE
   {database|FORMONLY}[WITHOUT NULL 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.

### 7.6.1 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],
DYNAMIC SIZE=n
```

### 7.6.2 Aubit 4GL GUI Attributes

The following Widgets can be used in an Aubit4GL GUI form (runnable only under GUI or HL_GTK)

```
tag=FORMONLY.field,
    WIDGET={BUTTON|CHECK|COMBO|ENTRY
                |DEFAULTS|LABEL|PIXMAP|RADIO
                |TEXT} [CONFIG="guiattr='value' [;...]"]
```

Each widget may or must be given a set of GUI attributes:

| | |
|---|---|
| BUTTON | [CONFIG="LABEL='label'"\| "IMAGE='file.xpm'"] |
| CHECK | [CONFIG="LABEL='label'" ; "VALUE='value'"] |
| COMBO | {CONFIG="LIST=item1,item2[,...]} |
| ENTRY | [CONFIG="MAXCHARS=n"] |
| DEFAULT | [CONFIG="MAXCHARS=n"] |
| LABEL | {CONFIG="CAPTION='string'"} |
| PIXMAP | {CONFIG="IMAGE='file.xpm'"} |
| RADIO | {CONFIG="NUM=n; L1="label1";V1="value2" ; L2="label2"; V2=value2; ... Ln="labeln"; Vn=valuen} |
| TEXT | [CONFIG="MAXCHARS=n"] |
| any | [CONFIG="WIDTH=xchars;HEIGHT=ylines"] |

## 7.7 VDC Forms

Replace SCREEN with LAYOUT. Format is:
LAYOUT container [tag][(attributes) [ ... ] END

Containers can be any of HBOX, VBOX, FOLDER, PAGE, GRID, TABLE. and may be nested.

Example:

```
LAYOUT
VBOX Sales
GRID Order (BORDER)
{
   [o001 ]
    ...
}
TABLE Items (BORDER)
{
[i001 ] [1002] ... [i00n]
  ...
}
END
```

| VDC Widgets | |
|---|---|
| EDIT | Default - normal 1 line edit field |
| BUTTON | |
| BUTTONEDIT | Edit with a button |
| CHECKBOX | |
| COMBOBOX | ITEMS=(”Mr”, ”Mrs”, ”Ms”) |
| DATEDIT | Buttonedit + calendar |
| IMAGE | |
| LABEL | |
| PROGRESSBAR | |
| TEXTEDIT | Multiline Textfield |
| WEBVIEW | Browser Widget |

## 7.8 Callbacks

**viafunc()** takes 4 arguments: (tabname, colname, type, length) and returns TRUE or FALSE

**sortfunc()** takes 2 array arguments: (COPYOF ar1, COPYOF arr2) DEFINEd same as each row of the array to be sorted and returns -1 or 0 or +1 meaning less than, equal to, or greater than respectively.

**filterfunc()** takes a string argument and returns a row of column values (for inserting into a table)

# Chapter 8

# Help system

## 8.1 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
```

## 8.2 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` 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.

## 8.3 help in programs

### 8.3.1 Within 4GL

```
CALL showhelp(3)
```

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

### 8.3.2 At runtime

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

## 8.4 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.

## 8.5 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.

## 8.6 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 9

# 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.

## 9.1 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.

## 9.2 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.

## 9.3 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 A
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
```

## 9.4 Converting SQL

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/co` 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
```

## 9.5 Conversion Syntax

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

### 9.5.1 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

## 9.5.2 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)

## 9.5.3 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 10

# 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.

### 10.0.4 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.

## 10.1 Makefiles

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

#### 10.1.0.1 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)  fcompile $<
%.ao : %.4gl
(TAB)  4glc -c $?
%.iem : %.msg
(TAB)  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.

#### 10.1.0.2 Make glossary:

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

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

**$<** = the first of the newer prerequisites. 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 %)

#### 10.1.0.3 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.

### 10.1.1 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).

### 10.1.2 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.

### 10.1.3 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 be 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

### 10.1.4 .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.

### 10.1.5 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.

### 10.1.6 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.

## 10.1.7 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 11

# amake

## 11.1 Introduction

With the Aubit 4GL compiler, compiling a small program can be trivial:

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

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

```
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
- 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.

## 11.2 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.

## 11.3 Converting old makefiles

### 11.3.1 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:

### 11.3.2 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
```

### 11.3.3 amakeallo

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

### 11.3.4 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.

## 11.4 2. amake

Examples:

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

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

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

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

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

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

### 11.4.1 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

### 11.4.2 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.

### 11.4.3 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

### 11.4.4 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`

### 11.4.5 #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.

### 11.4.6 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.

#### 11.4.6.1 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.

### 11.4.7 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.

#### 11.4.7.1 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.SUFFIX
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).

#### 11.4.7.2 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

### 11.4.8 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 12

# A4GL Utilities

## 12.1 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
```

## 12.2 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 multiple places (eg. either Informix or Postgres) and hence may well have a different meaning.

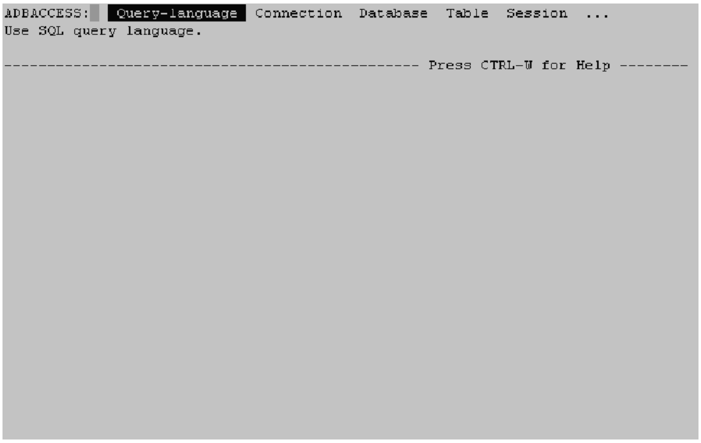
## 12.3 asql

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

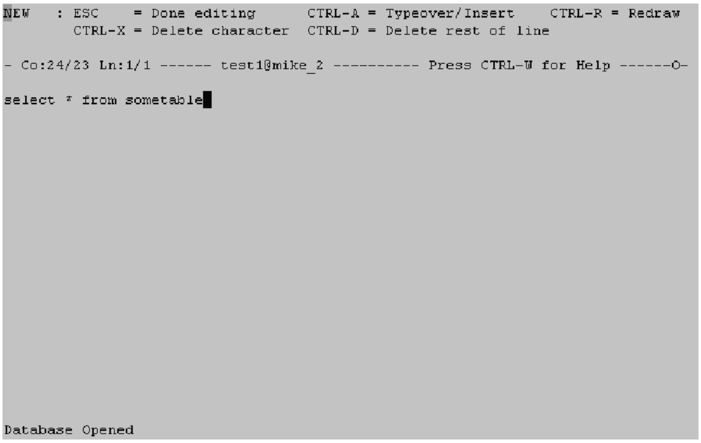
1. `asql_g.4ae` - Generic interface (For ODBC usage)
2. `asql_i.4ae` - Compiled using native Informix ESQL/C

3. `asql_p.4ae` - Compiled using native Postgres ecpg

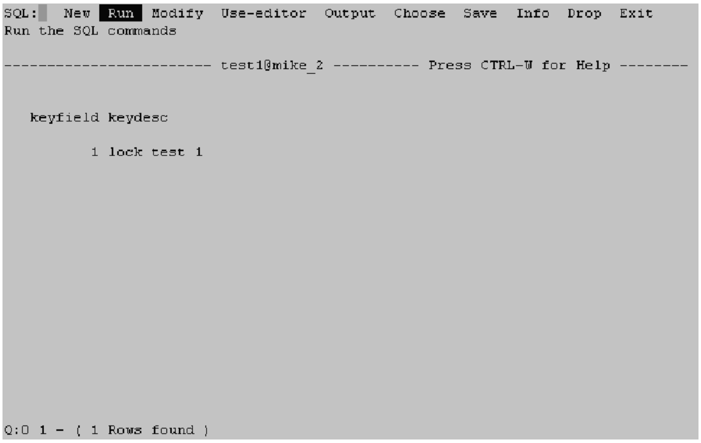
When the program starts - you'll be presented with a menu :



```
ADBACCESS:  Query-language  Connection  Database  Table  Session  ...
Use SQL query language.

------------------------------------------------ Press CTRL-W for Help --------
```

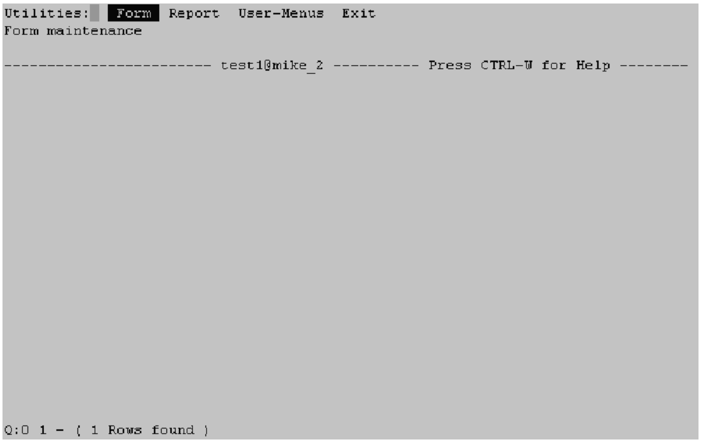


```
NEW   : ESC     = Done editing      CTRL-A = Typeover/Insert    CTRL-R = Redraw
        CTRL-X = Delete character  CTRL-D = Delete rest of line

- 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.



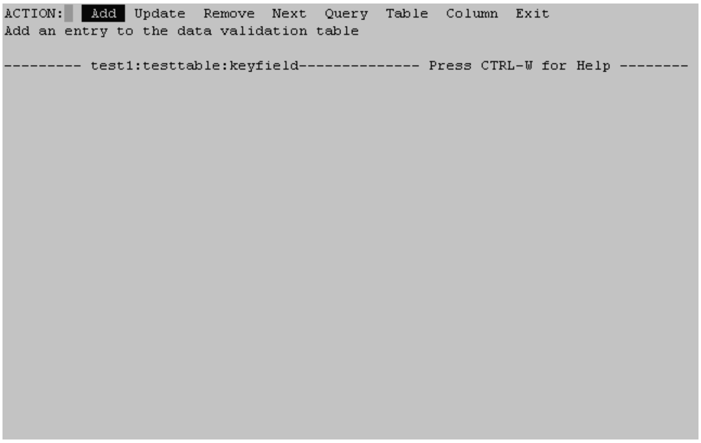
```
Utilities:   Form  Report  User-Menus  Exit
Form maintenance

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









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

## 12.3.1 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.

## 12.4 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.



```
ACTION:  Add  Update  Remove  Next  Query  Table  Column  Exit
Add an entry to the data validation table

--------- test1:testtable:keyfield-------------- Press CTRL-W for Help --------
```

## 12.5 P-Code Dropped

Aubit4GL used to include an experimental PCode compiler. Work on this has ceased because of proposed projects to develop a common bytecode for Java, Perl, and Python. When this is solidified, we intend to rewrite our Aubit4GL pcode compiler to output pcode conformant with the new standard.

| 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 |

## 12.6 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.

## 12.7 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.

## 12.8 default_frm

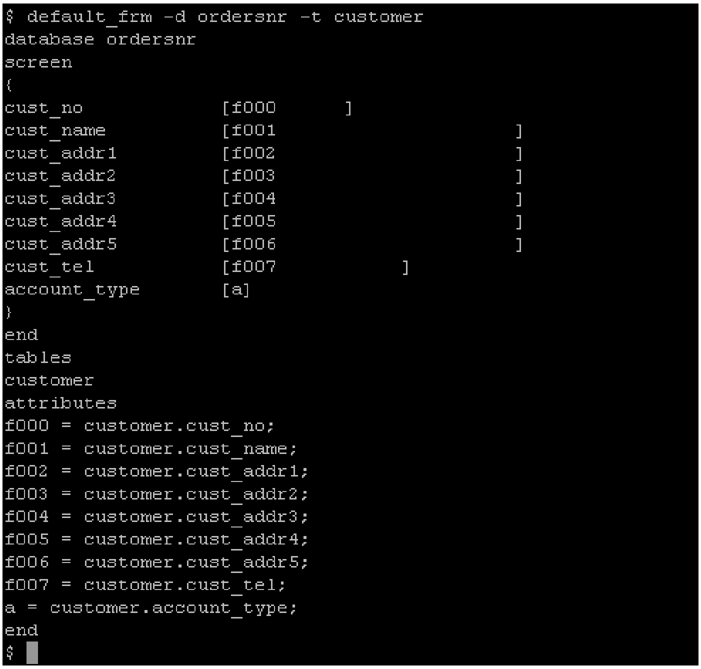
`default_frm` will generate a default form for a table(s) specified on the command line.

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
$
```

## 12.9 fcompile

This is the Aubit4GL tool to create a form binary from a .per form file.e.g.

Usage:

```
fcompile customer
Note: Omit the .per form suffix
```

will create a file with suffix .afr.

### 12.9.1 Builtin Forms

It is possible to build the compiled binary form into an Aubit4GL program. This allows you to distribute binaries without the need to bundle the .afr file with it. It also allows the file to run in an environment where the variables are incompatible with the environment where the forms were originally compiled.

The process to build in the form(s) is as follows:

- Set the environment:
  `A4GL_PACKER=PACKED`
  `A4GL_FORMTYPE=GENERIC`
- Use `fcompile -c <form>` to generate a .c file
- Inside the .4gl code, before displaying the form use the statement:
  `CALL form_is_compiled( form, "MEMPACKED", "GENERIC")`
- Compile the .c formfile with the program code. e.g.
  `4glpc progr.4gl form.afr.c -o prog.4ae`

You can (as of Version 1.2) create an xml version of a .per file by using the -xml option:

```
fcompile -xml customer
```

will create customer.per.xml

This option is intended for use with the new GUI Visual Display clients such as theVentas VDC program.

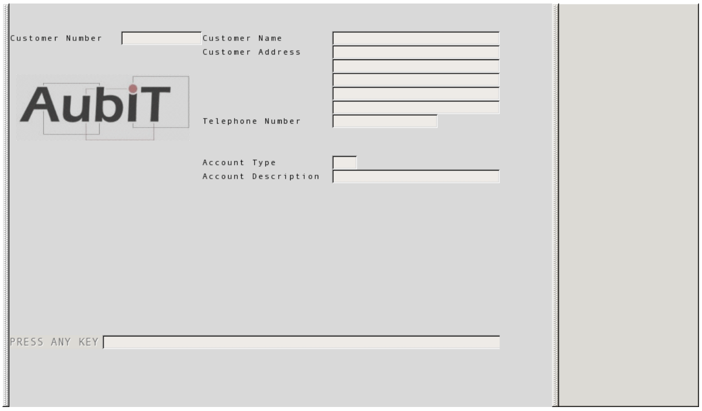
It is not strictly necessary to use the -xml option as Aubit4GL will convert a .afr file on the fly to xml for transmission to the XML front ends.

## 12.10 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



## 12.11 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.

## 12.12 mkpackage

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

## 12.13 prepmake

A utility script to convert makefiles to `amake` format

## 12.14 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

## 12.15 Internal Apps

### 12.15.1 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 13

# Packages

## 13.1 Packages

Packages are shared libraries that contain callable code.

Normally - you call these with code like :

```
call channel::open_file("fin","/etc/passwd","r")
call channel::open_file("fout","./passwd.new","w")
...
call channel::read(lv_in,[lv_rec.*]) returning b
```

You could import those functions so they can be called without the `channel::` prefix by creating a package and then importing it in the 4gl

```
IMPORT PACKAGE channel
```

or its synonym :

```
USE channel
```

This channel_package file would be just a text file normally in the directory:
`$AUBITDIR/etc/import`
which contains the library and function names:

```
channel open_file
channel open_pipe
channel set_delimiter
channel close
channel fgl_read
```

The channel_package file may also be in the current working directory or your home directory. (You can also specify a list of directories to search by using the `A4GL_CLASSPATH` environment variable.)

You can have functions from many libraries in a single package. That is why you need to have the library name as well as the function names.

To create these shared libraries:

```
$ 4glpc -as-dll module.4gl -o module.so
```

The standard AUBIT additional libraries are set in

`$AUBITDIR/lib/extra_libs/*/LIBRARYNAME.[4gl|so]`

These are NOT currently included in the standard binary downloads, but can be got from cvs:

```
cvs -d:pserver:anonymous@aubit4gl.cvs.sourceforge.net:/cvsroot/aubit4gl login

cvs -z3 -d:pserver:anonymous@aubit4gl.cvs.sourceforge.net:/cvsroot/aubit4gl \
co -P aubit4glsrc/lib/extra_libs
```

The 4GL files will then need to be compiled, either using

`$ 4glpc -as-dll module.4gl -o module.so`

or maybe by running the command: `make`

Local library modules may be placed in the current working directory or the HOME directory (assuming they are in LD_LIBRARY_PATH), or anywhere else in that PATH. The module name should be in the form:

`lib<NAME>.4gl`

The `lib` prefix should not be included in the channel_package file.

To use the channel library:

```
MAIN
  DEFINE lv_s char(200)
  CALL channel::open_pipe("pipename","ls -l","r")
  WHILE channel::read("pipename",lv_s)
CODE
    A4GL_trim_nl(lv_s);
ENDCODE
    DISPLAY lv_s
    END WHILE
  CALL channel::close("pipename")
END MAIN
```

Here it runs the command `ls -l` and reads the output. (You might want to open a pipe for writing, in which case the last parameter would be a "w", but you cannot open a pipe for both reading and writing).

`"pipename"` is just a name associated with the pipe and could be anything so long as it is unique within your program.

The string (`lv_s`) should be long enough to contain the string - and includes any trailing newlines. In the example we trimmed this by dropping to C and calling the Aubit4GL builtin: `A4GL_trim_nl()`

Note : the CODE and ENDCODE keywords must be the only things on their respective lines - no leading or trailing spaces or tabs.

But between the CODE and ENDCODE there can have as many spaces and newlines as you like!)

## 13.2 channel

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

### 13.2.1 Dependencies

None

### 13.2.2 Synopsis

```
DEFINE handle CHAR(64)
DEFINE filename CHAR(512)
DEFINE cmd CHAR(512)
DEFINE flag CHAR(1) # r, w, u, a
DEFINE delimiter CHAR(1) # e.g. '|'
USE channel
CALL open_file(handle,filename,flag)
CALL open_pipe(handle,cmd,flag)
CALL set_delimiter(handle,delimiter)
CALL close(handle)
CALL fgl_read(handle,n) # Obsolete
CALL read(handle,variable)
CALL read(handle,var1, var2, ... varn)
CALL write(handle,var)
CALL write(handle,var1,var2, ...varn])
```

Notes:

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)

## 13.3 file

This is a library exposing various STDIO functions from the standard C library. Handles are all standard 4GL INTEGER's. Mostly the Aubit4GL functions are wrappers around the standard C functions of the same name.

### 13.3.1 Dependencies

None

### 13.3.2 Synopsis

```
DEFINE filename CHAR(256)
DEFINE p_command CHAR(256) # e.g "ls -l"
DEFINE dirname CHAR(256)
DEFINE direntry CHAR(256)
DEFINE buff CHAR(256)
DEFINE mode CHAR(3)
DEFINE handle INTEGER
DEFINE r INTEGER # return value of C functions
DEFINE n INTEGER # count
DEFINE ok INTEGER # boolean
DEFINE c CHAR(1)
USE a4gl_file
LET r=popen(p_command,mode)
LET r=fopen(filename,mode)
LET r=ftell(handle)
LET r=ferror(handle)
LET r=fseek(handle,n)
LET r=fseek_from_end(handle,n)
LET n=fsize(handle)
LET buff=fgets(handle)
LET c=fgetc(handle)
LET r=fputs(buff, handle)
LET r=feof(handle)
LET r=fclose(handle)
CALL rewind(handle)
LET handle=opendir(dirname)
CALL readdir(handle)
    RETURNING direntry, ok
LET r=closedir(handle)
```

## 13.4 html

This library allows you to use Aubit4GL to create web pages. Originally contributed by Andrj Falout, it has been neglected for some time. You can find the source file: `libahtmllib.4gl` in the subdirectory: `tools/html` of the source distribution. Be warned that if you are looking for the source of individual functions, the htmlf_ and js_ prefixes appear in uppercase (HTMLF and JS_) while the html_ prefixes appear in lower case (html_). Routinely the 4GL compilers downshift all function names when creating C code.

Documentation of the html package is immature at this stage so be merciful if your esteemed editor errs in some details.

### 13.4.1 Dependencies

None

### 13.4.2 Synopsis

```
DEFINE HTTP_POST_VAR CHAR(1000)
DEFINE HTTP_GET_VAR CHAR(1000)
DEFINE www RECORD
    GATEWAY_INTERFACE   CHAR(20),
    SERVER_NAME         CHAR(60),
    SERVER_SOFTWARE     CHAR(100),
    SERVER_PROTOCOL     CHAR(20),
    REQUEST_METHOD      CHAR(10),
    QUERY_STRING        CHAR(300),
    DOCUMENT_ROOT       CHAR(200),
    HTTP_ACCEPT         CHAR(200),
    HTTP_ACCEPT_CHARSET CHAR(40),
    HTTP_ACCEPT_ENCODING CHAR(20),
    HTTP_ACCEPT_LANGUAGE CHAR(10),
    HTTP_CONNECTION  CHAR(20),
    HTTP_HOST           CHAR(200),
    HTTP_REFERER        CHAR(600),
    HTTP_USER_AGENT     CHAR(300),
    REMOTE_ADDR         CHAR(20),
    REMOTE_PORT         CHAR(10),
    SCRIPT_FILENAME     CHAR(300),
    SERVER_ADMIN        CHAR(100),
    SERVER_PORT         CHAR(10),
    SERVER_SIGNATURE    CHAR(100),
    PATH_TRANSLATED     CHAR(300),
    SCRIPT_NAME         CHAR(300),
    REQUEST_URI         CHAR(600)
END RECORD
DEFINE HTTPvar ARRAY [100] OF RECORD
        vname CHAR(20),
        value CHAR(60)
    END RECORD
DEFINE g_css CHAR(40)
DEFINE cssfilename CHAR(40)
DEFINE mytext CHAR(10000)
DEFINE pagetitle CHAR(300)
DEFINE string CHAR(512)
DEFINE level INTEGER
DEFINE url CHAR(200)
DEFINE w, h INTEGER
DEFINE linkurl CHAR(300)
DEFINE linktext CHAR(300)
DEFINE elem, elem1, elem2, ... CHAR(1000)
DEFINE target CHAR(10)
DEFINE varname CHAR(60)
#--------------------------------
# Simple HTML Functions
#--------------------------------
CALL html_css(cssfilename)
CALL html_display_para(mytext)
CALL html_end_body()
CALL html_end_center()
CALL html_end_para()
CALL html_head(pagetitle)
CALL html_headers()
CALL html_heading(string, level)
CALL html_hline()
CALL html_image(url, w, h)
CALL html_init(cssfilename)
CALL html_link(url, target, linktext)
CALL html_list(elem1, elem2, ...)
CALL html_meta(elem)
CALL html_params(params)
CALL html_redirect(url)
CALL html_start_body()
CALL html_start_center()
CALL html_start_para()
CALL html_title(pagetitle)
#---------------------
# HTML Form Functions
#---------------------
CALL htmlf_checkaspnews()
CALL htmlf_checksitenews()
CALL htmlf_countrylist()
CALL htmlf_email()
CALL htmlf_email_confirm()
CALL htmlf_firstname()
CALL htmlf_form_footer()
CALL htmlf_form_header()
CALL htmlf_lastname()
CALL htmlf_loginname()
CALL htmlf_radioitems()
CALL htmlf_submitbutton()
CALL htmlf_tableframeend()
CALL htmlf_tableframestart()
CALL htmlf_titleline()
CALL htmlf_transfervalue()
#-------------------------
# Javascript functions
#-------------------------
CALL js_endcode()
CALL js_launchhelpwin()
CALL js_launchnewwin()
CALL js_pageload()
CALL js_startcode()
CALL js_validateemail()
CALL js_validateform()
CALL js_windowname()
#-----------------------------
# Miscellaneous (mostly don't use them)
#-----------------------------
CALL show_webserver_vars()
CALL show_webserver_vars_page()
CALL vread(varname)
CALL vreadall()
CALL xxxgetstdin()#
CALL getstdin()
CALL get_webserver_vars()
```

The functions prefixed by html_, htmlf_, and js_ are intended for use by you the programmer. These functions all output to stdout HTML code (or Javascript in the case of the js_xxx functions) using the 4GL DISPLAY statement. The other functions are not intended for use by the programmer but are called from within the library.

In general, the functions with `start` in their names output the appropriate starting tags. The functions with `end` in their name output the closing tag.

e.g.

`html_start_para()` outputs: `<P>`

`html_end para()` outputs: `</P>`

The htmlf_ functions provide HTML structures such as tables, pick lists, etc for use in HTML Forms.

The easiest way to learn this library is to use it. Just write code importing the package, call the function, and look at its output.

### 13.4.3 Example

```
# myhtml.4gl
#   output a webpage
USE a4gl_html
MAIN
CALL html_head("John's Page")
CALL html_start_body()
CALL html_init("common") #load common.css
CALL html_heading("Example Page", 1)
CALL html_hline()
CALL html_start_center()
CALL html_start_para()
DISPLAY "This is an example of a centred paragraph"
DISPLAY "where the source file has two lines of text"
CALL html_end_para()
CALL html_end_center()
CALL htmlf_tableframestart()
CALL htmlf_form_header()
CALL htmlf_titleline()
CALL htmlf_radioitems()
DISPLAY "<\TR>"
DISPLAY "<TR>"
CALL htmlf_countrylist()
DISPLAY "<\TR>"
DISPLAY "<TR>"
CALL htmlf_firstname()
DISPLAY "<\TR>"
DISPLAY "<TR>"
CALL htmlf_lastname()
DISPLAY "<\TR>"
DISPLAY "<TR>"
CALL htmlf_loginname()
DISPLAY "<\TR>"
DISPLAY "<TR>"
CALL htmlf_email()
DISPLAY "<\TR>"
DISPLAY "<TR>"
CALL htmlf_email_confirm()
DISPLAY "<\TR>"
DISPLAY "<TR>"
CALL htmlf_submitbutton()
CALL htmlf_form_footer()
CALL htmlf_tableframeend()
CALL get_webserver_vars()
CALL show_webserver_vars()
CALL html_end_body()
END MAIN
```

Compile and run:

```
4glpc -o myhtml.4ge myhtml.4gl
./myhtml.4ge > myhtml.html
```

Point your browser at `myhtml.html`

## 13.5 memcached

This library allows access to memcached servers.

### 13.5.1 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

### 13.5.2 Synopsis

```
DEFINE host CHAR(20)
DEFINE port CHAR(20)
DEFINE key CHAR(255)
DEFINE value CHAR(255)
DEFINE statrec RECORD
  pid INTEGER,
  version CHAR(30),
  curr_items INTEGER,
  total_items iNTEGER,
  bytes INTEGER,
  curr_connections INTEGER,
  total_connections INTEGER,
  connection_structures INTEGER,
  cmd_get INTEGER,
  cmd_refresh INTEGER,
  cmd_set INTEGER,
  get_hits INTEGER,
  get_misses INTEGER,
  refresh_hits INTEGER,
  refresh_misses INTEGER,
  bytes_read INTEGER,
  bytes_written INTEGER,
  limit_maxbytes INTEGER
END RECORD
DEFINE r INTEGER # return value
DEFINE mc INTEGER
DEFINE val INTEGER
DEFINE bytes INTEGER
DEFINE ok INTEGER # boolean
DEFINE req INTEGER # boolean?
DEFINE res INTEGER
DEFINE ptr INTEGER
USE a4gl_memcache
LET mc=mc_new()
LET r=mc_server_add(mc,host,port)
LET r=mc_server_add4(lv_mc,lv_host)
CALL mc_add(mc, key, val, bytes)
CALL mc_add_str(mc, key, val)
LET ok=mc_replace(mc, key, value, bytes)
LET ok=mc_replace_str(mc, key, value)
LET r=mc_req_new()
LET r=mc_req_add(req, key)
CALL mc_get(mc, req)
LET value=mc_aget_str(mc,key)
CALL mc_aget_rec(mc,key,ptr,bytes)
CALL mc_set(mc, key, value, bytes)
CALL mc_set_str(mc, key, value)
CALL mc_res_free_on_delete(res, ok)
CALL mc_res_free(req, res)
CALL mc_stats(lv_mc) RETURNING statrec.*
CALL mc_delete(mc, key)
CALL mc_incr(mc, key, val)
CALL mc_decr(mc, key, val)
CALL mc_free(mc)
```

## 13.6 pcre

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

### 13.6.1 Dependancies

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

### 13.6.2 Synopsis

```
DEFINE re CHAR(1024) # a perl RE (regular expr)
DEFINE s CHAR(1024)
DEFINE buff CHAR(255)
DEFINE i INTEGER # 1 .. 30
DEFINE OK INTEGER # boolean
USE a4gl_pcre
LET buff=pcre_text(i)
LET OK=pcre_match(re,s)
```

pcre_text(i) returns the matched portion of the string (up to 30 portions are stored)

pcre_match(re,s) returns true if the string s matches the regular expression re.

e.g.

```
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
```

## 13.7 pop

This module allows you to download and delete email from a pop3 server. It is based on a small pop3 library created by Benoit Ruits. The original motivation for creating the Aubit4GL package was that a huge volume of spam emails was killing Spamassassin. So Mike Aubury wrote a 4GL program dealing with this. You can find it in the source in

AUBITDIR/lib/extra_lib/pop in the file: pop_killer.4gl

All the functions popxxx() are from the pop package and we could have called them using A4GL_pop::popxxx(). You should initiate a pop3 session with a call to popbegin() supplying the server, userid, and password. You close the session with a call to popend(). All the tedious details of setting up sockets and freeing them are taken care of by Benoit's libspop library. The Aubit4GL package also trims variables appropriately.

Beware however that there is no provision for accessing or downloading attachments to messages.

## 13.7.1 Dependancies

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

## 13.7.2 Synopsis

Most of the Aubit4GL popxxx() functions are wrappers around the libspop functions of the same name with the wrappers looking after details such as trimming any string args to functions etc.

```
DEFINE c20 CHAR(20)
DEFINE c256 CHAR(256)
DEFINE server CHAR(256)
DEFINE userid CHAR(64)
DEFINE password CHAR(64)
USE a4gl_pop
LET c20=popget(i, "From") # or "To", "Subject", "C", "Date", or
LET c256=poperr()
LET OK=popbegin(server, user, password)
LET n=popnum(i)
LET n=popbytes(i)
LET n=popmsgsize(i)
LET c256=popmsguid(i)
LET c256=popgetmsg(i)
LET c256=popgethead(i)
CALL popcancel()
CALL popend()
CALL popdelmsg(i)
```

A sample program to give you the flavour of the pop library:

```
MAIN
    DEFINE l_head CHAR(2048)
    DEFINE l_body CHAR(2048)
    DEFINE l_from CHAR(80)
    DEFINE l_subject CHAR(80)
    DEFINE i INTEGER
    DEFINE n INTEGER
    USE POP
    DISPLAY "Connecting to server...."
    IF popbegin("pop3.abc.com","john", "abc1234") then
        DISPLAY "OK: Connected"
    ELSE
        DISPLAY poperr()
        SLEEP 4
        EXIT 1
    END IF
    LET n = popnum()
    DISPLAY "You have", n using "####&"," Messages"
    FOR i = 1 to n
        LET l_head=popgethead(i)
        LET l_body=popgetmsg(i)
        LET l_from=popget(i, "From")
        LET l_subject=popget(i, "Subject")
        # do some processing .....
        # CALL popdelmsg(i)
    END FOR
    IF int_flag THEN
        CALL popcancel() #undo any popdelmsg(i)
    END IF
    CALL popend()
END MAIN
```

## 13.8 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.

### 13.8.1 Dependancies

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

### 13.8.2 Synopsis

```
DEFINE msg CHAR(200)
DEFINE server CHAR(200)
DEFINE addressee CHAR(512)
DEFINE mimetype CHAR(40)
DEFINE mimedesc CHAR(256)
DEFINE emailaddr CHAR(255)
DEFINE filename CHAR(255)
DEFINE hint CHAR(10)
DEFINE has_error INTEGER
DEFINE n INTEGER
DEFINE session INTEGER
DEFINE partno INTEGER
DEFINE partno2 INTEGER
DEFINE msgno INTEGER
DEFINE port INTEGER
DEFINE flags INTEGER
DEFINE ismine INTEGER
USE a4gl_smtp
CALL set_errmsg(msg)
CALL clear_err()
CALL set_server(server)
LET server=get_server() # defaults to $SMTP_SERVER or "mail"
LET msg=get_errmsg()
LET session=start_message(lv_sender,lv_subject)
CALL add_recipient(session, addressee)
LET partno2=mime_type_new(msgno,partno, mimetype)
LET partno2=mime_type_new_with_description(msgno,
           partno,mimetype,mimedesc)
LET partno
   =mime_type_new(msg,0,"multipart/mixed")
LET lv_textpart
   =mime_type_new(msgno,mixedpart,"text/plain")
IF filename MATCHES "*.pdf" or hint="PDF" THEN
   let lv_pdfpart
    =fgl_smtp::mime_type_new_with_description(lv_message,
           lv_mixedpart,"application/pdf",lv_rep_filename)
ELSE
  let lv_reppart
    =fgl_smtp::mime_type_new_with_description(
           msgno,mixedpart,"text/html",filename)
END IF
CALL connect(msg,server,port,flags,ismime)
CALL disconnect(msgno)
CALL send_to(msgno,emailaddr)
CALL send_to_cc(msgno,emailaddr)
CALL send_to_bcc(msgno,emailaddr)
CALL part_send_file_html_listing(msgno,lv_file,lv_last)
CALL part_send_file(msgno,lv_file,lv_last)
CALL send_report(hint,filename,emailaddr) # used by REPORT TO EMAIL
```

part_send_file() 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.

## 13.9 string

This module includes 3 string handling functions which may be useful from within a 4GL program.

### 13.9.1 Dependencies

None

### 13.9.2 Synopsis

```
DEFINE line CHAR(256)
DEFINE n INTEGER
DEFINE fields ARRAY[100] OF CHAR(256)
DEFINE haystack CHAR(512)
DEFINE needle CHAR(512)
DEFINE c CHAR(1)
USE a4gl_string
CALL split(line,n)
   RETURNING fields[1], fields[2], ... fields[n]
LET n=strstr(haystack,needle)
LET n=strchr(haystack,c)
```

split() splits a string into up to 100 space separated fields

strstr() finds the first location of a string within a string (or 0)

strchr() finds the first location of a character within a string

## 13.10 sxml

The sxml package is a an Aubit4GL wrapper round a set library functions from Fabrizio Bruno's Freshmeat project SXML which facilitates the creation and reading of the limited set of XML commonly used in software configuration files.

The Aubit4GL functions are mostly wrappers around functions of the same name in the SXML library.

The SXML functions use C language pointers to XML structs. Pointers do not exist in 4GL so they get coerced to INTEGER in the Aubit4GL wrappers. The variables named xmlptr, xmlptr2, etc are INTEGERS serving as pointers to the SXML tree structures.

### 13.10.1 Dependencies

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

### 13.10.2 Synopsis

```
DEFINE xmlptr, xmlptr2 INTEGER
DEFINE key CHAR(255)
DEFINE name CHAR(255)
DEFINE errbuff CHAR(255)
DEFINE value CHAR(1024)
DEFINE txt CHAR(32000)
USE a4gl_sxml
CALL sxml_free_tree(xmlptr)
LET xmlptr=sxml_readfile(fname) # Read file into an XML tree
LET ok=sxml_writefile(fname, xmlptr)
LET xmlptr2=sxml_get_next(xmlptr, key, n)
LET xmlptr2=sxml_get_sub(xmlptr, key, n)
LET value=sxml_get_value(xmlptr, key, n)
LET txt=sxml_get_xml_as_text(head, xmlptr)
LET xmlptr=sxml_put_next(xmlptr, name)
LET xmlptr=sxml_put_sub(xmlptr, name)
CALL sxml_put_value(xmlptr, value)
LET errbuf=sxml_curr_errstr()
LET errbuf=sxml_errstr(n)
LET n=sxml_get_errno()
```

The functions get_next and put_next operate on the nth node matching key at the same level as the xmlptr.

The functions get_sub and put_sub operate on the nth node next matching key at the first sublevel of the xmlptr.

## 13.11 dynamic

### 13.11.1 Dependencies

None

### 13.11.2 Function list

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

# Chapter 14

# 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.

## 14.1 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
```

## 14.2 Associative Arrays

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.

Example:

```
    DEFINE lv_desc  ASSOCIATE [CHAR](1) WITH ARRAY[40] of CHAR(40)
    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!)

## 14.3 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.

## 14.4 Slices

Aubit4GL allows you to use an INPUT ARRAY or a DISPLAY ARRAY statement on a slice of an array. Sample syntax:

```
DISPLAY ARRAY items SLICE ( itemno THROUGH ext )
INPUT ARRAY items SLICE ( itemno THROUGH ext )
```

For header-detail type forms, the Aubit4GL SLICE allows you to use an existing form array and run the DISPLAY and/or INPUT ARRAY statements on a contiguous subset of the fields (typically excluding the foreign key fields) whereas standard 4GL would require you to declare a separate array in both the form and in the 4GL program code.

## 14.5 TODO statement

Mike Aubury has introduced a new statement TODO ... END TODO similar in structure to a CASE ... END CASE but which iterates until all its WHEN branches are DONE.

```
TODO  [condition]
   WHEN condition
     statements ...
   IF cond THEN
     DONE
   END IF
   ...
   [ ALWAYS statements ... ]
END TODO
```

The idea is that each WHEN must be marked DONE before the loop exits. There is an optional conditional at the top of the loop which can also force a premature exit, along with EXIT TODO and/or CONTINUE TODO. The idea is to have a list of workflow and batch type operations - which can be performed in any order - but all must be performed before we can proceed. Each item is only executed when the WHEN condition is met - but the

DONEs must still be executed (ie. at some point - the WHEN condition must evaluate to TRUE and the commands executed, resulting in a DONE)

There is also an ALWAYS item which shouldn't/can't be marked DONE but is executed each time around the loop (maybe for a SLEEP, or a DISPLAY etc.)

Eventually, Mike might look to see if each of these WHENs can get executed on a separate thread - so it would be useful for writing parallel executing code.

## 14.6 ODBC Data access

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

## 14.7 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.

## 14.8 Constants

Aubit4GL allows you to define a CONSTANT in one place (for example to define an array size) then refer to it throughout all the modules in a program. To increase the array, simply edit the DEFINE CONSTANT statement and recompile. In standard 4GL, you have to find every instance of the value of the constant and change it before recompiling with the risk of having missed some.

Syntax:

```
    DEFINE CONSTANT pi 3.1415923
    DEFINE CONSTANT codemax 256
```

## 14.9 Callback Functions

Aubit4GL has three extensions to the classic 4GL grammar which allow the programmer to write callback functions:

- CONSTRUCT ... VIA viafunc
- SORT arrayvar USING sortfunc [LIMIT n]
- LOAD FROM file USING FILTER filterfunc
  INSERT INTO tabname [(col, ...)]

In the above statements viafunc, sortfunc, and filterfunc are callback functions - i.e programmer supplied functions which the Aubit4GL program will call to perform lower level operations.

The callback functions allow you to modify at low level the behaviour of the CONSTRUCT, SORT, and LOAD statements.

### 14.9.1 CONSTRUCT VIA

Aubit4GL now allows you to write code like this:

```
CONSTRUCT lv_str ON a,b
  FROM a,b
  VIA viafunc
```

The VIA clause is an Aubit4GL extension.

The `viafunc` is a user supplied function which takes 5 parameters and returns the required SQL boolean expression.

The parameters which Aubit4GL's CONSTRUCT statement will pass to the callback function are:

1. Table Name: char(18)
2. Column Name: char(18)
3. String: char(300)

4. Type: integer
5. Length: integer

You use these however you wish to generate an appropriate SQL boolean expression which will be PREPAREd and EXECUTED (usually in a FOREACH loop).

In standard CONSTRUCT statements, Aubit4GL builds the boolean for each field using its builtin function:
`aclfgl_get_construct_element()`
which takes the same 5 parameters as above.

#### 14.9.1.1 VIA Example

```
MAIN
  DEFINE a RECORD
    a CHAR(10),
    b CHAR(20)
    END RECORD
  DEFINE lv_str CHAR(200)
  OPEN WINDOW w1 AT 1,1 WITH FORM "f1"
# Here - we want to use a callback function in the
# construct to allow us to amend the CONSTRUCT string
# before the construct returns it.
#
# This can be used (for example) to automatically add
# '*' around the string, or to search an address
# (by doing something like
# '( addr1 MATCHES .. OR addr2 MATCHES ...)' etc etc
#
  CONSTRUCT lv_str ON a,b from a,b VIA viafunc
  DISPLAY lv_str
  CLOSE WINDOW w1
END MAIN
FUNCTION viafunc(lv_tabname, lv_colname,
      lv_string,lv_dtype, lv_dtypelength)
  DEFINE lv_tabname CHAR(18)
  DEFINE lv_colname CHAR(18)
  DEFINE lv_string CHAR(300)
  DEFINE lv_dtype,lv_dtypelength INTEGER
# Normally - we'd want to generate the construct portion
# - but with maybe different column names
# In this callback we want to map 'a' to be 'blah'...
#
  IF lv_colname="a" THEN
# aclfgl_get_construct_element takes 5 parameters
# tablename, column name, search string,
# datatype and datatype length
# (eg for decimal or character length)
   LET lv_string=aclfgl_get_construct_element(
     lv_tabname , "blah", lv_string ,
     lv_dtype, lv_dtypelength)
  ELSE
   LET lv_string=aclfgl_get_construct_element(
     lv_tabname , lv_colname , lv_string ,
     lv_dtype, lv_dtypelength)
  END IF
  RETURN lv_string
END FUNCTION
```

This gives total flexibility, because we need to return the string for each field. Normally we can just call the default aubit4gl function to generate that string (`aclfgl_get_construct_element()`), but we dont need to use that, or we can call it with different parameters from what the CONSTRUCT would use.

In the following example, the programmer wants to select data from a different column depending on whether the length of the vin value is more than 8 chars or not (assuming the column is called vin on the form) :

```
FUNCTION viafunc(lv_tabname, lv_colname,
  lv_string,lv_dtype, lv_dtypelength)
  DEFINE lv_tabname CHAR(18)
  DEFINE lv_colname CHAR(18)
  DEFINE lv_string CHAR(300)
  DEFINE lv_dtype,lv_dtypelength INTEGER
  IF lv_colname="vin" THEN
    IF length(lv_string)=8 THEN
```

```
          LET lv_colname="vin8"
        END IF
        IF length(lv_length)>8 THEN
          LET lv_colname="vinunq"
        END IF
      END IF
      RETURN aclfgl_get_construct_element(
         lv_tabname , lv_colname, lv_string ,
         lv_dtype, lv_dtypelength)
    END FUNCTION
```

### 14.9.2 SORT ... USING sortfunc

In classical 4GL, sorting arrays is clumsy, partly because you cannot pass arrays to and from functions, and partly because there is no builtin sort function. Aubit4GL improves on this with 3 extensions:

- `SORT arrayvar USING sortfunc` [LIMIT n] statement which allows the programmer to sort an array relying on the callback function to apply the appropriate ordering of successive rows in the array. If you suppy the LIMIT n clause, the sort will apply only to the first n rows.
- `COPYOF` operator which can bulk-copy (using the C library function `memcpy()`) an array or record to a function argument
- `COPYBACK` operator which reverses the action of COPYOF by bulk-copying back to its source variable

**sortfunc()** is a user supplied function which takes 2 identical COPYOF arguments:
`sortfunc(COPYOF arg1, COPYOF arg2)`
with arg1 and arg2 DEFINEd as a record to match each row of the array to be sorted and returning -1 or 0 or +1 meaning less than, equal to, or greater than respectively. Aubit4GL will pass rows, 2 at a time, to the function and rely on the return value to rank them.

**COPYOF** is an operator for function arguments which tells the Aubit4GL compiler to use the C-library function `memcpy()` to bulk-copy a large mass of data in one operation to the 4GL function stack. In standard

4GL all the elements of a record are pushed individually onto the stack and then pulled off individually. This not efficient in a data and memory intensive operation such as sorting where there will be very many such calls to the callback function. The use of the COPYOF operator allows the fastest possible function call and return. Another benefit of COPYOF (not used in the context of SORT ... USING callback) is that this operator takes the curse off the passing of arrays to a function (an operation which results in a compile-time error in standard 4GL).

**COPYBACK** is the reciprocal of COPYOF and can be used to return large local data structures (e.g. records or arrays) using the C-library function `memcpy()`. The syntax is:
COPYBACK varname
which will overwrite the original source variable with the current data in varname local to the callback function. (Aubit4GL remembers where it copied it from at function invocation and performs `memcpy()` with the original arguments reversed.)

#### 14.9.2.1 Example code

This example shows how to use the Aubit4GL statement: SORT ... USING sortfunc

```
DATABASE test1
DEFINE lv_array ARRAY[100] OF RECORD LIKE systables.*
FUNCTION qsort_tabname(COPYOF lv_sys1, COPYOF lv_sys2)
  DEFINE lv_sys1,lv_sys2 RECORD LIKE systables.*
  IF lv_sys1.tabname>lv_sys2.tabname THEN RETURN 1 END IF
  IF lv_sys1.tabname=lv_sys2.tabname THEN RETURN 0 END IF
  RETURN -1
END FUNCTION
MAIN
  DEFINE lv_cnt integer
  DEFINE lv_a integer
  DECLARE c1 CURSOR FOR
        SELECT * FROM systables WHERE tabid<99
  LET lv_cnt=1
  FOREACH c1 INTO lv_array[lv_cnt].*
```

```
            LET lv_cnt=lv_cnt+1
      END FOREACH
      FOR lv_a=1 TO 100
         IF lv_array[lv_a].tabname IS NOT NULL THEN
            DISPLAY "BEFORE:",lv_array[lv_a].tabname CLIPPED
         END IF
      END FOR
      SORT lv_array USING qsort_tabname
      FOR lv_a=1 TO 100
        IF lv_array[lv_a].tabname IS NOT NULL THEN
           DISPLAY "AFTER :",lv_array[lv_a].tabname CLIPPEd
        END IF
      END FOR
    END MAIN
```

#### 14.9.2.2 Example 2

This example shows how you can (in Aubit4GL) use COPYOF to pass arrays to and from functions saving a lot of time if passing identical large structures. Omit the COPYOF in the example, and the 4GL compiler will produce a compile time error.

```
    DEFINE cnt INTEGER
    DEFINE a ARRAY[20] OF RECORD
            b CHAR(20),
            c INTEGER
    END RECORD
    MAIN
      DEFINE b ARRAY[20] OF RECORD
            b CHAR(20),
            c INTEGER
        END RECORD
      FOR cnt=1 TO 20
            LET a[cnt].b=cnt
            LET a[cnt].c=cnt
      END FOR
      CALL bibble(COPYOF(a))
      FOR cnt=1 TO 20
```

```
          DISPLAY ">", a[cnt].b," ",  a[cnt].c," <"
    END FOR
  END MAIN
  FUNCTION bibble(COPYOF lv_a)
    DEFINE lv_a array[20] OF RECORD
          b CHAR(20),
          c INTEGER
      END RECORD
    FOR cnt=1 TO 20
      DISPLAY ">", a[cnt].b," ",
        a[cnt].c," <", lv_a[cnt].b," ",
        lv_a[cnt].c
      LET lv_a[cnt].b=99-cnt
    END FOR
    COPYBACK lv_a
  END FUNCTION
```

Notice that both the CALL and the FUNCTION definition need the COPYOF operator applied to their arguments.

Here lv_a is a local copy of the array a. Instead of passing in 40 separate parameters (two for each element in the array), we just pass in a single block of memory. Internally Aubit4GL uses the C function `memcpy()` to copy this over the lv_a when the function is called, so it is a byte for byte copy.

(The sizes of the COPYOF parameter must match - or an error is flagged up)

Now, the normal behaviour (if the COPYBACK is commented out) would be that, at the end of the function, the changes made to lv_a would be lost. But as it was a COPYOF parameter, we can copy those details back to the original array a using the COPYBACK command(which basically does a `memcpy()` the other way).

### 14.9.3 LOAD ... USING FILTER fname ...

Aubit4GL has added an optional `USING FILTER filterfunc` clause to the 4GL LOAD command.

Syntax:

```
LOAD FROM filename
   USING FILTER filterfunc
   INSERT into tabname[(col, ...)]
```

The filter function needs to return the data to insert (one value per column) and can return 0 values, in which case the line is ignored and no insert performed.

There is a builtin CSV parser filter function available:
`aclfgl_parse_csv()`
which you can use to load a CSV file. e.g. :

```
LOAD FROM myfile.csv
  USING FILTER aclfgl_parse_csv
  INSERT INTO mytab
```

The callback function takes a string which is the entire line from the load file and returns the individual columns to insert. Returning no values means that that line in the file will be skipped. This can be very useful for data validation (only load certain rows) or ensuring foreign keys exist etc. ( or possibly inserting blank ones).

There is a new builtin function
`aclfgl_split_on_delimiter()`
which can be used to split the fields. e.g. :

```
CALL aclfgl_split_on_delimiter ("A|B|C|")
  RETURNING lv_rec.*
CALL aclfgl_split_on_delimiter ("#","A#B#C#")
  RETURNING lv_rec.*
```

Obviously, this adds possibilities over and above inserting data. For example, you could LOAD the /etc/passwd with a callback which calls the above split function but returns nothing so that it won't try to do any inserts but you will have captured the values in an array of 4GL RECORDs.

There is a variant of the LOAD command which takes a variable holding the INSERT statement:

```
LOAD FROM file USING FILTER filtfunc varname
```

The variable varname would contain something like:

```
INSERT into mytab
```

## 14.10 Error Hooks

At the instigation of Andrej Falout, Mike Aubury has altered error handling in Aubit4GL.

Now when an error occurs (other than one which causes a core dump or is the result of EXIT PROGRAM), the program calls a function errlog() in a shared (i.e. dynamically linked) library.

To customise error handling you do the following:

- write a function `errlog()` as per example below in a file say `myerr.4gl`
- compile myerr.4gl  
  `4glpc --as-dll myerr.4gl`
- put the compiled myerr.so somewhere sensible  
  where it will be found because of LD_LIBRARY_PATH or ldconfig
- export A4GL_ERRHOOK=myerr  
  Note: No need for .so or .dll suffix

There is a default implementation of the `errlog()` function in the standard Aubit4GL libraries.

### 14.10.1 A4GL_ERRHOOK

You can write your own shared library and set an environment variable A4GL_ERRHOOK to use your library implementation of errlog() instead of the default.

You don't need to link anything with your application.

### 14.10.2 errlog()

The function errlog() takes 4 parameters

1. Line Number: integer
2. Module Name: char(64)
3. Error Number: integer
4. Error Message: char(1024)

It does not return a value.

### 14.10.3 Example

Mike Aubury has created an illustrative example in `lib/extra_libs/errhook` in the aubit4gl source directory

You'll find there a makefile, a README, and a 4gl module: sample.4gl

sample.4gl simply DISPLAYs all the information you could think of. but it would be easy to modify it to output via a report and send the result via email etc.

Heres an example output (We have wrapped some lines for this book):

```
********************************************
* ERRHOOK
********************************************
* User : aubit4gl
* Time : 2007-12-13 17:49:27
* Module : menuprog.4gl
* Line : 70
* Error : -3002
* PID : 11577
* lastkey : 2016
********************************************
* ScrDump : /tmp/scr.out.err_11577
********************************************
```

```
Customer Maintenance: Query Next Previous FireErr Add
  Update Delete Exit
[ ]
Customer Number [ 2] Customer Name [ABC CHEMICALS PLC ]
Customer Address [Water Way. ]
[Runcorn ]
[Cheshire ]
[WA7 9LN ]
[ ]
Telephone Number [01925 849313 ]
Account Type [2 ]
Account Description [Credit: 60 days ]
6 Record(s) have been found
********************************************
* Errmsg :
* Program ./menuprog stopped at 'menuprog.4gl',
   line number 70.
* Error status number -3002.
* Wrong number of parameters passed to function.
*
* 4gl function call stack :
* menuprog.4gl (Line 8) calls MAIN
********************************************
```

To compile and use Mike's sample.4gl:

```
cd lib/extra_libs/errhook
make
export CALLERRHOOK=errhook_sample
```

### 14.10.4 sample.4gl

```
function errlog(ln,mod,err,errmsg)
  define ln integer
  define mod char(64)
  define err integer
  define errmsg char(1024)
  define lv_cline char(2000)
  define lv_continue integer
  define a integer
  define lv_fname_scrdump char(300)
  define lv_l text
  let lv_fname_scrdump="/tmp/scr.out.err_",
     fgl_getpid() using "<<<<<<"
  call aclfgl_dump_screen(lv_fname_scrdump)
  locate lv_l in file lv_fname_scrdump
  display "********************************************"
  display "*                 ERRHOOK                   "
  display "********************************************"
  display "* User    : ", aclfgl_get_user()
  display "* Time    : ", current year to second
  display "* Module  : ",mod clipped
  display "* Line    : ",ln
  display "* Error   : ",err
  display "* PID     : ",fgl_getpid()
  display "* lastkey : ",fgl_lastkey()
  display "********************************************"
  display "* ScrDump : ",lv_fname_scrdump clipped
  display "********************************************"
  display lv_l
  display "********************************************"
  display "* Errmsg :"
# The error message contains embedded \n's which mess up
# the display..
# here we're just searching for them
# and using them as delimiters
  let lv_continue=1
  while lv_continue
let lv_continue=0
for a=1 to length(errmsg)
if ord(errmsg[a])=13 then
let errmsg[a]=' '
end if
if ord(errmsg[a])=10 then
if a>1 then
display "* ", errmsg[1,a-1]
```

```
else
display "*"
end if
let errmsg=errmsg[a+1,1024]
let lv_continue=1
exit for
end if
end for
  end while
  if length(errmsg) then
display "* ", errmsg clipped
  end if
  display "*******************************************"
end function
```

## 14.11 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.

## 14.12 New Types

Aubit4GL allows you to define a new type once then refer to that type whenever you need to create a variable or parameter of that type. e.g.

```
DEFINE NEW TYPE staff
   RECORD
      staffno INTEGER,
      surname CHAR(20),
      firstname CHAR(20),
      addr1 CHAR(20),
      ....
   END RECORD
DEFINE person staff
LET person.staffno = 205
LET ....
...
OUTPUT REPORT stafflist( staff )
```

## 14.13 Variable IDs

You can specify and reference all 4GL objects in the.

`_variable (str)` is used to replace hard coded identifiers. e.g for PREPARE statements or CURSOR names or WINDOW names

So

```
OPEN WINDOW w1 at at 1,1 WITH 10 ROWS, 10 COLUMNS
```

could be written as

```
LET lv_str="w1"
OPEN WINDOW _variable(lv_str) at at 1,1 WITH 10 ROWS, 10 COLUMNS
```

Might seem a little unnecessary at first - but what if you wanted to open 20 windows ?

## 14.14 Passing IDs

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

## 14.15 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.

When you use a `FUNCTION fname()` statement, Aubit4GL applies a prefix `aclfgl_` to your function name thereby renaming it
`aclfgl_fname()`
in the .c file which the compiler generates. Aubit4GL does this to avoid inadvertent clashes with the names of C-library standard functions.

If, for example, you wrote a function:

```
FUNCTION printf()
...
END FUNCTION
```

and Aubit4GL did not mangle the function name, your code would not be able any longer to access (within CODE ... END CODE) the standard C-library function `printf()`

You can change this by setting A4GL_NAMESPACE to a different prefix or eliminate it entirely:

```
$ export A4GL_NAMESPACE=""
$ 4glpc mod.4gl
```

You will need to make sure that all your modules are compiled like this, or you'll have the same trouble calling functions across 4GL modules!

Don't forget that Aubit4GL lets you embed blocks of C code which may also help you. e.g. :

```
MAIN
DEFINE a CHAR(5)
LET a="World"
CODE
{
  printf("Hello %s",a);
}
ENDCODE
END MAIN
```

This saves all the trouble of having to push/pop from the 4gl stack.

## 14.16 MOVE WINDOW

```
MOVE WINDOW w1 TO row, col
SHOW WINDOW w1
HIDE WINDOW w1
```

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

## 14.17 WHENEVER

```
WHENEVER SUCCESS statements
WHENEVER SQLSUCCESS statements
```

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

## 14.18 Multilevel Menus

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

## 14.19 Extended DISPLAY

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.

## 14.20 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.

## 14.21 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.

## 14.22 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.

## 14.23 a4gl_get_info()

An example :

```
DATABASE loadz
MAIN
  DEFINE lv_cnt INTEGER
  DEFINE lv_a INTEGER
  DEFINE lv_name CHAR(18)
  DEFINE lv_type,lv_length,lv_scale INTEGER
  PREPARE p1 from "select * from customer2"
  CALL a4gl_get_info("Statement","info_p1","NoColumns")
    RETURNING lv_cnt
  DISPLAY lv_cnt , " columns"
  FOR lv_a=1 to lv_cnt
    CALL a4gl_get_info("Statement","info_p1",
                  "Name"||lv_a using "<<<<<")
      RETURNING lv_name
    CALL a4gl_get_info("Statement","info_p1",
                  "Type"||lv_a using "<<<<<")
      RETURNING lv_type
    CALL a4gl_get_info("Statement","info_p1",
                    "Length"||lv_a using "<<<<<")
      RETURNING lv_length
    CALL a4gl_get_info("Statement","info_p1",
                    "Scale"||lv_a using "<<<<<")
      RETURNING lv_scale
    DISPLAY lv_name, " ",lv_type ," ",lv_length, " ", lv_scale
  END FOR
END MAIN
```

## 14.24 get_error_details()

An example:

```
MAIN
   WHENEVER ERROR CALL bibble
   OPEN c1 # not declared - so it should error...
END MAIN
FUNCTION bibble()
   DEFINE lv_line INTEGER
   DEFINE lv_mod CHAR(255)
   DEFINE lv_msg CHAR(255)
   DEFINE lv_stat INTEGER
   CALL aclfgl_get_error_details()
     RETURNING lv_line,lv_mod,lv_stat,lv_msg
   DISPLAY "There was an error : "
   DISPLAY "in module: ", lv_mod CLIPPED," line ",lv_line
   DISPLAY "status = ",lv_stat," sqlca.sqlcode=", sqlca.sqlcode
   DISPLAY "message : ",lv_msg CLIPPED
END FUNCTION
```

## 14.25 Dynamic Form Fields

Dynamic form fields are enabled by setting the attributes as follows:

`f001=tabname.fieldname, dynamic size=N ;`

where N is the maximum size to be allowed.

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.

## 14.26 Remote Functions

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....

## 14.27 LINKED TO

Aubit4GL allow us to link a record to a table and record it's primary key. The statement:

```
DEFINE r LINKED TO tab PRIMARY KEY(col1)
```

effectively DEFINEs r as a RECORD LIKE tab and tells the compiler that col1 is its primary key. Having done that we can set the values of the record's primary key field to a value and use the following Aubit4GL statements:

```
INSERT USING r
SELECT USING r
UPDATE USING r
DELETE USING r
```

An example program:

```
DATABASE test1
DEFINE lv_systables LINKED TO systables primary key (tabid)
MAIN
   LET lv_systables.tabid=1
```

```
        SELECT USING lv_systables
        DISPLAY "Hello World : ",lv_systables.tabname
    END MAIN
```

The compiler generates the appropriate WHERE x=y automatically. e.g.:

```
    SELECT USING lv_systables
```

becomes under the bonnet:

```
    SELECT * INTO lv_systables.* FROM systables
    WHERE systables.tabid=lv_systables.tabid
```

You have saved yourself a lot of typing!

## 14.28 ON ANY KEY etc

Aubit4GL has extended the INPUT ARRAY and DISPLAY ARRAY statements to react to events other than the standard ON KEY (key)

Examples:

ON ANY KEY statements

ON IDLE 5 MINUTES statements

ON INTERVAL 10 SECONDS statements

ON CHANGE(field) statements

## 14.29 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

## 14.30 SCHEMA v DATABASE

```
SCHEMA dbname -- (for compile time)
MAIN
  DEFINE l_dbname CHAR(64)
  DATABASE l_dbname -- (for runtime)
  ...
END MAIN
```

When you use the DATABASE statement at the top of a program, 4GL will connect you to the database when it enters the MAIN function. If however you replace the usual DATABASE keyword with SCHEMA at the head of the file containing the MAIN function, Aubit4GL will not connect to the database on beginning execution of MAIN - you will need to explicitly place a DATABASE statement in MAIN or use CONNECT or SESSION statements

The SCHEMA statement (a 4Js extension implemented by Aubit4GL for compatibility with 4J) will allow you to compile against a local database intending to run against a different database at runtime (perhaps on a different server).

You can also use a schema file instead of a database connection at compile time if you do the following:

Run the command:

```
adbschema -d dbname -fileschema > dbname.schema
export A4GL_SQLTYPE=FILESCHEMA
```

Now, in your program, use the usual DATABASE statement at the top of the file:

```
DATABASE dbname
MAIN
   ...
END MAIN
```

The format of the dbname.schema file is :

```
[table1]
column datatype size
column datatype size
column datatype size
[table2]
column datatype size
column datatype size
column datatype size
etc...
```

## 14.31 SESSIONS

Use the SESSION statements to access different databases within the same program.

(One limitation is that you cannot link across databases in a single query.)

Example:

```
MAIN
  DEFINE lv_a CHAR(10)
  OPEN SESSION s_id1 TO DATABASE test1
  OPEN SESSION s_id2 TO DATABASE test2
  USE SESSION s_id1 FOR SELECT USER INTO lv_a
    FROM systables WHERE tabid=1
  DISPLAY lv_a
  USE SESSION s_id2 FOR SELECT USER INTO lv_a
    FROM systables WHERE tabid=1
  DISPLAY lv_a
  SET SESSION s_id1
  ...
END MAIN
```

## 14.32 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.

## 14.33 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.

## 14.34 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 IDs and the ability to pass IDs to functions as parameters. This functionality alone can save significant time in the coding process, and allow you to isolate ID related problems easily.

## 14.35 A4GL Wizard

### 14.35.1 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.

## 14.36 PDF Reports

Built using PDFlib, allows you to produce reports in PDF format with fancy fonts. In particular, PDF Reports are useful for producing barcodes. See the separate chapter on PDF reports.

## 14.37 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.)

This facility will not be developed any further. We advise you to use the new Display Clients which are documented elsewhere in this manual

## 14.38 Packages

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

```
CALL library::function()
or
CALL library.function()
```

## 14.39 a4gl IDE

### 14.39.1 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.

## 14.40 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 15

# ACE reports

Informix has a set of report utilities which generate, compile, and run ACE reports. The default ACE report looks like this:

```
DATABASE mydb END
SELECT
    col1,
    col2,
    ....
FROM tab1 END
FORMAT EVERY ROW END
```

Informix source files have the suffix: .ace. The Informix program `aceprep` compiles source into a binary with suffix .arc. Another program `acego` runs the binary to produce output.

If you have legacy ACE reports, you may use the Aubit4GL utilities to create 4GL source which you may then compile and run. Aubit4GL provides a set of utilities:

- generate_aace to create default ACE reports
- aace workalike for aceprep
- aace_runner workalike for acego

- aace_4gl translates .aarc binary to 4GL source
- aace_perl translates .ace or .aace file to perl (needs report.pm)

### 15.0.1 generate_aace

`generate_aace` will produce default Informix-style ACE report for a given database and table. The generated file should be given a `.aace` or a .ace suffix added to the table name.

```
generate_aace -d mydb -t agents > agents.aace
```

## 15.1 aace

`aace` will translate a .aace file into a `.aarc` binary file

```
aace agents
```

will create a binary file: `agents.aarc` from `agents.[a]ace`

## 15.2 aace_4gl

`aace_4gl` translates a `.aarc` file and outputs a `.4gl` file on stdout. You can compile this with `4glpc` to make an executable.

```
aace_4gl agent[.aarc] > agent.4gl
```

ACE reports and 4GL reports have traditionally behaved differently in their handling of aggregates. 4GL `sum()` and `group sum()` return the counts or rows processed so far, while ACE `total of` and `group total of` return the aggregate of all data. Mike Aubury has added a Compatibility option to force the ACE behaviour.

The new compatibility options are: -C, -I, -B

### 15.2.1 -C Compatibility

Mike Aubury has added a -C option to aace_4gl which generates code which should match the normal ACE behaviour, at the expense of populating and querying a temporary table...

It should autodetect if it can ignore the compatibility mode if it is not required, so you should be safe to -C the aace_4gl anyway..

### 15.2.2 -I Insert Cursor

There is also another option -I which uses an INSERT cursor to speed up the report (only required if you use -C).

### 15.2.3 -B Batch Size

you can also specify a batch size with -B..

So for some examples :

```
aace_4gl -C m1.aarc -o m1.4gl
```

or

```
aace_4gl -C -I m1.aarc -o m1.4gl
```

or

```
aace_4gl -C -I -B 100 m1.aarc -o m1.4gl
```

# Chapter 16

# New Display Clients

## 16.1 New GUI Front Ends

New with version 1.2 of Aubit4GL is support via an XML protocol for separate Graphical front end applications to provide display services for Aubit4GL programs in the spirit of X display servers.

### 16.1.1 History

#### 16.1.1.1 TUI

The Aubit4GL project began with support for the traditional ASCII Informix form files displaying on an 80x24, 80x25, or 128x24 character screen. This is still the default for Aubit4GL but it can be enforced by setting A4GL_UI=TUI.

#### 16.1.1.2 GTK

Then support was added for rendering the same forms in a graphical user interface using the GNOME Tool Kit (GTK) library. This rendered the standard Informix menus and form fields with graphical equivalents from the GTK library.

#### 16.1.1.3 HL_TUI, HL_GTK

Later the Aubit4GL syntax was extended to give specific support for graphical widgets (BUTTON, COMBO, LABEL, PIXMAP, TEXT, etc). In order to support further expansion of graphical capabilities, Mike Aubury embarked on splitting the interface code into High Level code (with a prefix HL e.g. HL_TUI) with common calling functions. The idea was that the possible graphical interfaces of the future would share the same High Level interface and implement the functionality with calls to the various libraries in C++, C#, and Java through their C interfaces.

With version 1.2 onwards this approach has lapsed. For the non-graphical text user interface, use TUI (not HL_TUI) by setting A4GL_UI=TUI.

#### 16.1.1.4 Graphical Front Ends

For graphical user interfaces, you can still use GTK by setting
A4GL_UI=HL_GTK,
but there will not be much future development of the GTK GUI. Its place is being taken by a new generation of graphical front ends being built by commercial enterprises which are part of the Aubit4GL fraternity. These draw their inspiration from 4Js applications and emulate much of their style and syntax.

At the time of writing early July 2010, the most advanced of these is Ventas Display Client (VDC) from
`www.ventas.de`.

There are also a C# application and a Java application in the wings all using the same XML protocols to interface with 4GL programs.

Aubit4GL now has a XML formats (DTDs) for

- Forms. The command `fcompile -xml <formname>` will produce an XML file understood by the new generation of graphical front ends. Even if you haven't compiled your .per file to XML, Aubit4GL will convert your .afr file to xml on the fly provided you have set A4GL_UI=XML.
- Client -> 4GL. An XML protocol for sending user input from the front end client to a 4GL program (remote or local)

- 4GL -> Client. An XML protocol for sending data and commands from the Aubit4GL program to a graphical front end.

## 16.2 VDC

VDC (Ventas Display Client) is a rich thin client application which provides a Graphical User Interface (GUI) to Aubit4GL programs.

Ventas have initiated the development of a front end graphic user interface to display both Informix 4GL traditional forms and a newly devised, more graphics-oriented XML format which displays standard GUI objects such as pull-down menus, buttons, popup lists, arrayed vertically, horizontally, or in tabbed pages.

The VDC frontend is written in C++ and uses the QT4 libraries to implement the graphics. QT from Trolltech is available in Linux, Apple OSX, and Microsoft so that the VDC front ends are able to be on platforms separate from the server running the Aubit4GL programs.

## 16.3 Requirements

Assuming that you have installed both the Ventas Display Client and Aubit4GL either on the same or different machines, they will communicate with each other using TCP/IP with an agreed port (default: xxxx) and using an agreed XML format.

To run the VDC, you will need to run the back end 4GL program with the following environment variables set :

A4GL_UI=XML

AFGLSERVER=ip.address.of.client

and if you choose to operated via a proxy:

PIPEDIR=/path/to/dir

BASEPROGRAMS=/path/to/dir

The forms used by both front end and back ends need to be in XML format (which requires the environment variables):

## 16.4 Ventas Display Client

### 16.4.1 Linux

Note : On Linux we recommend compiling from source if possible.

#### 16.4.1.1 Source

Make sure you have the QT development libraries installed and get the source from SVN:

```
$ svn co https://aubit4gl.svn.sourceforge.net\
/svnroot/aubit4gl/trunk/remote_ui remote_ui
$ cd remote_ui/QT
$ qmake
$ make
```

#### 16.4.1.2 Binary

Don't do it!

### 16.4.2 Windows

#### 16.4.2.1 Source

You will need QT installed - make sure QT downloads the Mingw compiler also if you do not have that already installed.

Start a DOS command window from the QT command prompt (Start->All Programs->QT open source->QT command prompt)

Obtain the current code from the svn repository :

```
svn co https://aubit4gl.svn.sourceforge.net/\
  svnroot/aubit4gl/trunk/remote_ui/QT QT
```

or grab the latest client tarball.

`cd` to the top of the tree (should contain the client.pro file) and execute `qmake`. This is a program provided by the QT development tools which creates a makefile.

After `qmake` finishes - execute `make`, and you should end up with `client.exe`

#### 16.4.2.2 Binary

The current binary should be available from the front page at aubit.com.

Install this to `c:\aubit`. DBPATH should be set to include `c:\aubit` so that the Client can pick up any image files it needs.

These images should be in `c:\aubit\pics` or `c:\aubit\images` (or you add further directories to DBPATH for your own images).

## 16.5 Proxy

The proxy is an application server program which listens for connection requests from the Client, starts the required 4gl program, and then maintains communications back to the front end client.

The proxy allows the use of a Client behind firewalls and on Dynamic IP addresses.

In order to use the proxy - you need to set up 2 environment variables :

**PIPEDIR** This points to a directory where the proxy will store some temporary *named pipe* files.

**BASEPROGRAMS** This points to a directory where the 4gl applications exist. This is used to ensure that users cannot execute arbitrary programs - and can only use those which you specify..

When a Client connects to the proxy – there is a brief handshake – which involves :

```
PROTOCOL? → UIVERSION 1.0
PROGRAMNAME? → <program to execute>
USER? → <username>
PASSWORD? → <password>
```

#### 16.5.0.3 Authentication

The username/password combination must exist in the password file. The password file ($A4GL_PROXYPASSWD) is read and must contain username:password combinations.

The passwords will be encrypted with a simple password encryption mechanism – so passwords are not stored in a very secure way, but can at least not be read in plain text. Encrypted passwords begin with an '!', this means that a normal password cannot begin with a '!'.

Note: All processes run as the same user as the Proxy. The username which was passed in will be set as the environment variable PROXYUSER

The proxy program is called proxy. If it is not in your PATH, you may have to compile it from the source tar ball. Look for it in the subdirectory:
`lib/libui/ui_xml`

## 16.6 Form Layouts

### 16.6.1 SCREEN

The new GUI interfaces accept the normal TUI form layout and render the form elements using graphical widgets.

### 16.6.2 LAYOUT

To exploit the graphical capabilities of the new GUI front ends, replace the SCREEN keyword with LAYOUT GRID or LAYOUT TABLE.

GRID and TABLE have the same syntax as the TUI .per files in that you draw the screen within curly braces, and you provide tags within field delimiters which will link user input to program variables. The TABLE layout is intended for screen arrays. (You still have to declare the screen array in the INSTRUCTIONS section of the .per file.)

In addition to the mandatory GRID or TABLE container, you can exploit other containers:

- HBOX ( pack horizontally )
- VBOX ( pack vertically)
- GROUP (?)
- FOLDER (array of PAGEs tabbed)
- PAGE (a PAGE contained in a FOLDER)

These containers can be nested to influence the position of the graphical elements in the display.

#### 16.6.2.1 GRID

To emulate the traditional SCREEN section, do something like this:

```
LAYOUT
GRID
{ Label1 [f001]
  Label2 [f002]
  ....
}
END
```

#### 16.6.2.2 TABLE

To display the traditional SCREEN ARRAY, do something like this:

```
LAYOUT
TABLE
{
   Column1   Column2
  [f001   ] [f002   ]
  [f001   ] [f002   ]
   ....
}
END
```

#### 16.6.2.3 HBOX, VBOX

These containers respectively pack their contents horizontally and vertically. For example:

```
LAYOUT
VBOX Sales
GRID Order (BORDER)
{
   [o001]
   ....
}
TABLE Items (BORDER)
{
[i001] [i002] ... [i00n]
[i001] [i002] ... [i00n]
 ....
}
END
END
```

Will lay out the GRID above the TABLE. HBOX would lay out the TABLE alongside the GRID.

The contents of HBOX and VBOX can be any number of layout containers which in their turn contain containers in nested fashion.

#### 16.6.2.4 FOLDER

The Folder container is special and can only contain 1 or more PAGEs. e.g.

```
LAYOUT
FOLDER
  PAGE P1 .... END
  PAGE P2 .... END
  ...
END
END
```

The PAGEs are expected to contain GRID or TABLE containers with the usual tag assignments.

An reasonably full example:

```
database formonly
LAYOUT
FOLDER
PAGE Class
GRID
{
[f1      ]
}
END
END
PAGE Codes
TABLE
{
    [f001 ] [f002    ]
    [f001 ] [f002    ]
    [f001 ] [f002    ]
    [f001 ] [f002    ]
    [f001 ] [f002    ]
    [f001 ] [f002    ]
    [f001 ] [f002    ]
}
END
END
end
tables
formonly
attributes
TEXTEDIT f1 = formonly.class;
EDIT f001 = formonly.code;
EDIT f002 = formonly.descr;
INSTRUCTIONS
  SCREEN RECORD s_codes[7](code, descr);
end
```

#### 16.6.2.5 Container Syntax

The containers above all have the syntax:

```
type id (attributes) container END
```

The ID is optional and is a non-quoted string e.g. Sales

The Attributes are a optional parenthesised list (comma or space separated) consisting of any or all of the following:

```
STYLE="somestyle"
HIDDEN
AUTOSIZE
TEXT="some text"
ACTION=string
BORDER
```

e.g.

```
(AUTOSIZE BORDER)
```

### 16.6.3 Field Widgets

The graphical elements that you can put into the GRID or TABLE are the following:

- EDIT ( Normal One Line Edit Field ) This is the default.
- BUTTON
- BUTTONEDIT ( Edit with a Button (with Icon) on the right side)
- COMBOBOX (Dropdown LineEdit)
- CHECKBOX
- DATEEDIT (ButtonEdit with a CalendarWidget when button is clicked)

- IMAGE
- LABEL
- PROGRESSBAR
- TEXTEDIT ( Multiline TextField )
- WEBVIEW ( Browser Widget )

#### 16.6.3.1 Widget Syntax

To assign a widget type to a field in the .per file, precede the usual entry in the ATTRIBUTES section with a widget type. e.g.

```
f001=formonly.startdate, type=date;
```

could become

```
DATEEDIT f001=formonly.startdate, type=date;
```

The default type of widget is EDIT.

Each widget can have a TITLE attribute:

```
EDIT f003=formonly.firstname, TITLE="First Name";
```

The front end program will position the title appropriately according to its layout policy and you should not include the title in your layout of the GRID or TABLE section of the .per file.

To supply the options for a COMBOBOX, use the ITEMS attribute.

```
COMBOBOX f005=formonly.title,
   ITEMS=("Mr", "Mrs", "Ms", "Dr", "Sir", "Lady");
```

## 16.7 Settings/Environment Variables

The client executes in a Client/Server mode with the 4gl application. This separation is achieved using a TCP/IP connection.

When using the proxy, all the required settings are made by the proxy program.

If you want to start the 4gl application and have the client operate in Listen mode then you need to set these settings manually.

In order for a 4GL application to talk to the client, you need to tell Aubit4GL where the client is that it needs to connect to, by using the Clients IP address.

Setting A4GL_UI=XML makes Aubit4GL use the XML protocol required for communicating with the Client program.

You therefore need to set :

```
$ export AFGLSERVER=ip.addr.of.client
```

(You may also need to set A4GLPORT - although this default to the same values within the Client and the XML UI module.)

```
$ export A4GL_UI=XML
```

### 16.7.1 Debugging

If you `export LOGPROXY=Y`, Aubit4GL will write to the following files:

- logproxy.out
- logproxy.in
- proxy.log

You can set the level of verbosity to a value between 1 and 9 with:

```
export PROXYDBGLVL=Y
```

These files are all valuable in debugging the behaviour of the Aubit4GL proxy server and its communication between the back end front end.

## 16.8 Special functions

The following functions are useful in communicating with the new GUI display clients:

```
aclfgl_client_ui_call
opendir
openfile
savefile
aclfgl_add_to_toolbar
aclfgl_set_window_title
aclfgl_set_application_xml
aclfgl_dump_screen
aclfgl_set_display_field_delimiters
```

Other example calls:

```
DEFINE akt_window ui.Window
LET akt_window = ui.Window.getCurrent()
DEFINE akt_form ui.Form
LET akt_form = ui.Form.getForm()
CALL akt_form.setFieldHidden("fieldname", bool hidden)
CALL akt_form.setElementHidden("elementname", bool hidden)
CALL ui.interface.refresh()
```

**fgl_drawbox()** Has no effect in this UI

**aclfgl_send_to_ui()** Not normally required - used internally for sending Client UI specific commands

**aclfgl_client_set()** Makes settings within the ClientUI. Settings will be dependant on which UI is being used.

## 16.9 Ventas GUI Client

Check the Ventas website for definitive documentation when it becomes available.

### 16.9.1 Startup

Assuming that you have `cd`ed to the directory where you have downloaded the Ventas client, you can start the Ventas Display Client (VDC) with either of the following commands:

```
./Client -l
./Client <shortcut name>
```

## 16.10 Look & Feel

You can modify the default controls, styles, menus, and toolbars by constructing appropriate XML files and transmitting them to the display client using `sendfile_to_ui()`.

VDC expects these files to have the suffixes:

- `.4st` for style files
- `.4sm` for start menus
- `.4tb` for toolbars
- `.4ad` for actions

There are some examples below to help you.

### 16.10.1 Toolbars

The Toolbar is is sent via a XML-File (extension .4tb) to the Client by the sendfile_to_ui command()

Its structure is:

```
<ToolBar buttonTextHidden="0/1" iconHidden="0/1"
   hidden="0/1" buttonTextPosition="beside/under">
<ToolBarItem name="actionname" image="image.ext"
   text="Text" comment="Tooltip" />
<ToolBarSeparator/>
</ToolBar>
```

### 16.10.2 Images

Images have to reside inside a subdirectory `pics` or `images` or must be in the DBPATH/pics or DBPATH/images

### 16.10.3 Application Launcher

The StartMenu is is sent via a XML-File (extension .4sm) to the Client by the sendfile_to_ui command

Its structure is:

```
<StartMenu text ="RootElementText">
<StartMenu text ="RootElementText">
<StartMenuGroup text ="RootNodeText">
<StartMenuCommand text ="ChildNodeText"
      exec ="Command to Execute" />
</StartMenuGroup>
</StartMenu>
```

You can define different Menustyles for the Menu, at the moment there is a "tree-menu" or a menubar.

The Style can be defined with the Style attribute in 4gl or within the styles xml file with the startMenuPosition parameter

#### 16.10.3.1 STYLES

Styles are sent via a XML-File (extension .4st) to the Client by the sendfile_to_ui command

Its structure is:

```
<StyleList>
<Style name="sytlename" >
<StyleAttribute name="attributename"
   value="attributevalue" />
</Style>
</StyleList>
```

The Stylename can be one of the following:

- * (matches everything)
- a style name (should be referenced as style attribute in 4gl)
- an elementtype (e.g. ButtonEdit)
- stylename.elementtype
- (stylename.)elementtype:modifier (e.g. ButtonEdit:focus)

The Stylelist can contain many different styles.

Colour values can be defined in CSS or RGB values

#### 16.10.3.2 Attributes

| Attribute | Value | Description |
|---|---|---|
| startMenuPosition | menu \| tree | Display startmenu as Tree or Filemenu |
| ringMenuPosition | none \| top \| bottom \| left \| right | Position of the Ringmenu |
| actionPanelPosition | none \| top \| bottom \| left \| right | Position of the actionpanel |
| toolBarPosition | none \| top \| bottom \| left \| right | Position of the toolbar |
| color | RGB or CSS | Value Color of text |
| background-color | RGB or CSS | Value Backround color |
| hideButtons | yes \| no | Hide Option hides or just disable menubuttons |
| windowType | normal \| modal | Sets the windowtype to normal or modal |

#### 16.10.3.3 Modifiers

| Modifier | Description |
|---|---|
| :active | This state is set when the widget is in an active window |
| :enabled | This state is set when the widget is enabled |
| :disabled | This state is set when the widget is disabled |
| :focus | The item has input focus |
| :hover | the mouse is hovering over the item |
| :read-only | The item is read-only |

#### 16.10.3.4 Actions

Actions are sent via an XML File (extension .4ad) to the Client by the sendfile_to_ui command

```
<ActionDefaultList>
<ActionDefault name="actionname" attribute="attributevalue" />
</ActionDefaultList>
```

With the Actions file you can define default attributes for interactive Actions like Menubuttons or ON KEY or ON ACTION events or the Toolbar.

You can define more than one attribute for an Action but the name attribute is mandatory.

#### 16.10.3.5 Attributes

| Attribute | Value | Description |
|---|---|---|
| name | String | defines the actions name |
| text | String | if a button is displayed this is the buttons text |
| comment | String | this defines the tooltip |
| image | String | name of the image that is to be shown |
| defaultView | yes/no/auto | show button in the action panel or not |
| acceleratorName | String | 1st shortcut name |
| acceleratorName2 | String | 2nd shortcut name |
| acceleratorName3 | String | 3rd shortcut name |
| acceleratorName4 | String | 4th shortcut name |
| acceleratorName5 | String | 5th shortcut name |

The Acceleratornames can be

- a-z
- 0-9
- F1-F35
- Control-Shift

### 16.10.4 SSH client mode

Details to be supplied

## 16.11 Other GUI clients

Other specialised GUI client will be developed – UI's written in C# and Java are in development – please contact mike.aubury@aubit.com for further details.

## 16.12 Protocol

The New Graphical UIs are based on an XML protocol. The protocol is optimised so that it should run well even on high latency lines because data is only sent when required.

This does mean that there is very limited error checking (eg. for displaying to fields which dont exist on the form etc.) as many DISPLAY statements may be sent in a single *envelope*, which may well be sent some significant time after the DISPLAY command is executed.

Normally - this output is flushed when user input is required (eg - when doing a MENU or INPUT), or when an internal buffer becomes full.

If you are doing some processing which does not require user input - but does require the output to be displayed quickly then you can flush this internal buffer using aclfgl_flush_ui. e.g.

```
CALL aclfgl_flush_ui()
```

The protocol is open – in so far as the *on the wire* protocol is well defined and it should be easy to add your own custom front ends. You will just need to react to the datapackets defining the current UI statements and return the appropriate TRIGGERED package.

(See below for the DTD for the XML protocols used in both directions. XSD definitions are available from the aubit.com website)

### 16.12.1 Testing

The communications are normally conducted on STDIN/STDOUT – so you can try the protocol just by starting the program from the command line with the correct environment variables set :

E.g. For a simple 'hello world' program :

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

$ export A4GL_UI=XML

$ unset AFGLSERVER

$ ./myprog

```
<ENVELOPE ID="1">
<COMMANDS>
<PROGRAMSTARTUP PROGRAMNAME="./x1" ID="0">
<ENV NAME="DBPATH"
  VALUE=":/home/aubit4gl/aubit4glsrc/tools/">
<ENV NAME="DBDATE" VALUE="dmy4/"/>
</PROGRAMSTARTUP>
</COMMANDS>
</ENVELOPE>
<ENVELOPE ID="1">
<COMMANDS>
<DISPLAY>Hello World</DISPLAY>
<PROGRAMSTOP EXITCODE="0" ID="0">
</PROGRAMSTOP>
</COMMANDS>
</ENVELOPE>
```

Here is another example – this one uses a PROMPT to get some input from a user (reply from the client is in indentded and in bold):

```
<ENVELOPE ID="3">
<COMMANDS>
<PROGRAMSTARTUP PROGRAMNAME="./x1" ID="3">
<ENV NAME="DBPATH"
  VALUE=":/home/aubit4gl/aubit4glsrc/tools"/>
<ENV NAME="DBDATE" VALUE="dmy4/"/>
</PROGRAMSTARTUP>
</COMMANDS>
</ENVELOPE>
<ENVELOPE ID="3">
<COMMANDS>
<PROMPT CONTEXT="0" PROMPTATTRIBUTE="-1" FIELDATTRIBUTE="-1"
  TEXT="Please enter your name" CHARMODE="0" HELPNO="0"
  ATTRIB_STYLE="" ATTRIB_TEXT="">
<EVENTS>
</EVENTS>
</PROMPT>
<WAITFOREVENT CONTEXT="0" />
</COMMANDS>
</ENVELOPE>
<TRIGGERED ID="ACCEPT" LASTKEY="ACCEPT">
<SYNCVALUES><SYNCVALUE>Mike Aubury</SYNCVALUE></SYNCVALUES>
</TRIGGERED>
<ENVELOPE ID="3">
<COMMANDS>
<FREE TYPE="PROMPT" CONTEXT="0"/>
<DISPLAY>Mike Aubury </DISPLAY>
<PROGRAMSTOP EXITCODE="0" ID="3">
</PROGRAMSTOP>
</COMMANDS>
</ENVELOPE>
```

The front end simply replies with the TRIGGERED packet to send data back to the 4GL program.

All interactions are handled within a UI Context. Each UI statement which requires user input establishes a Context which is used by a subsequent WAITFOREVENT. When that Context is no longer required – the context is FREEd.

### 16.12.2 DTDs

For those developing other front ends for Aubit4GL there are DTDs and the equivalent XSD files available in a separate document and on the Aubit website. These define the formats for the packets which go from Client to Program, Program to Client, and the XML version of `.per` forms.

The XML DTDs and XSDs are too voluminous to include in this manual.

# Chapter 17

# PDF Reports

## 17.1 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).

## 17.2 Introduction

PDF report syntax is 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(...)
```

## 17.3 Output Section

The output section of a 4GL PDF report is slightly different from 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.pdf"
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).

### 17.3.1 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"
```

### 17.3.2 Report Structure

The report structure will be identical to that of a normal 4GL report.

In addition to the normal 4GL report statements, you can use the following Aubit4GL PDFREPORT extensions:

```
SET BARCODE TYPE expr
PRINT BARCODE [NO TEXT] [AT x, y] [WIDTH w HEIGHT h]
FILL
FILL STROKE
LINE TO [TOP] y, x
MOVE TO [TOP] y, x
SET COLOR r, g, b #all 0-1
SET FILL COLOR r, g, b
SET FONT NAME name
SET FONT SIZE n
SET PARAMETER name, value
SET STROKE COLOR r, g, b
STROKE
```

BARCODE expr can be one of 2, 5, 8, 13, 25, 39, 128A, 128B, 128C, or QR (all quoted)

### 17.3.3 Extras

In order to generate nice reports, there are a couple of extra features available.

#### 17.3.3.1 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
```

#### 17.3.3.2 Images

It is also possible to include an image within the PDF report, this is done in either of 2 ways:

- 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", "AS TIFF", "AS PNG", "AS JPEG".
- using the PRINT IMAGE statement with a filename. Supply the absolute path to the file.

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
print image "/local/images/oglion.png" as png
```

### 17.3.4 Example program

Please see pdf_report.4gl in test/

## 17.4 Barcodes

The following description has been supplied by Michael Krauss (mike@holzamer.de):

The supported barcodes in PDFREPORT are:

- ean-8 An EAN-8 is a barcode and is derived from the longer European Article Number (EAN-13) code. It was introduced for use on small packages where an EAN-13 barcode would be too large; for example on cigarettes
- ean-13 or UPC-A is an EAN-13 barcode (originally "European Article Number", but now renamed ”International Article Number” even though the abbreviation has been retained) is a 13 digit (12 + check digit) barcoding standard which is a superset of the original 12-digit Universal Product Code (UPC) system developed in the United States.[1] The EAN-13 barcode is defined by the standards organisation GS1.
  - addon ean-2 is a supplement to the EAN-13 and UPC-A barcodes. It is often used on magazines and periodicals to indicate an issue number.
  - addon ean-5 is a supplement to the EAN-13 and UPC-A barcode used on books. It is used to give a suggestion for the price of the book.

- barcode 39 (also known as "USS Code 39", "Code 3/9", "Code 3 of 9", "USD-3", "Alpha39", "Type 39") is a barcode symbology that can encode uppercase letters (A through Z), digits (0 through 9) and a handful of special characters like the $ sign.
- barcode 25 or barcode image *, code 25 *, standard 2of5 *, industrial 2of5, space code25, barcode space, bar code symbology, also known as standard 2 of 5, or industrial 2 of 5, is designed to encode numeric-only data.
- barcode QR is a matrix barcode (or two-dimensional code), readable by QR scanners, mobiles phones with camera, and smartphones.

The PDFREPORT statement to set the barcode type is:

```
SET BARCODE TYPE expr
```

where expr is one of:

```
"2", "5", "8", "13", "25", "39", "QR"
"128A", "128B", or "128C"
```

but the system is tolerant of variations such as:

```
"ean8", "ean-8", "ean13", "ean-13"
```

or upper case variants of these.

The following demonstrates how how to create barcodes of types: 2, 5, and 8

```
#   PDFREPORT
#
# Example for barcode ean-13
#    with addon ean-2 and ean-5
# and use of ean-8
#
# Contributed by Michael Krauss
# mike@holzamer.de
DEFINE i INTEGER
MAIN
    START REPORT pgtrex to "example.pdf"
    FOR i = 1 TO 20
       OUTPUT TO REPORT pgtrex(i)
END FOR
    FINISH REPORT PGTREX
    RUN "acroread ./example.pdf"
END MAIN
PDFREPORT pgtrex(r)
    DEFINE r INTEGER
OUTPUT
    TOP MARGIN 0 INCHES
    LEFT MARGIN 0.5 INCHES
    RIGHT MARGIN 0.1 INCHES
    BOTTOM MARGIN 0.2 INCHES
    PAPER SIZE IS a4
FORMAT
PAGE HEADER
    SET TEXT FORMAT "utf8"
    SET FONT NAME "Times-Roman"
    SET FONT SIZE 8
    SET BARCODE TYPE "13"
    SKIP TO 25 mm
    PRINT COLUMN 0.5 INCHES;
    PRINT BARCODE "4023500760409"
        WIDTH 1.8 INCHES  HEIGHT  0.8 INCHES
    SET BARCODE TYPE "5"
    SKIP TO 25 mm
    PRINT COLUMN 2.4 INCHES;
    PRINT BARCODE "08000"
         WIDTH 0.6 INCHES  HEIGHT 0.7 INCHES
    SKIP TO 60 mm
    PRINT "The barcode type '13' is one carton of Marlboro",
        " with 10 packages ",
        "and the price (barcode type '5') of €80.00."
    SKIP 1 LINE
    PRINT "Example of Tabacco-Branch:"
```

```
        PRINT "digit 1 to 3: '402' is the branch number."
        PRINT "digit 4 to 7: '3500' is the company number."
        PRINT "digit 8 to 12: '76040' is the article number."
        PRINT "digit 13; checksum"
        PRINT "But the use of digits and their definition ",
            "is from branch to branch and country different."
        SET BARCODE TYPE "8"
        SKIP TO  95 mm
        PRINT COLUMN 0.5 inches;
        PRINT BARCODE "40235462"
             WIDTH 1.2 inches HEIGHT  0.7 inches
        SKIP TO 125 mm
        PRINT "The barcode type '8' is one package of Marlboro."
        SKIP 1 LINE
        PRINT "Example:"
        PRINT "digit 1 to 4: '4023' is the company number."
        PRINT "digit 5 to 7: '546' is the article number."
        PRINT "digit 8: checksum"
        SET BARCODE TYPE "13"
        SKIP TO 150 mm
        PRINT COLUMN 0.5 inches;
        PRINT BARCODE "4196176209951"
              WIDTH 1.8 inches  HEIGHT  0.8 inches
        SET BARCODE TYPE "2"
        SKIP TO  150 mm
        PRINT COLUMN 2.4 inches;
        PRINT BARCODE "52"
            WIDTH 0.2 inches  HEIGHT  0.7 inches
        SKIP TO 190 mm
        PRINT "This example of ean-13 is a Magazine. "
            "The ean-2 is the issue,  week number 52 of year."

    END REPORT
```

### 17.4.1 Barcodes 128

The new barcode 128 has variants: EAN-128A,B,C + GS-128

This barcode has 3 different code sets:

**Code Set A** which includes all of the standard upper case alphanumeric characters and punctuation characters together with the symbology elements (e.g. characters with ASCII values form 00 to 95) and seven spectial characters.

**Code Set B** which includes all of the standard upper case alphanumeric characters (e.g., ASCII characters 32 to 127 inclusive and seven special characters.

**Code Set C** which includes the set of 100 digit pairs from 00 to 99 inclusive, as well as three special characters. This allow numeric data to be encoded as two data digits per symbol character.

#### 17.4.1.1 Tilde Method

The special characters and the Application Identifer (AI) encode with the Tilde Method. When using this method, the parentheses are not included in the data encoded; instead the tilde character plus 3 digits are used to represent the AI. Use the following extended ASCII character:

| | |
|---|---|
| ~212 | 2 digit AI |
| ~213 | 3 digit AI |
| ~214 | 4 digit AI |
| ~215 | 5 digit AI |
| ~200 | DEL |
| ~201 | FNC3 |
| ~202 | FNC2 |
| ~203 | SHIFT |
| ~204 | CODE C |
| ~205 | CODE B |
| ~206 | CODE A |
| ~207 | FNC1 |

#### 17.4.1.2 Type 128B

GS1-128: to encode (8100)712345(21)12345678 with the Tilde

Method

```
set barcode type "128B"
skip to 25 mm
print column 0.5 inches;
print barcode "~2148100712345~2122112345678"
```

But 48 character is maximal length to encode (GS1-128).

```
#
#
# Example for barcode ean-128 A,B,C + GS1-128 in pdfreport
# Michael Krauss
# mike@holzamer.de
#
DEFINE i INTEGER
MAIN
    CALL startlog("errorlog")
    START REPORT pgtrex TO "example.pdf"
#select * \d
    FOR i = 1 TO 1
      OUTPUT TO REPORT pgtrex(i)
    END FOR
    FINISH REPORT pgtrex
    RUN "acroread ./example.pdf"
END MAIN
PDFREPORT pgtrex(r)
    DEFINE r INTEGER
OUTPUT
    TOP MARGIN 0 inches
    LEFT MARGIN 0.5 inches
    RIGHT MARGIN 0.1 inches
    BOTTOM MARGIN 0.2 inches
    PAPER SIZE IS a4
FORMAT
```

```
    PAGE HEADER
    SET TEXT FORMAT "utf8"
    SET FONT NAME "Times-Roman"
    SET FONT SIZE 8
#Example for 128-A
    SET BARCODE TYPE "128A"
    SKIP TO 25 mm
    PRINT COLUMN 0.5 inches;
    PRINT BARCODE "CODE-128A"
        WIDTH 3 inches HEIGHT 0.8 inches
#Example for 128-B
    SET BARCODE TYPE "128B"
    SKIP TO 60 mm
    PRINT COLUMN 0.5 inches;
    PRINT BARCODE "CODE-128b"
        WIDTH 3 inches HIGHT 0.8 inches
#Example for 128-C
    SET BARCODe TYPE "128C"
    SKIP TO 90 mm
    PRINT COLUMN 0.5 inches;
    PRINT BARCODE "01234567890122"
        WIDTH 3 inches HEIGHT 0.8 inches
# Example for GS1-128;
# We use here 128B with "~204" is change to code 128C
    SET BARCODE TYPE "128B"
    SKIP TO 130 mm
    PRINT COLUMN 0.5 inches;
    PRINT BARCODE "~204~2120195012345678903~2143102000400"
        WIDTH 6 inches HEIGHT 0.8 inches
END REPORT
```

## 17.5 Printing

PDFREPORTs produce PDF files. REPORTs produce textfiles (ASCII, ISO8859 or other encodings depending on $LANG). They may need different treatment when you print them.

### 17.5.1 PDF

Linux distributions and Mac OSX can print PDF or Postscript files direct to the printer using CUPS (Common Unix Printing System). Even when the target printer does not support PDF or Postscript directly, CUPS will intervene and translate the PPD properties into the printer control language the printer uses. So, when using PDF reports in Aubit4GL applications, output to a filename with a .pdf extension. Then you can view it with the likes of acroread, kpdf, or whichever and from there use CTRL-P to invoke the printer interface. Alternatively from the command line:

```
    lp -d laser rep.pdf
```

If you need to manipulate the PDF files (e.g. to rotate them on printers which cannot do landscape), you may need to install the pdfjam package and use its commands. e.g. :

```
    pdf90 rep.pdf | lp
```

### 17.5.2 ASCII, ISO8895, etc

Traditional reports of ASCII characters are a different matter and may need a bit more intervention from you as an administrator or user. If your printer is not a traditional dot-matrix printer with monospaced characters and fixed width characters, you may need to read on ...

### 17.5.3 CUPS

You can manage print queues on Linux/Unix and Mac OSX systems using CUPS (Common Unix Printing System) which supports both the traditional lp (Bell Labs System V) and lpr/lpd (BSD Berkeley Software Distribution) systems. This means that you can use familiar command line tools like `lpstat`, `lpctl`, `lpadmin`, etc as well as the new CUPS `lpinfo`, `lpoptions`, `lphelp`, etc. Use `info` or `man` to find out more.

CUPS is built around 2 protocols: IPP and PPD.

- IPP is the Internet Printing Protocol designed to allow network browsing and control of printers, queues, and jobs.
- PPD is the Postscript Printer Description. All printers have a .ppd file which lists their capabilities in terms of Postscript values. Unix/Linux has the concept that all files to be printed are passed to ghostscript for interpretation using the ppd file then are filtered by a program appropriate for the printer which will translate the postscript language into the printer's native language (e.g. PCL for HP printers). This eliminates the concept of a **driver** program for each printer.

The pièce de résistance of CUPS is its web interface for managing printers (whether local or remote). Point a browser to `localhost:631` or to `<printserver>:631` and you will be able to configure any and all aspects of printer maintenance and job control.

You will need to have been added to the CUPS list of trusted users via the command: **lppasswd -a <username>** by a user with root privileges. The password supplied to CUPS need not be the same as your normal login password.

If you have a graphical user interface, use your normal browser (e.g. firefox, konqueror, or safari,). If your access is restricted to a text interface use the text browser: lynx

```
lynx localhost:631
```

#### 17.5.3.1 Print Queue Set Up

Unlike most present day applications, traditional 4GL applications produce report files of ASCII text.

The reports produced by 4GL applications are likely to come in two formats:

- 66 lines per page with 80 chars per line in portrait orientation
- 43 lines per page with 120 chars per line in landscape orientation

The above formats fitted the traditional dot-matrix printers of the 1980s which used fixed pitch character sets on continuous fan-fold paper.

Modern printers generally use scalable vector fonts with proportional character sizes on cut sheets of paper (usually A4).

If report files produced by Aubit4GL reports don't fit the printed pages (e.g. the pages creep and/or lines wrap) then you may have to configure a print queue by setting its cpi and/or lpi appropriately so that the 2 report formats print pages correctly on each sheet without line creep or extra blank pages at end of file.

#### 17.5.3.2 cpi and lpi

Set cpi (characters per inch) to a value which will cause the printer to scale the font characters to fit 80 or 120 characters to each line.

Set lpi (lines per inch) to a value which will cause the printer to scale the font characters to fit 66 or 43 lines per page.

Printers vary considerably from model to model, and from make to make, in the size of each one's printable region on a sheet of A4.

A4 has dimensions: 210mm x 286mm (8.25in x 11.654in). Unfortunately most printers cannot print right to the edges of the paper. Generally laser printers can print closer to the edges than inkjet printers.

To determine the dimensions of the printable region, print a file on the printer with the option: *-o page-border=single.* You will see a border printed right on the border of the printable region of the A4 sheet. Measure the width and height of the border rectangle (with an inch ruler if possible). If you don't have an inch ruler, divide the millimetre measurement by 25.4.

Once you know the dimensions of the printable region, calculate cpi by dividing 80 (or 120) character count per line by the horizontal dimension: e.g. 80 / 7.5 (which equals 10.67).

Similarly calculate lpi by dividing 66 (or 43) lines per page by the size (in inches) of the vertical dimension.

Typical lpi values which have worked correctly on some HP printers are:

| | |
|---|---|
| portrait: | 66 lines in 10.625 inches: *-o lpi=6.21* |
| landscape: | 43 lines in 7.5 inches: *-o lpi=5.73* |

Note that cpi and lpi are floating point numbers and you can use 2 or more decimal places.

An example lp command:

```
    lp -s -d laser1 -o cpi=10.5 -o lpi=6.21
```

Note that each option needs its own -o prefix.

The -o options for the lp command come in two categories:

- standard options general to CUPS. Get a list of these from the command:
  `info lp`
  example options are: landscape, portrait, cpi=N, etc
- printer model specific options. Get a list of these from the command:
  `lpoptions -h printer -l`
  where printer is the hostname for the printer. The output from this command is in the form:
  OptionID/Label: Value1 Value2 ... Valuen
  e.g.
  `TonerDensity/Toner Density:  1 2 *3 4 5`
  The * indicates the default value

#### 17.5.3.3 CUPS Documentation

Point your browser (e.g. Firefox , Konqueror, Safari, or lynx) to:
`http://localhost:631` for complete documentation of CUPS.

In the past the printers were set up by sending a stream of codes (in the PCL language) to choose fonts, pitch size, etc., suitable for the output of reports. With CUPS, this is now done by the system administrator who sets up a unique queue for each configuration of a printer using the appropriate -o options.

### 17.5.4 Print Queue Problems

If a print queue goes down (e.g. if it runs out of paper), you can check the queue with the command:

```
    lpstat -t
```

This will show which queues are disabled, e.g.:

```
printer S2 id idle.  disabled since Thu 28 Feb 2009 03:00:42 PM NZDT
```

and which jobs are are waiting in which queues, e.g.:

```
S2-10109   rkh1914    3072   Sat 28 Feb 2009 03:48:12 PM NZDT

S2-10111   ts10273    1024   Sat 28 Feb 2009 04:00:00 PM NZDT
```

After the cause of the problem has been resolved you can then enable a queue with the command:

```
    cupsenable <queuename>
```

which requires root authority. After that you can monitor the queue again with:

```
    lpstat -t
```

until all the jobs have disappeared from the queue (i.e. have been printed), but note:

- typically the first job in the queue (the first one queued after the queue went down) will not print and will stay in the queue
- If so you should reprint it yourself manually, so the users don't miss it (they may not know which ones are missing).

#### 17.5.4.1 a2ps

As an alternative to setting up a CUPS queue with special characteritics, you can use the utility a2ps for one-off formatting of printouts. You will need to have installed the psutils package.

a2ps exploits the ability of printers to scale and rotate content to fit on a page. It allows you to print 2up (or other nup), place headers and borders, and so on. It can be useful for unusually wide report formats. An example call will give you the idea:

```
a2ps -B \
    --medium=A4 \
    --columns=1 \
    --rows=1 \
    --lines-per-page=66 \
    --chars-per-line=160 \
    --font-size=6 \
    $*
```

The above is a sample source for a script named `wprint` which (without the font-size option) can be used to print wide 160 characters per line reports.

# Chapter 18

# 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.

## 18.1 Invoking a logical report

The report function is unchanged - but the calling procedure is enhanced to include:

```
START REPORT report-name TO CONVERTIBLE
```

as well as the familiar TO PIPE/TO FILE etc.

This creates the meta data file (in /tmp) which can be processed later.

### 18.1.1 FINISHing the report

The processing is done via the FINISH REPORT statement, CONVERT REPORT statement or via an external program `process_report`.

The extended Aubit4GL FINISH REPORT statement now accepts the following syntax :

```
FINISH REPORT report-name CONVERTING TO "filename" AS "type" [ USING
"layout" ]
```

(You can also use CONVERTING TO PRINTER, TO PIPE)

```
FINISH REPORT report-name CONVERTING TO EMAIL AS "type" [ USING
"layout" ]
```

```
FINISH REPORT report-name CONVERTING TO MANY
```

### 18.1.2 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.

### 18.1.3 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

depending on 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)

### 18.1.4 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 CONVERTIBLE
OUTPUT TO REPORT r1 (1,2)
FINISH REPORT r1 CONVERTING TO "myfile1.pdf"
   AS "PDF" USING "layout1"
START REPORT r1 TO CONVERTIBLE
OUTPUT TO REPORT r1 (1,2)
FINISH REPORT r1 CONVERTING TO "myfile2.pdf" AS "PDF" # uses def
START REPORT r1 TO CONVERTIBLE
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
```

## 18.2 Saved Meta Data

There are 3 things you can use with the meta data

### 18.2.1 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 what has 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.

### 18.2.2 The layout editor

This is another GTK2 application which embeds the report viewer and allows you to edit a logical report output.

For now 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... (e.g. 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.

### 18.2.3 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 libLOGREPPROC_?.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)

### 18.2.4 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..

## 18.3 Helper programs

process_report

report_viewer

layout_engine

# Chapter 19

# 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.

## 19.1 Coredumps

To find the reason for core dumps, create debuggable files!

If 4glc (or fcompile etc) is core dumping, then recompile the program or form to have debugging information included. To do this set the CFLAGS in incl/Makefile-common to have a `-g`, and recompile the relevant Aubit4GL 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..

```
4glpc -g hello.4gl
```

```
gdb 4glc core
```

Now type `bt` inside gdb - that will give you a backtrace (with any luck).

## 19.2 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.

## 19.3 All other errors

When a 4GL module or form is compiled, the compiler will generate a .err file if the compilation is not successful.

## 19.4 compiler errors

## 19.5 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 20

# Web Services

## 20.1 4GL Web Tools

Mike Aubury has created the following tools to facilite writing both Web Servers and Web Clients in 4GL:

- `4glpc -t WRITE somemod.4gl`
- `fglproto -w somemod`
- `wsdl2fgl somefile.wsdl`

These work with the gSOAP supplied tools (`soapcc2` and `wsdl2h`) to generate 4GL and/or C programs which when linked to the gSOAP library (via the `-lgsoap` option) will handle all the necessary translation to and from XML of C structures and handle the webservice Remote Procedure Call (RPC) processes.

Note: If you cannot run `soapcc2` and `wsdl2h`, then the 4GL web tools will not work. See the install instructions for gSOAP.

Note: If you cannot run the `fglproto` program, it may not have been compiled when you installed Aubit4GL. To fix this, you should be able to compile it in the aubit4glsrc source code distribution/CVS by

- cd compilers/4glc
- make fglproto

### 20.1.1 Client

To access a web service from 4GL the short story is:

```
wdsl2fgl somefile.wsdl # or somefile.h
4glpc -o client.4ae \
    <your 4GL files ...>
    client.c soapC.c soapClient.c \
    -lgsoap
```

For the longer story read on ...

### 20.1.2 Server

To create an Aubit4GL program to act as a web service, the short story is

```
4glpc -t WRITE functions.4gl
4glpc -o functions.o functions.4gl
fglproto -w functions
soapcpp2 -c  prototypes.h
4glpc -o server  \
    soapC.c soapServerLib.c server.c  \
    functions.o prototypes_server.c \
   -lgsoap
```

Again for the longer story, read on ...

## 20.2 WSDL and SOAP

WSDL is the WWW proposed standard Web Service Description Language - an XML format for defining endpoints, services, and data structures for transmission over the world wide web. A WSDL file is the core file for a web service program.

SOAP (originally Simple Object Access Protocol) is a WWW standard for delivery of web services. These 2 standards work together.

## 20.3 gSOAP

gSOAP is an opensource implementation of SOAP which provides:

- automated SOAP and XML data binding for data structures in C (using -c option) or C++ (the default)
- automated program code generation and data mapping of both native and user defined data types. Using gSOAP you can translate to and from XML without needing to write either conversion code or XML verification code.
- an optional webserver - or you can deliver the service as a CGI program - see the gSOAP document in the `gsoap/doc` subdirectory

gSOAP is a prerequisite for Aubit4GL's web service programming.

Get gSOAP from

`http://www.genivia.com/Products/downloads.html`

Select gSOAP toolkit standard from the list of software packages.

For our purposes the important tools provided by gSOAP, in the `gsoap/bin` subdirectory, are:

- `wsdl2h -c` a WSDL/schema importer and data binding mapper tool. For example the command
  `wsdl2h -c -o calc.h http://www.genivia.com/calc.wdsl`
  will output a file: calc.h which translates the WSDL into C language `#define` statements
- `soapcpp2 -c` a stub/skeleton code generator. For example the command
  `soapcpp2 -c calc.h`
  will create for you the following files:

  - `calcStub.h` an annotated copy of the input definitions
  - `calcH.h` declares XML serialiser functions
  - `calcC.c` defines XML serialiser functions
  - `calcClient.c` Client calling stubs
  - `calcLibClient.c` SOAP Client library functions
  - `calcServer.c` Server calling stubs
  - `calcLibServe.c` Server library functions
  - a bunch of other `.xml` and `.nsmap` files

### 20.3.1 Warning

gSOAP licenses `wsdl2h` and the code generated by it for GPL opensource products, educational use, and non-commercial use. If you wish to exploit it commercially, consult the licence which comes with the gSOAP download.

## 20.4 wsdl2fgl

The Aubit4GL tool `wsdl2fgl` generates 4GL programs and calls the gSOAP tools to generate the SOAP .c files needed to access web services.

To create a 4GL web client, you can call `wsdl2fgl` with either a WSDL file or a gSOAP generated .h file.

`wsdl2fgl service.wsdl` will run `wsdl2h` to create the header file service.h first then run `soapcpp2 -c`

`wsdl2fgl service.h` will run `soapcpp2 -c` (without the need to run gSOAP's `wsdl2h`)

`wsdl2fgl` will generate a set of files:

- soapClient.c
- soapC.c
- Client_4gl.c

Check the generated file: Client_4gl.c and look for which functions are available - they will be of the form:
`aclfgl___ns2__funcname( ... )`
where `aclfgl_` is the usual prefix that 4GL applies to 4GL function names to avoid namespace clashes with standard C library calls. Put appropriate CALL statements into a MAIN ... END MAIN function using the appropriate arguments and link your main module with all the above stuff to create your 4GL web client. In your 4GL code omit the aclfgl_ prefix that you find in the Client_4gl.c file. The `__ns2__` or `__ns3__` or whataver are gSOAP generated function prefixes similar in spirit to Aubit4GL's `aclfgl_` prefix.

### 20.4.1 Client Example

For a real world example, download the wsdl file from
`http://xmethods.net/ve2/ViewListing.po?key=427560`
and save the file as driving.wsdl

Now run the command: `wsdl2fgl driving.wsdl`

Looking at the file: Client_4gl.c we find:

```
     aclfgl___ns2__getdirections (
          // Parameter 1 - fromAddress CHAR(...)
          // Parameter 2 - toAddress CHAR(...)
          // Parameter 3 - distanceUnit CHAR(...)
          // Parameter 4 - expresswayEnabled CHAR(...)
    )

     aclfgl___ns3__getdirections (
          // Parameter 1 - fromAddress CHAR(...)
          // Parameter 2 - toAddress CHAR(...)
          // Parameter 3 - distanceUnit CHAR(...)
          // Parameter 4 - expresswayEnabled CHAR(...)
    )
```

One is calling via service name `drivingSOAP`, the other via `drivingSOAP2`.

So, using the first one with applicable parameters we can write a file (say `main.4gl`):

```
MAIN
   DEFINE lv_data char(1024)
   CALL __ns2__getdirections(
      '1600 Amphitheatre PArkway Mountain View, CA, USA'
      '2775 Middlefield Road Palo Alto, CA, USA',
      'miles',
      'true')
   RETURNING lv_data
   DISPLAY lv_data CLIPPED
END MAIN
```

Note: The `Client_4gl.c` file has a function declaration for
`aclfgl__ns2__getdirections()`
The Aubit4GL compiler 4glpc prepends the `acl4gl_` to all 4GL function names when translating to C. So we have to omit the the `aclfgl_` prefix from our 4GL CALL statement. Rest assured that the 4glc program will supply the missing `aclfgl_` prefix to access the C function.

(The output gives directions from Google HQ to a Starbucks close by)

Now - to compile it all up :

```
4glpc -o client.4ae main.4gl \
   soapClient.c soapC.c \
   Client_4gl.c -lgsoap
```

## 20.4.2 Web Server

In order to export a function, a declaration of the function has to be created and used. This can be done using the `fglproto` program. First create a datafile containing your 4gl definition with :

```
4glpc -t WRITE somemod.4gl
```

You can do this with multiple 4GL modules. Just run the command for each 4gl module. At this stage though, you only need to specify modules which

1. Contain functions you wish to export

2. Contain functions used directly in RETURN statements for functions you are exporting

You may wish to consolidate the functions into a single module - or create a wrapper function to call your real functions so you don't export all of the functions in a complex 4gl!

Remember - here we are wanting to create stub functions to manage the Web service. Later we will need the modules containing all the other functions that are needed to link our application together, but that is not at this stage ...

After you have run `4glpc -t WRITE` for each module, we can generate the stub functions for all non-local functions in our 4gl module. We do this with `fglproto`:

```
fglproto -w somemod
```

If you have specified more than one 4gl module - add more to that one :

```
fglproto -w somemod1 somemod2
```

Remember - here we will create the logic to export all of the functions not marked as local - you don't need to use all the modules for the application, you can omit any that are needed just for linking..

This should generate some files : `prototypes_server.c` `prototypes_client.c` `blacklist` `prototypes.h`

(You can ignore `blacklist` completely). Now, in order to create a server you need to use `soapcpp2` to process our `prototypes.h` file :

```
soapcpp2 -c prototypes.h
```

(We're using normal C generation so we need to use the `-c` option to tell `soapcpp2` not to generate C++ code)

That should generate us some more files ;-)

For our server - we will need to use the soapcpp2 generated C files :

```
soapC.c soapServerLib.c prototypes_server.c
```

You will also need a function to start the server itself - you can use this basic one as a starting point (see `tools/test/gsoap/server.c`) :

```
#include "soapH.h"
/* get the gSOAP-generated definitions */
#include "fglserver.nsmap"
/* get the gSOAP-generated namespace bindings */
#include <math.h>
int aclfgl_run_server(int n)
{ int m, s; /* master and slave sockets */
   int port;
   struct soap *soap = soap_new();
   if (n==0)
   { soap_serve(soap); /* serve as CGI application */
     soap_done(soap);
     free(soap);
     return 0;
   }
   port=A4GL_pop_int();
   printf("Listening on port %d\n",port);
   m = soap_bind(soap, NULL, port, 100);
   /* bind to the port supplied as command-line arg */
   if (m < 0)
   {
      soap_print_fault(soap, stderr);
      exit(-1);
   }
   fprintf(stderr,
    "Socket connection successful: master socket = %d\n",
    m);
   for (;;)
   {
      s = soap_accept(soap);
     fprintf(stderr,
      "Socket connection successful: slave socket = %d\n",
      s);
     if (s < 0)
```

```
        { soap_print_fault(soap, stderr);
          exit(1);
        }
        soap_serve(soap);
        soap_end(soap);
      }
     soap_done(soap);
     free(soap);
     return 0;
   }
```

You can call this function from within 4gl with something like:

```
main
   call run_server(9090)
end main
```

Putting it all together ...

Assuming you've put

- aclfgl_run_server() function in a file called `server.c`,
- `main .. end main` in a file called `main.4gl`:

```
4glpc -o server \
   soapC.c soapServerLib.c \
   server.c functions.o \
   prototypes_server.c \
   main.4gl
   <all your 4gl modules> \
  -lgsoap
```

Eg. if your module containing functions you want to serve is called `functions.4gl`' but needs `subrouts.4gl` to link :

```
4glpc -t WRITE functions.4gl
fglproto -w functions
```

```
soapcpp2 -n -c prototypes.h
4glpc -o functions.o functions.4gl
4glpc -o subrouts.o subrouts.4gl
4glpc -g -o server \
   soapC.c soapServerLib.c \
   server.c \
   functions.o subrouts.o \
   main.4gl \
   prototypes_server.c \
   -lgsoap
```

Now - if you want to create a client to use the webservices for these functions - its very straightforward.

You simply need to compile your 4gl along with the generated client code and the gsoap library, and optionally alter the function to include the URL where they will be served.

```
4glpc -g -o client.4ae client_4gl \
   soapC.c soapClient.c \
   prototypes_client.c \
   <your 4gls> \
   -lgsoap
```

The functions can have an optional first parameter specifying the URL where the functions will be served from. This is a default of
`http://localhost:9090`
if not specified, but the default can be manually changed in the `prototypes.h` generated by `fglproto -w`

eg.

```
main
   call get_tabname(54321)
    returning lv_tabname
   call get_tabname("http://localhost:9090",54321)
    returning lv_tab name
    display lv_tabname,":"
end main
```

This assumes that you have exported a function similar to :

```
database test1
main
    call run_server(9090)
end main
function get_tabname(lv_id)
   define lv_tabname char(18)
   define lv_id integer
   display "lv_id=",lv_id
   select tabname into lv_tabname
      from systables
      where tabid=lv_id
   return lv_tabname
end function
```

See the code in `tools/test/gsoap` for this example.....

### 20.4.3 Limitations

#### 20.4.3.1 Single Threaded

All functions are served from what should be thought of as a single threaded application.

There is every likelihood though that you will not connect back to the service which you have previously called. No state will be preserved etc. You should therefore consider function calls to be atomic. For example, you cannot declare a cursor in one web service call, then read it in another. That cursor might well not exist when the 2nd call runs.

#### 20.4.3.2 Limited Datatypes

We cannot handle arrays.

#### 20.4.3.3 Unsupported Services

If there is a particular web service that you want to use which cannot be compiled, please pass on the details to:
`support@aubit.com`

# Chapter 21

# Revisions

## 21.1 2010-8-23

- Revise sections re install and configure
- Moredetail about configuring pg, mysql, sqlite3
- Describe new features: TODO, Web services, XML interface to new Visual Display apps
- Extensions to INPUT/CONSTRUCT statements: callbacks, etc
- COPYOF and COPYBACK operators
- New PDFReport features (e.g. barcodes)

## 21.2 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

## 21.3 2005-9-9

Just editing changes:

- Fix numerous spelling mistakes
- Fix some infelicities of English expression
- Fix punctuation, syntax, and some grammatical errors

## 21.4 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

## 21.5 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

## 21.6 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.

## 21.7 Problems

- Tables bug in latex2html is now fixed (thanks to Ross Moore of MacQuarie University)
- Stylesheets still not right (latex2html configuration problem?)

# Chapter 22

# Environment Variables

The following list of environment variables was derived from the configurator program's description file.

There is a great deal of complexity in dealing with different

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=EC, 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_
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 Ex-

tended 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/aubit4glsr
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_ERRHOOK Name of library file containing function errolog().
Omit the .so suffix. The file must be known to the system vis LD_LIBRARY_PATH or ldconfig

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=Y
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.

A4GL_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

### 22.0.1 Version 1.2

The following environment variables have come into use by Aubit4GL since version 1.10:

A4GL_CONSTRUCT_NO_MATCHES RUNTIME/SQL/UI
Disable generation of MATCHES keywords
=YES|NO
This setting stops construct from generating a MATCHES when it sees a [, * or ? You can use this setting if your target SQL does NOT have the MATCHES keyword.. See also A4GL_CONSTRUCT_LIKE.

A4GL_CONSTRUCT_LIKE RUNTIME/SQL/UI
Enable generation of LIKE for construct
=YES|NO
If set to Yes this allows % and _ to be detected and used to for a LIKE within a generated construct string. This uses the same flag (ismatch==1 for MATCHES, ==2 for LIKE) as for matches. So - If it looks like a matches, matches takes priority

(unless CONSTRUCT_NO_MATCHES IS SET).
Note: This does NOT translate from an entered MATCHES format string to a LIKE format.

A4GL_DIALOG_OK LANG Text to display on OK button

A4GL_DIALOG_CANCEL LANG Text to display on CANCEL button

A4GL_DIALOG_ABORT LANG Text to display on ABORT button

A4GL_DIALOG_RETRY LANG Text to display on RETRY button

A4GL_DIALOG_IGNORE LANG Text to display on IGNORE button

A4GL_DIALOG_YES LANG Text to display on YES button

A4GL_DIALOG_NO LANG Text to display on NO button

A4GL_DIALOG_ERROR LANG Text to display for ERROR window

TUICANCELKEY TUI Allows the assigning of interrupt key. For example in aterm's I set it to 263

A4GL_USE_PANGO_ML UI Allow markup for GUI widgets labels
=YES|NO When set to Yes, any form label can use the pango-markup-language: (e.g.: DISPLAY '<span background='blue'>Blue BG</span>' to my_label)

SWAP_SQLCA62 SQL Enable sqlca.sqlerrd[6] (ROWID/OID of last inserted record) translation
=Y|N
This is apparently needed only with PostgreSQL (with IFX compatibility patches)

A4GL_SYSCOL_VAL SQL the name of the 'syscolval' table

A4GL_SYSCOL_ATT SQL the name of the 'syscolatt' table

A4GL_CLASSPATH COMPILE/RUNTIME Path to search for 'import' packages

RESTARTLOG RUNTIME Restart the error log
=YES|NO
This causes the CALL startlog('...') to restart with an empty log file each time the program is run. The default is to append to the end of the current log

A4GL_EC_LOGSQL COMPILE Add extra SQL logging information
=YES|NO
This option adds extra code into the generated ESQL/C code which will call a library function to log SQL calls as they are made. The actual SQL will be written to a file only if the variable 'LOGSQL' is set at runtime. See also LOGSQL

COMSPEC COMPILE/RUNTIME On Windows used to determine that we are on Windows. Not to be changed by user - this is a Windows system environment variable

CORE_ON_ASSERT RUNTIME/DEBUG Force a core dump when encountering an assert. Aubit4GL uses a series of assertions to check data being used internally. When one of these assertion fails - it generally indicates something has gone wrong internally. Setting this option will force a Core dump (Seg Fault) which should allow any attached debugger to check exactly where the assertion failed and investigate why it failed.

DATE_AS_ISO_DATE_STRING SQL/ODBC Use character strings for dates
=YES|NO
This specifies that when communicating with the ODBC driver, dates should always be refered to as strings. Some ODBC do not correctly handle dates using the proper ODBC SQL_DATE datatype - when this situation occurs, specifying DATE_AS_ISO_DATE_STI
may help

DTIME_AS_CHAR SQL/ODBC Use character strings for datetimes
=YES|NO
This specifies that when communicating with the ODBC driver, datetimes should always be refered to as strings. Some ODBC do not correctly handle datetimes using the proper ODBC SQL_C_DATETIM
etc.W hen this situation occurs, specifying DTIME_AS_CHAR may help

DUMP4GL DEBUG generates a file /tmp/file.out
=YES|NO
This generates a file /tmp/file.out which gets written to as the 4glc compiler compiles 4GL code. This can be useful in debugging the 4glc compiler

EC_EXT COMPILE Specify the extension for an ESQL/C file
Defaults: Informix=.ec Postgres=.cpc SAP=.cpc

A4GL_INCOMPAT_AT_RUNTIME COMPILE Allow incompatible statements at compile-time, stop at run-time
=YES|NO
When A4GL_INCOMPAT_AT_RUNTIME is set to YES, statements that are not implemented or not possible when using particular database (when it is known at compile-time) will not cause compile-time error. Instead, they will compile successfully, and cause the error at run-time. This will allow 4GL source code using such features to be compiled, but if execution of it is attempted, program execution will be halted. Default is to abort with error on compile-time (when target database is known at compile time, which is tipically only when using A4GL_LEXTYPE=EC)

INFORMIXOPTIONS UI Use informix case matching for NEXT OPTION/SHOW OPTION/HIDE OPTION
=YES|NO
This is the default setting for case options. Informix seems to be case sensitive for NEXT OPTION, but case insensitive for SHOW OPTION and HIDE OPTION Using this setting will emulate this behaviour

CASEOPTIONS UI Always use case when matching for NEXT OPTION/SHOW OPTION/HIDE OPTION
=YES|NO
This setting will ensure a case sensitive match for all the menu operations This is the default behaviour for NEXT OPTION under informix, SHOW and HIDE option seem to be case insenitive on informix

CASEIGNOPTIONS UI Never use case when matching for NEXT OPTION/SHOW OPTION/HIDE OPTION
=YES|NO
This setting will ensure a case insensitive match for all the menu operations This is the default behaviour for SHOW/HIDE OPTION under informix, NEXT OPTION seem to be case sensitive on informix

EMAIL_RECIPIENT REPORT Email address to use for START REPORT TO EMAIL
This setting is used to specify a default email address to send a convertible report to. Emails are sent using send_report() function in the fgl_smtp module

FGLFRMCOMPILE FORM Explicitly state if a form compile is to use 4GL rather than PERFORM syntax
=YES|NO
Setting this will enable/disable the use of keywords used only in the INSTRUCTIONS section of a perform screen which may otherwise interfere with otherwise legitimate identifiers

FLASH_BADMENUKEY UI Flash the screen when an error occurs in a menu
=YES|NO
When a key is pressed in a menu which does not correspond to an option, then this indicates that the screen should flash as a non-audiable warning. See BEEP_BADMENUKEY

BEEP_BADMENUKEY UI Ring the terminal bell when an error occurs in a menu
=YES|NO
When a key is pressed in a menu which does not correspond to an option, then this indicates that the terminal should beep as a audiable warning.Note : Not all terminals are capable of making the beep noise. See FLASH_BADMENUKEY

FORCE_CLOSE SQL Force the closing of the current database connection
=YES|NO
This option will force the closing of the database connection when the program exits

FORCE_ROLLBACK_AT_EXIT SQL Send an explicit rollback work on closing the connection
=YES|NO
ODBC specific. This option requests that a rollback work be sent before closing a database connection

FREE_SQL_MEM SQL/COMPILE Free SQL memory when compiling the 4gl code

=YES|NO
This option should not be required as the 4gl compiler will free all the memory at the end of compiling a module. If you are finding you are running out of memory you could try setting this option.

FUDGE_STATUS COMPILE Add some extra code to clear down the status variable
=YES|NO
EC/C generation only. This option forces the clearing down of the status variable.

IFX_LIBS COMPILE Informix libraries to use on the link line This setting should be automatically set by the configure script If this isn't set correctly, it should be set manually to the result of 'esql -libs'

INFORMIX_ROUNDING RUNTIME Specifies which rounding algorithm to use
=YES|NO
This option specifies using an informix compatible rounding algorithm

LOGALL RUNTIME/UI Log non-printable keys
=YES|NO
This specifies that all key presses should be logged, as opposed to just the printable characters See KEYLOG and KEYFILE

NAMESPACE COMPILE Namespace to use for compiling programs

NEED_SIGCHLD RUNTIME Add a SIGCHLD event handler
=YES|NO
Some SQL backends (for example some versions of Informix) raise a SIGCHLD event which must be caught. This option adds the relevant event handler

NEED_SIGPIPE RUNTIME Add a SIGPIPE handler
=YES|NO
Some SQL backends (for example some versions of Informix) raise a SIGPIPE event which must be caught. This option adds the relevant event handler

NO_CONV_ERR RUNTIME Don't raise an error if a conversion into a variable fails
=YES|NO

A4GL_DLL_EXT COMPILE Class shared library extention

SO_EXT COMPILE Normal Shared library extention

KEY_RECALL UI Keycode for a recall field key
This option specifies a keycode for a field recall key. This is only effective for the HL_... series of UI modules

KEY_RECALL_LAST UI Keycode for a recall field key
This option specifies a keycode for a field recall key (last value only). This is only effective for the HL_... series of UI modules

LIST_ERRS COMPILE Try to allow multiple errors to be reported when compiling a 4GL module
=YES|NO
Typically the 4gl compiler will stop at the first error. Setting this option will try to continue the compilation so that multiple errors in the 4gl module will all be reported

LOGSQL RUNTIME/SQL Filename to log SQL statements to
Some SQL modules will allow for SQL logging at runtime. LOGSQL should contain the filename to log these SQL statements in

RUNNING_TEST COMPILE Internal variable
=0|1
This variable indicates that the compilation/run is part of the regression thests. This enables code to store errors generated during regression tests

LOG_TEXT COMPILE Internal variable
This variable should contain a subdirectory and is used to indicate where to store errors generated during regression tests

A4GL_TRACEDLL UNKNOWN UNKNOWN
=YES|NO
Trace DLL calls. Calls will be written to a file called /tmp/trace.txt

PREFIX_DLLFUNCTION RUNTIME Add an underscore when calling a DLL function
=YES|NO
Some platforms require that functions in a DLL are prefixed with an underscore This option ensure that all functions are called with the expected name

ODDOPTIONS UI Reset options when opening a new window
=YES|NO
For some bizarre reason, Informix resets the options display attribute when you open a new window. This option is used for compatibility

ONKEY_ACCEPT UI Apply different logic for ON KEY (ACCEPT)
=YES|NO
Traditional key handling Assume F1 is the Accept Key... The normal truth table would be :
ON KEY (f1) ON KEY (ACCEPT) Press Key FGL_LASTKEY ACTION
N N F1 F1 Exit Display *
Y N F1 F1 Do ON KEY(f1) *
N Y F1 N/A Exit Display *
Y Y F1 F1 Do ON KEY(f1) *
ie - ON KEY(ACCEPT) is completely ignored
Setting ONKEY_ACCEPT=Y changes this :
ON KEY (f1) ON KEY (ACCEPT) Press Key FGL_LASTKEY ACTION
N N F1 F1 Exit Display
Y N F1 F1 Do ON KEY(f1)
N Y F1 ACCEPT Do ON KEY (ACCEPT)
Y Y F1 F1 Do ON KEY(f1)

SHELL UNKNOWN UNKNOWN

SPACESCORE FCOMPILE Convert underscores to spaces when compiling a form
=YES|NO
This option changes any '_' characters to spaces when compiling a form

START_ALLOC RUNTIME Performance enhancement
=YES|NO
This option allocates and then frees 10Mb of memory when the program starts. This normally means that subsequent memory allocations will run much faster.

TRACE_AS_TEXT REPORT Enable XML style text logging of convertible reports
=YES|NO
This options makes the convertible reports write their meta data in an XML style output file

WAIT_FOR_GDB_ATTACH RUNTIME Wait for a debugger to connect to this process on SegFault
=YES|NO
When a program segfaults it can do one of three things.
1) Die (possibly with a core dump)
2) Start the debugger
3) Wait for a debugger on another terminal to connect to this process
This option specifies that we should wait (for up to 5 minutes) for a 'gdb -p pid' to failed process

DBL2DEC_USING RUNTIME Use a USING string to convert a double to a decimal
=YES|NO
This uses a USING string as the basis for converting from a double to decimal

DEBUGFILE RUNTIME debug filename Specifies the debug filename to use (normally debug.out)

LOCALOUTPUT COMPILE strips any leading directories while compiling
=YES|NO
This option specifies that any leading directories are removed when compiling modules. This means that compiling directory/module.4gl may (for example) generate module.c in the current directory rather than directory/module.c

LOGREP REPORTS Internal logical reports variable This is used internally - do not set it manually

LOGSQLFUNCTIONS COMPILE Save used stored procedure names
=YES|NO
This option will create a file called "/tmp/sqlcall.log" which should contain all the SQL functions (stored procedures) which are called from the module being compiled. These can be collected to determine what functions/procedures might need porting/copying when moving to a new database/rdbms.

LOG_STRINGS UNKNOWN UNKNOWN
=YES|NO
This will log all of the literal strings encountered in the source-code in a file called /tmp/strings.log

NEVER_CONVERT UNKNOWN UNKNOWN
=YES|NO

ALWAYS_CONVERT UNKNOWN UNKNOWN
=YES|NO

IGNORE_DATEFMT UNKNOWN UNKNOWN
=YES|NO

MAPSQL UNKNOWN UNKNOWN

MONITORPORT UI Specify the TCP/IP port for monitoring You will need to edit lib/libaubit4gl/fglwrap.c and add
#define USE_MONITOR to use this functionality

MONITOR4GL TUI Writes to a 'monitor' TCP/IP connection the current screen
=YES|NO
This option will log the current screen to. You can use 'telnet host $MONITORPORT' to view the output. You will need to edit lib/libaubit4gl/fglwrap.c and add
#define USE_MONITOR to use this functionality See also MONITORPORT

NOSETODBCCURSORS ODBC Use ODBC driver manager cursors
=YES|NO
If this is set to Y then the odbc sql connector will use the driver manager cursorlibrary if available. This is done using the following ODBC call :

SQLSetConnectAttr (hdbc, SQL_ATTR_ODBC_CURSORS, (SQLPOINTER) SQL_CUR_USE_ODBC, (SQLINTEGER) 0);

NO_INIT_COL_CLR TUI/HL_TUI Inhibit the clear screen on a refresh from linemode
=YES|NO

LOGPCODE PCODE log debugging information to a file called pcode.run
=YES|NO
This is read directly from the environment and so can't be set in an aubitrc file

PADNULLSTRING RUNTIME sting concatenation behaviour
=YES|NO
This option changes the way a string concatenation would work if one or other of the strings is null.

PG_COPTS COMPILE Postgres C compiler flags

SAVE_COMMENTS COMPILE Filename to save comments to
=YES|NO
This option specifies the filename to save any comments encountered in the sourcecode to. These comments can then be more easily analysed.

AUBITPLUGINDIR COMPILE/RUNTIME Specifies the directory where plugins reside This is typically $AUBITDIR/plugins-version

INPUTREQUIREDTYPE TUI sets the REQUIRED method for an INPUT
=FIELD|INPUT
This specifies when the REQUIRED tag on a field on an input is checked.

CLRFIELDSTATUS TUI Clear 'field touched' after the BEFORE INPUT
=YES|NO
In certain versions of Informix4GL it appears that field status flags are cleared *AFTER* a BEFORE INPUT/BEFORE CONSTRUCT. This means that any DISPLAYsdone during the BEFORE INPUT or BEFORE CONSTRUCT do not set the field_touched flags for the fields. Setting this flag emulates this behaviour..

A4GL_FRM_BASE_LIST RUNTIME Specify a list of extensions which may be used for a formname If set - this is a colon separated list of file extensions which may be used when opening form files. If not set, then the standard FRM_BASE_EXT will be used

SETODBCCURSORS UNKNOWN UNKNOWN

SUPPRESSWARNINGS UNKNOWN UNKNOWN

TARGET_OS UNKNOWN UNKNOWN

TRIMFIELD UNKNOWN UNKNOWN

USECURSORFORLOAD UNKNOWN UNKNOWN

A4GL_LOGSQLERR UNKNOWN UNKNOWN

A4GL_PARSER UNKNOWN UNKNOWN

A4GL_QUOTE_OWNER UNKNOWN UNKNOWN

A4GL_RDYNAMIC UNKNOWN UNKNOWN

A4GL_NUMERIC UNKNOWN UNKNOWN

A4GL_DB_NUMERIC UNKNOWN UNKNOWN

A4GL_SCANF_NUMERIC UNKNOWN UNKNOWN

ALWAYS_CONVERT_PREPARED UNKNOWN UNKNOWN

ASQLERRM UNKNOWN UNKNOWN

CONSTRUCT_MATCH_FIX UNKNOWN UNKNOWN

DBNAME UNKNOWN UNKNOWN

DOING_CM UNKNOWN UNKNOWN

DROP_WHERE_CURRENT_OF UNKNOWN UNKNOWN

ENV_NOT_SET_AS_STR UNKNOWN UNKNOWN

ERROR_ON_NVALS UNKNOWN UNKNOWN

FCOMPILE_SILENT UNKNOWN UNKNOWN

HALFDELAY UNKNOWN UNKNOWN

HASHNOCOMMENT UNKNOWN UNKNOWN

HIDEEMPTYBUTTONS UNKNOWN UNKNOWN

KEEPNLONREAD UNKNOWN UNKNOWN

BINDDATEASINT UNKNOWN UNKNOWN

SINGLEFORM FCOMPILE Use 'single form mode' when compiling
=YES|NO
This option emulates the behaviour of Informix4GL in just overlaying multiple screens over each other. (Normally Aubit4GL will put separate screens on separate screens!) This is needed to emulate a "feature" of Informix4GL which allows displaying of data to fields on the second screen to overwrite the screen displayed for the first (to blank out fields for example)

A4GL_NULLBADARGVAL RUNTIME Returns NULL instead of ' ' from ARG_VAL This option returns NULL when ARG_VAL is called with an invalid parameter

C4GLFUNCRETCOMPAT COMPILE Sets the compiler to ignore any surpluss return values
=YES|NO
The (some versions of the) Informix c4gl silently discards extra return values. Setting this will emulate that behaviour

ESQLDESCRIPTORLENGTH ESQL/RUNTIME Sets the maximum length of the ESQL descriptors for the Informix connector This sets the maximum length (normally 128) of the informix descriptor strings. These are used internally for prepares and cursors etc. Some versions of the Informix ClientSDK have a smaller size for the descriptor and this setting can be changed accordingly

# Chapter 23

# This Manual

This manual was produced using LyX and LaTeX, both of which come out of the box with Linux distributions. The sourcefile for the manual is the file: `a4glman.lyx`

The document is typeset to be printed on A5 paper so that the resulting bound volume fits easily on normal bookcase shelves. If you want to change this to A4 then in LyX do the following: Document -> settings -> Page Layout and replace A5 with A4.

## 23.1 LyX

LyX is a GUI frontend to the LaTeX typesetting software which allows you to produce professional-quality typesetting without the need to learn the TeX language, or to know the rules of professional quality typesetting. Please note that, when properly typeset, the inner margins should each be about half the size of the outer margins so that when the book is opened out to lie flat, the margin whitespace between text blocks should appear equal.

LyX documents are text files and can therefore also be edited manually with standard editors such as vi(m), (x)emacs, etc. They are compact (about 3 kilobytes per printed page). Typically the PDF version of a LyX file will be twice that size. LyX files are converted into the TeX language (also text) and then compiled by the program: `latex` into a device independent .dvi format (binary) which various converters can render into .ps (Postscript), .pdf (Portable Document Format), .html (HyperText Markup Language), .odt (Open Document Format), plain text using common opensource utility programs, and several other formats which require 3rd party software.

## 23.2 TEX

The TEX typesetting application is the result of nearly 20 years of work by Donald Knuth, Professor Emeritus of Stanford University who put it into the public domain. Knuth is the author of the famous quintet of books entitled THE ART OF COMPUTER PROGRAMMING which has been the staple text of university computer science courses for over 20 years. Knuth derived the name TEX from the Greek word (techne) which means both art and skill.

In Greek: $\tau\epsilon\chi\nu\eta$

It is pronounced tek but as Knuth says: if you say it aloud, the computer screen should become moist. (The Greek letter X (chi) sounds a bit like the ch in loch). Knuth wrote TEX because he was concerned at the poor quality of the typesetting of the early editions of his own books and at the expense of getting it done correctly. The stated aim of TEX is to produce beautiful books.

## 23.3 LATEX

LATEX is a set of TEX macros created by Leslie Lamport.

LYX is a document processor rather than a word processor. It relies on the typesetting features of LATEX. Using LYX you concentrate on the content and trust LATEX to look after the presentation.

LATEX provides the following document processing benefits which are not available in word processors:

- Consistent sizing and spacing and layout of headers for chapters, sections, subsections, etc.

- Appropriate kerning of characters (squeezing or spreading of adjacent fat and thin letters)
- Ligatures (special glyphs) for letter combinations such as ff, fl, fi, etc.
- Automatic elimination of widows and orphans
- Treating pages and paragraphs (rather than lines) as units for typesetting
- Correct pagination: Odd pages on the right, Chapters starting on odd pages, etc.

## 23.4 PDF and HTML

### 23.4.1 PDF

To produce a PDF version of this manual, use the LyX File Menu option:
File -> Export -> PDF(pdflatex)
This will provide Table of Contents navigation and will embed the Latin Modern Fonts and the 4 or so images into the output file. The other PDF export options (dvipdfm and ps2pdf) will not work correctly.

Alternatively you can do the export to PDF from the command line:
`lyx -e pdf2 a4glman.lyx`

It is important to use Latin Modern as the font for PDF. Latin Modern is a scalable vector font which works well on screen as well as on the printed page. LaTeX's default bitmap fonts are bungled by the Adobe Acrobat PDF reader program.

In order to get URLs properly typeset, similarly use the hyperref TeX package.

In order to activate PDF navigation, you need to conditionally execute the TeX \pdfcatalog statement within a TeX `\ifpdf ... \fi` clause.

Here is the LaTeX source included in the LyX preamble:

```
\usepackage{hyperref}
\usepackage{ifpdf}
```

```
\ifpdf
  \usepackage{lmodern}
  \pdfinfo{ /Title (Aubit4GL Manual)
          /Subject(Aubit 4GL Programming)
         /Author(John O'Gorman) }
        \pdfcatalog{/PageMode(/UseOutlines)}
\fi
```

The somewhat puzzling discrepancy in the syntax above with regard to slashes is because

- the backslashed keywords are in the TEX language
- the forwardslashed keywords are embedded PDF language (within the TEX \pdfinfo macro)

### 23.4.2 HTML

To convert LYX/LATEX documents into HTML format, we use the program latex2html. With some versions of latex2html the following adaptation may be needed to render tables correctly:

```
  }
push (@pieces, $after);
+ $within_preamble=0
}
print "$replacements new-command"
```

In the above the + line is to be added (without the +). The rest is context which allows you to locate where to insert the `$within_preamble=0` statement. As is evident, latex2html is a perl script (usually found in /usr/bin). If it is missing from your distribution, install it from the openSUSE DVD or failing that from a TEX CTAN.

To produce the HTML files from within LYX, take the following steps:

1. Create a slightly cut down version of the primary document: a4glman.lyx by stripping a few things from the LYX file: the preamble and Greek characters. The cut down version is: a4glmanhtml.lyx

2. Export a4glmanhtml to TEX format
3. Convert the TEX format to HTML using latex2html

#### 23.4.2.1 Export LYX to TEX

Use the LYX File menu option:
File -> export -> LATEX(pdflatex)

This will produce a file: a4glmanhtml.tex

You can also produce a tex file from the command line:

```
lyx -e latex a4glmanhtml.lyx
```

Remove the \ifpdf statement from the preamble:

```
sed -i '/^\usepackage{ifpdf}/,/^\fi/d' a4glman.tex
```

This last step is necessary because latex2html cannot cope with the \ifpdf statement.

In fact, we have chosen to remove the whole preamble (which is user-entered to add PDF functionality) and remove the code which produces Greek characters using the following sed commands:

```
sed -e '/\\begin_preamble/,/\\end_preamble/d' \
    -e '/In Greek:/,/\\end_inset/d' \
     a4glman.lyx > a4glmanhtml.lyx
```

#### 23.4.2.2 Convert TEX to HTML

From the command line:

```
latex2html -split 3 \
    -link 4 \
    -local_icons \
    -show_section_numbers \
    -html_version 4.0 \
    a4glman.tex
```

The `split 3` option will cause latex2html to split the document into separate html pages for each subsection and higher partitions (sections, chapters).

The `link 4` option will render a table of contents with navigation links for each subsubsection and higher partitions (subsections, sections, chapters).

The result of the above command will be the creation of a directory: a4glmanhtml which will be populated with HTML files including an index.html file - all linked appropriately with `<a>` tags.

Of course, when you convert to HTML, you lose all the special TEX typesetting features and rely on the standard CSS stylesheets to lay out the document. These you can edit to improve the presentation of the HTML.

The latex2html command above creates set of html files in the directory: a4glmanhml and the directory contains many dozens of files with names: index.html node1.html node2.html ... node159.html

### 23.4.3 Makefile

To help automate the above translation processes, we have a Makefile:

```
L2H=latex2html
L2HARGS=-split +3 -link 4 -local_icons -show_section_numbers\
         -html_version 4.0
all: a4glman.tex a4glman.pdf a4glmanhtml \
     tarball htmltarball
clean:
        rm *.log
srcclean:
        rm *.pdf *.tex *.aux *.log
a4glmanhtml.lyx:a4glman.lyx
        sed -e '/\\begin_preamble/,/\\end_preamble/d' \
            -e '/^In Greek:/,/\\end_inset/d' \
           a4glman.lyx > a4glmanhtml.lyx

a4glmanhtml.tex:a4glmanhtml.lyx
        lyx -e latex a4glmanhtml.lyx

a4glmanhtml: a4glmanhtml.tex
```

```
        ${L2H} ${L2HARGS} a4glmanhtml.tex
a4glman.pdf:a4glman.lyx
        lyx -e pdf2 a4glman.lyx
htmltarball:
    tar cvfz ../a4glmanhtml.tar.gz a4glmanhtml
tarball:
    tar cvfz ../a4glman.tar.gz .
```

Be aware that the indented lines above MUST have tabs (not spaces). Otherwise `make` will not work!

# Index

BARCODE, 255
Barcode, 257, 260
barcode, 225

PostgreS, 144, 145
Postgres, 146

Finis