



*by S.Metzeler*

**an Object Oriented  
Framework for  
Oberon-2**

**Release: 1.7.6**

*Date of printing: Tuesday, 03 February 2009*



Page intentionally left blank

The Amadeus set of development tools  
were written and designed  
by **Stefan Metzeler**

You can reach the author for support questions, training  
and consulting contracts at the following address:

Stefan Metzeler  
Ch. du Stand 19E  
1024 Ecublens  
SWITZERLAND

*Telephone:* +41-21-881.57.54  
*Fax* : +41-21-881.57.55  
*Email* : [amadeus@neomailbox.com](mailto:amadeus@neomailbox.com)  
*Internet* : [www.amadeusITSolutions.com](http://www.amadeusITSolutions.com)

**Copyrights and Trademarks:**

Amadeus, Amadeus-2 and Amadeus-3 are © 1987 - 2007 by Stefan Metzeler

Btrieve is © by Pervasive Inc; previously by Btrieve Technologies Inc; previously a division of Novell Inc.

Pervasive.SQL © by Pervasive Inc

Extacy is © 1991 - 1995 by xTech

XDS is © 1996 - 2002 by XDS Ltd.

Linux is Freeware, originally written by Linus Thorvald

MS Windows, MS C, Visual C++, Visual Basic are © by Microsoft Inc.

Oberon is <sup>TM</sup> of the Swiss Institute of Technology, Zürich

POW is © by Robinson Associates, Inc.

Watcom C/C++ is © by Watcom, Inc.

Page intentionally left blank

## Table of contents

Release: 1.7.6.....	1
1. Introduction.....	14
2. OVERVIEW.....	20
2.1 Concepts.....	20
2.1.1 Context.....	20
2.1.2 The interface between the OS and Amadeus-3.....	23
2.1.3 The interface between the GUI and Amadeus-3.....	23
2.1.4 Overall Amadeus-3 Program structure .....	25
2.1.5 Execution Flow in an Amadeus-3 Application .....	26
2.2 Object Oriented Programming .....	27
2.3 Object Persistence .....	27
2.3.1 Basic Persistent Object.....	28
2.3.2 Internalisation and externalisation of persistent objects .....	31
2.3.3 Generating a new persistent object class.....	33
2.3.4 Object script basics .....	36
2.3.5 Ordering of keywords .....	36
2.3.6 A complete project encoded.....	37
2.3.7 Amadeus-3 standard class names .....	40
2.4 The Data Dictionary.....	42
2.5 Windows and display.....	44
2.5.1 Windows .....	44
2.6 Display objects.....	48
2.6.1 Standard display objects.....	49
2.6.2 Creating your own display objects.....	55
2.7 Guidelines: Advanced Display Object Placement .....	60
2.7.1 Internal representation.....	61
2.7.2 Scroll window columns.....	63
2.7.3 Guideline sample.....	64
2.8 Window layers .....	65
2.8.1 Layer structures and methods .....	66
2.9 Window layouts .....	68
2.10 Message handling.....	69
2.10.1 What messages does your application have to handle?.....	72
2.10.2 Other messages that you may wish to handle occasionally: .....	72
2.10.3 Command codes.....	73
2.10.4 Timer Events .....	74
2.10.5 Power Management.....	74
2.11 Generic sequential data access .....	75
2.12 Scrolling.....	76
2.12.1 Analysis of Module ScrollDemo.....	80
2.12.2 Enhancing Scroll Window Presentation.....	81
2.12.3 Other functions.....	81
2.12.4 Item tagging (selection) .....	81
2.12.5 Example of GetColour procedure .....	82

2.13	Drag & Drop, Amadeus-Style.....	83
2.13.1	How it works for the user.....	83
2.13.2	Other standard mouse operations.....	84
2.13.3	Internal architecture for standard mouse operations and Drag & Drop.....	85
2.14	Databases.....	86
2.14.1	Basic interface.....	86
2.14.2	Major features of the Amadeus-3 database interface.....	87
2.14.3	Basic Database Objects.....	88
2.14.4	About the “Manager” Object Class.....	89
2.14.5	Pervasive.SQL Requirements.....	89
2.14.6	ODBC.....	89
2.14.7	Transactional File Location.....	90
2.14.8	Default database.....	90
2.14.9	Relational table location.....	90
2.14.10	Structure of database information.....	91
2.15	Transactional database programming.....	92
2.15.1	Composite Indexes (segmented keys) explained.....	92
2.15.2	Sample database construction.....	94
2.15.3	Database sample object script.....	95
2.15.4	Code for database access.....	99
2.16	ODBC Data Access.....	101
2.16.1	Sample code for ODBC access.....	102
2.17	Database Editor DbViewer.....	104
2.17.1	Editing features:.....	104
2.17.2	Copying records.....	105
2.18	Reports.....	106
2.18.1	Detailed study of report elements.....	109
2.18.2	Access to data sources.....	117
2.18.3	Direct access to database tables.....	118
2.18.4	DbReport Extensions.....	119
2.18.5	Generic Database Report.....	120
2.18.6	Insertig objects to fill in page.....	121
2.20	Automatic report generation from scroll window.....	123
2.21	Report Debugging.....	123
2.22	Export to stream.....	124
2.23	Code generation.....	125
3.	THE APPLICATION EDITOR A3EDIT.....	127
3.1	User interface.....	128
3.1.1	Command keys.....	128
3.1.2	Mouse actions.....	128
3.1.3	Searching for list items by name.....	128
3.1.4	Shortcuts.....	128
3.1.5	Shortcut handling logic.....	129
3.1.6	Ordering objects in a list.....	129
3.1.7	Object placement and attribute editing.....	130

3.2	Drag & Drop operations.....	132
3.2.1	Special Drag & Drop functions.....	132
3.2.2	Drag & Drop to scroll windows.....	133
3.2.3	Adding objects.....	134
3.2.4	Editing the Data Dictionary.....	135
3.3	Editing decorative objects.....	136
3.3.1	Adding application-specific classes.....	137
3.3.2	Using application-specific classes.....	137
3.3.3	Changing the class of an object.....	138
3.3.4	Viewing and editing window layers and objects.....	139
3.3.5	Arranging window objects.....	140
3.3.6	Editing menus.....	142
3.4	Editing database structures.....	143
3.5	Creating windows based on Templates.....	144
3.6	Starting a new project.....	145
3.7	Starting a new application.....	146
3.8	Loading existing applications for editing.....	146
3.9	Handling Parsing Errors.....	147
3.10	Loading an application by specifying a start-up parameter.....	148
3.11	Saving application data.....	148
3.12	Generating code for object declarations.....	148
3.13	Command line arguments.....	149
3.14	Importing interface elements.....	149
3.14.1	Import Amadeus-2 TX2 files.....	149
3.14.2	Import scroll columns from INI files.....	149
3.14.3	Import CSV Files.....	149
4.	Tutorial.....	150
4.1	Making sure the environment is properly set.....	150
4.2	Hello World !.....	151
4.3	Compiling the “Hello World” application.....	153
4.4	Running the application.....	153
5.	Elements of AMADEUS-3 APPLICATIONS.....	169
5.1	Directory organisation.....	169
5.2	Files used in Amadeus-3 Developments.....	171
5.2.1	The main module "Application.OB2".....	173
5.2.2	The string resource file "Application.INI".....	175
5.2.3	Windows, OS/2: The Standard Resource File Application.RC.....	177
5.2.4	Basic application objects.....	178
5.3	Files to be distributed with end-user applications.....	179
5.3.1	Installation tool.....	180
5.4	Licence Editor.....	181
5.5	Network Application Launcher.....	182
5.5.1	Parameters.....	183
6.	Important Programming Rules.....	184
6.1	Guiding principles in coding for Amadeus-3.....	184
6.1.1	Correctness.....	184

6.1.2	Functionality .....	184
6.1.3	Readability .....	185
6.1.4	Performance .....	185
6.2	Compiler settings .....	185
6.3	Linker settings.....	185
6.4	Coding standards.....	186
6.4.1	Naming convention.....	186
6.4.2	Parameter passing .....	187
6.4.3	Code formatting .....	188
7.	Modules.....	190
7.1	Common.....	192
7.2	MemAlloc .....	192
7.3	MemOps.....	192
7.4	Str.....	193
7.5	Convert.....	193
7.6	StrFmt .....	193
7.6.1	Output Formatting.....	194
7.6.2	Parsing of input strings .....	195
7.6.3	Accelerated translation through pre-compiled templates.....	195
7.7	Expressions .....	196
7.8	NumberAsString .....	197
7.9	Hashing .....	198
7.10	Parsing.....	198
7.11	DateOps.....	199
7.12	TimeOps.....	199
7.13	DateTime.....	199
7.14	Common.....	200
7.15	Resource.....	200
7.15.1	Variable Parameters .....	200
7.15.2	Registry access.....	200
7.16	Commands .....	201
7.17	Persist.....	202
7.18	Values .....	202
7.19	StrVal .....	205
7.20	NumVal.....	205
7.21	RealVal .....	205
7.22	Decimals .....	205
7.23	DecimalVal .....	205
7.24	DateVal .....	205
7.25	TimeVal .....	205
7.26	NumToStr .....	205
7.27	DictTools.....	206
7.28	Projects.....	207
7.29	Startup.....	207
7.30	EndUp .....	207
7.31	PStore.....	207

7.32	Sequence .....	208
7.33	MemList.....	209
7.34	ListView.....	210
7.35	GrpView.....	211
7.36	DbView .....	211
7.37	SQLView .....	211
7.38	Streams.....	212
7.39	MemStream.....	212
7.40	Files.....	212
7.41	Paths.....	212
7.42	Db.....	213
7.43	Btr .....	213
7.44	SQLDb .....	213
7.45	Pointer .....	214
7.46	Cursor.....	214
7.47	Sounds.....	214
7.48	Graphics .....	214
7.49	Metrics .....	214
7.50	Colours.....	214
7.51	Menus.....	214
7.52	Keys .....	214
7.53	Events.....	214
7.54	Fonts.....	215
7.55	CmdCntrl.....	216
7.56	Dialogs .....	216
7.57	WinMgr.....	216
7.58	WinEvent .....	216
7.59	WinTools.....	216
7.60	Scroll.....	216
7.61	ScrollDecor .....	217
7.62	Actions .....	217
7.63	Interactions.....	217
7.64	Controls.....	218
7.65	Fields.....	218
7.66	Toggles.....	218
7.67	Buttons .....	218
7.68	Scrollbar.....	218
7.69	DropList.....	218
7.70	DbSelect.....	219
7.71	SrcSelect .....	219
7.72	TextEdit.....	219
7.73	ValueDsp.....	219
7.74	Bitmaps .....	220
7.75	ImgFiles .....	220
7.76	PCXFiles.....	220
7.77	Icons.....	220
7.78	DrawObj.....	220

7.79	Reports .....	221
7.80	DbReport.....	224
7.81	RptDebug .....	224
7.82	Printing.....	224
7.83	RptTable.....	225
7.84	Debug.....	226
7.85	Licence.....	227
7.86	GenCode .....	227
8.	Version Information.....	228
8.1	Adapting existing code to support window layers .....	228
8.2	TX3 Syntax change.....	229
8.3	Major new features overview.....	230
8.4	Change log .....	232
9.	INSTALLATION.....	270
9.1	Install the compiler first .....	270
9.1.1	Installing Amadeus-3 under Windows 95/98/NT/2000/XP .....	270
9.1.2	Installing Amadeus-3 under DOS and OS/2 .....	270
9.1.3	Installing Amadeus-3 under LINUX.....	270
9.2	Directory structure for a typical development environment .....	271
9.3	Extacy Oberon-2 to C Translator .....	273
9.3.1	SETUP .....	274
10.	Environment.....	276
10.1	Compiling and linking .....	277
10.1.1	Compiling with XDS .....	277
10.2	Sample XDS project file for A3Edit.....	278
10.2.1	Debug mode .....	278
10.3	Optimized code .....	279
10.4	Debugging.....	280
10.5	Working with Multi-Edit .....	281
10.6	Working with Notepad++ .....	281
10.6.1	DESCRIPTION.....	281
10.6.2	FEATURES .....	281
10.6.3	INSTALLATION.....	282
11.	Annexe .....	283
11.1	Applications developed with Amadeus-3 .....	283
11.2	Common File Extensions .....	283
11.3	Object Commands.....	285
11.3.1	Persist.Object .....	285
11.3.2	Persist.Container .....	285
11.3.3	WinMgr.Window .....	285
11.3.4	Values.DictEntry.....	285
11.3.5	Values.Value .....	285
11.3.6	Db.Database .....	285
11.3.7	Db.File .....	285
11.3.8	Sequence.Source .....	286
11.3.9	Sequence.Stepper .....	286

11.3.10	Bitmaps.Object.....	286
11.3.11	GlobalDb.Object.....	286
11.3.12	MemList.List.....	286
11.3.13	Scroll.Mask.....	286
11.4	Typical errors, pending issues and work-arounds.....	287
11.4.1	After compilation, make sure the linker runs successfully.....	287
11.4.2	Make sure the data is correct.....	287
11.4.3	Database operations causing problems.....	287
11.4.4	Window operations – Initial Window Placement doesn't work.....	287
11.4.5	Strange behavior during window updating.....	287
11.5	Oberon-2 vs. C++.....	289
11.5.1	How important is Notation?.....	290
11.5.2	Objective criteria.....	290
11.5.3	Choosing Modula-2 over C.....	291
11.5.4	Choosing Oberon-2 over C++.....	291
11.5.5	Oberon-2 is MUCH MORE THAN an advanced Pascal.....	292
11.5.6	Oberon-2 is NOT in the same family as Ada or Eiffel.....	293
11.5.7	Constraints, Safe code and Programmer Freedom.....	293
11.5.8	Features, Features.....	294
11.6	Description of an exploit against Microsoft Outlook.....	294
11.6.1	Major flaws of C++.....	296
11.7	Using Amadeus Code with Terminal Services / Citrix.....	304
11.8	Table of Figures.....	305

## Foreword

Programming is still one of the most challenging tasks in the modern world. For probably 30 years now, the software crisis rages on. In 1995, the magazine "Scientific America" published an article on the subject of Software engineering, that concluded that after over 40 years of development, very little progress had been achieved.

This bleak picture needs to be corrected. Immense progress has been made in the science of programming and the productivity of the best has greatly improved. I recommend the books by Gerald Weinberg – in particular “Understanding the Professional Programmer” – who dwells on this very issue. Unfortunately, not much of the achievements of the brightest minds have been transferred to the world of application programming. Often out of consideration for the preservation of existing investments on one hand, on the encroachment of de-facto "standards" on the other hand, some counter-productive tools and ideas stay around much longer than they should.

Preserving investment is a very sound objective and should be encouraged, but not to the point where the “Year 2000 Bug” could force huge investments just to repair some bad programming. Great development systems are sold by various companies to improve programmer productivity, but they mostly attempt to overcome the problems created by the use of inappropriate programming languages and operating systems. If carpenters used bad hammers, they too would make their job harder than necessary.

One of the common features that can be observed in many modern development systems is the use of code generators that are supposed to relieve the programmer from the tedious task of having to write code. It clearly shows two things: that the designers didn't find a good way to reduce the amount of code required on the application side by designing better frameworks and that programmers are more and more afraid to use their programming language! This is certainly wrong, a programmer should always feel very comfortable writing code and even more so reading it, as this fills about 80% of a programmer's professional life. Not to mention that code is the most expressive language to convey information on algorithms and can be a powerful documentation tool.

No matter how fervently many will defend their choice of one of the popular programming languages – almost as if it was a matter of religion, not reason – the facts are there to confirm that despite a huge number of published books on C / C++, Visual Basic and Java, the results are meagre. Java took one step in the right direction, towards safer, more efficient programming. So did C#. Both included strict declarations and garbage collectors and several other elements of safe programming. But why not go the whole way, if there is a language that has all those features and more? Especially if it is easy to learn.

From 1986 to 1992, Prof. Niklaus Wirth and his colleagues from the ETH Zürich designed a great programming language, Oberon-2, as the direct successor of Pascal and Modula-2. Oberon-2 combines simplicity and expressiveness in a really unique way. Several books and articles about it have been published and it has found a large following in the academic world, but only a very limited acceptance in practical application development work. This is due to several factors, including its original integration with the operating system Oberon, which will never be used outside an academic or private setting and the absence of viable tools to make it suitable for commercial software developments.

Fortunately, Oberon-2 compilers for various standard platforms became quickly available, which made it possible to create **Amadeus-3**, with the following goals in mind:

- to make the academic language Oberon-2 viable for commercial developments by extracting it from its own operating system context
- to provide an efficient and consistent object oriented interface with the underlying operating systems in a platform independent way
- to supply an extensive application framework with all the necessary functionality for industrial-strength development

These are very ambitious goals, but after successfully using **Amadeus-3** for several years, by various teams for the development of small and large projects in diverse application areas –

including in Fortune 100 companies – the goal of rapid and reliable application development has been met.

### Notations and Abbreviations

Diagrams follow essentially the UML standard. Meta-syntax: where a syntax is explained (such as for commands, a programming language or an object script etc.), the convention is the following:

[ ] Optional element                      { } Optional repeatable

Sections of code and identifiers appear in the following font and colour:

```
PROCEDURE SomeProcedure (): BOOLEAN;
```

Operating System Commands are shown in the following font and colour:

```
COPY file1 file2
```

Internet web, ftp and Email addresses are show in the following font and colour:

```
amadeus-3@amadeus-3.com  
www.excelsior-usa.com
```

Abbreviation	Stands for
<b>OS</b>	Operating System
<b>DOS</b>	Disk Operating System
<b>GUI</b>	Graphical User Interface
<b>API</b>	Application Programming Interface
<b>SDK</b>	Software Development Kit
<b>DLL</b>	Dynamic Link Library
<b>FTP</b>	File Transfer Protocol
<b>OOP</b>	Object Oriented Programming
<b>UML</b>	Unified Modelling Language
<b>ETH</b>	Eidgenoessische Technische Hochschule (Swiss Institute of Technology)
<b>Win32</b>	Windows 95/98/NT

## 1. Introduction

Amadeus-3 is a Application Framework for Oberon-2. It helps with all the common tasks in application development, including, but not limited to, those related to the user interface. It makes full use of Oberon-2's object oriented features, which is absolutely required for developments targeted at the modern graphical user interface systems (GUIs).

Amadeus-3 tries to take some of the initial design decisions off your shoulders. But it does not get in the way. When you want to do things your own way, you can tell Amadeus-3 to step back and let you take control.

Various platforms and compilers will be supported in the future. The current implementation supports Microsoft 32-bit Windows, i.e. Windows 95 / 98, NT, 2000 and XP, with the XDS development system (which already exists for many platforms).

For legacy applications and old computer systems, you can still use a version of Amadeus-3 that works with Windows 3.x. You have to use the Oberon-2 to C converter and then generate the actual machine code with a C compiler, as XDS does not have a native compiler for the 16-bit platform, which is understandable, as it is definitely disappearing fast.

Programs written with Amadeus-3 will appear as native applications, but with some special features that make them quite unique, as you will see.

### A little history

Before the graphical user interface systems, programming was very simple. DOS did so little, that everything was up to the programmer, including the most basic design decisions and system operations, actually anything beyond the file management (just what it pretended to be, a Disk Operating System). This was a blessing and a curse. No interference, no complications, but also no available standard services. A framework such as Amadeus-2 (designed for Modula-2) could take complete control over the machine, along with its own virtual windowing system etc.

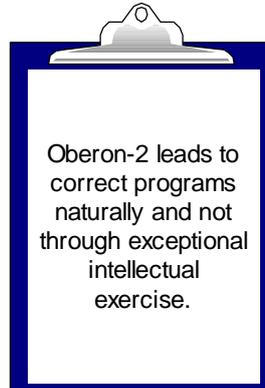
Things didn't stay that simple. Now there are many operating systems with GUIs (Graphical User Interfaces), multitasking, huge APIs (Application Programming Interfaces), system-specific resources etc. To mention just a few: Windows, Mac OS and Rhapsody, X-Windows with incarnations under various workstations and Linux; and on top of X-Windows, Motif, KDE, GNOME and many others.

Of course, all these GUIs supply a lot of functionality: graphics, windows, utilities and so on. But at what price... Just look at the sheer volume of their documentation to grasp the challenge you face when you try to master any one of them, let alone several.

That's where Amadeus-3 comes in. It's an "expert system" that condenses knowledge about GUIs and operating systems and puts it into a more digestible and consistent form, reducing your own learning curve for creating your custom solutions.

## About Oberon-2

Let's point out that without Oberon-2, Amadeus-3 would never have been possible in this form. Oberon-2 is probably the best general-purpose language created to date (as of 1999). It's a small, efficient, logically consistent, language that allows a professional software professional to implement just about any given algorithm with minimum investment of time and resources. You don't have to spend much time learning the language, which is fully defined on only 15 short pages. Instead of learning intricacies of the programming language, a complex syntax and convoluted semantics, you can work on concepts and algorithms right away. The most important feature is that:



Oberon-2 is statically and dynamically<sup>1</sup> safe. That means that once the code compiles, it is likely to be a correct implementation of your algorithms, without any language-related inconsistencies, such as typos accepted as syntactically correct code, incorrect linking due to name clashes, programming errors due to incorrect parameter passing and other problems common in the most popular programming languages. The compiled code will run without memory overwrites or dangling pointer references, which is ensured by the garbage collector and strong compile-time checks. A large number of run-time checks and the use of assert statements will allow you to eliminate any remaining bugs quickly.

Oberon-2 is strongly typed, which means that you are much less likely to cause bugs through the inconsistent use of variables. In addition, it supports the unique feature of open array types, even with multiple dimensions. Contrary to most other languages, upper bounds of all arrays can be found dynamically.

Oberon-2 is fully object oriented. It supports Encapsulation, Inheritance, Polymorphism, Late Binding via methods and Sub-Typing / Sub-Classing (though not as separate concepts). You can also determine the class of an object at run-time.

Oberon-2 is also modular. The module structure allows information hiding and name spaces at no additional cost. By always requiring fully qualified identifiers, any piece of code is immediately comprehensible, without extensive knowledge of the imported modules and without fear of conflicting identifiers. Modules are self-contained and may have an initialisation section, which automatically initialises all required data structures. The initialisation sequence is automatically defined by the compiler and will always be correct, as Oberon-2 does not allow circular imports.

Readability is much better than average for programming languages. Being case-sensitive, Oberon-2 identifiers and keywords always look the same (something suspiciously absent for Ada). There is very little variability in code presentation between programmers, as the syntax doesn't invite many different coding styles. Given that reading code consumes 80 to 90% of any programming effort, it doesn't matter how long it takes to write code<sup>2</sup>, it only matters how quickly it can be read and understood.

---

<sup>1</sup> if used with a garbage collector

<sup>2</sup> given the various syntax-aware editors that automatically write complete code structures at the push of a button, the speed of typing hasn't been an issue for a long time

Your analysis of the problem and the algorithms you use obviously still depend on you alone. They will only be as good or bad as you make them. But Oberon-2 allows you to concentrate on the real issues, instead of having to waste time with “traps and pitfalls” inherent in most programming languages<sup>3</sup>.

### **What about code reuse?**

Oberon-2 was designed from the ground up for code reuse. Given its readability, modularity and object orientation, there is no doubt that it makes it very easy to write code that will be reused often and at little cost.

Now if you think of the reuse of legacy code, programs and libraries written for other languages, such as C/C++ and maybe Assembler, Pascal, Modula-2, Fortran or some other old language; or commercial libraries written for one of the popular programming languages: don't worry, as long as whatever code you wish to use conforms to the platform standard for object modules and/or libraries (including dynamic link libraries, DLLs under Windows), you can use it with Oberon-2.

In fact, as long as the Oberon-2 compiler supports interfacing with external libraries (all of those outside of the ETH Oberon System do), it is usually enough to translate the standard .H header files, which are usually supplied with commercial libraries, into an acceptable format. The XDS and Extacy compilers accept Modula-2 style definition modules. These can be generated automatically from .H header files, for example by using the utility program H2D.EXE supplied at no cost by XDS (cf. corresponding documentation, or download it from their web site – [www.xds.ru](http://www.xds.ru) - if you don't use their compiler).

Unfortunately, this possibility is largely ignored by many programmers. Even many computer magazines perpetuate the assumption that a program that wants to use an API written in C must also use C. According to this logic, all programs would still have to be written in machine language, as all programming languages must be translated into – or must interface with – machine language at some point, which is of course an absurd conclusion.

So there is no relationship between the “system implementation language”, the most popular programming language and what you can or should use. It is simply required that whatever language you use conforms to your own goals, tasks, preferences and style. It should also be able to interface with 3rd-party libraries, if you expect to work on large software projects.

### **How does Oberon-2 compare to other programming languages**

There are many valuable, well designed programming languages in use throughout the world of computer science and practical software development. Some programming languages are very specialised for some application area, others are written with a specific concept in mind, such as functional, real-time or parallel programming.

While there may still be reason to write Assembler code in a few limited situations, such as for brand-new or high-performance hardware, for which no good high-level language compiler exists (e.g. for graphics chip sets), any self-respecting software engineer should attempt to select the best possible high-level programming language that he can find for his needs. Except for those who work exclusively in one specialised domain, it will not make much sense to choose Prolog, APL or Smalltalk for their everyday programming needs. Those languages are so specific, require a very particular mind-set and have a steep learning curve, that there will always be only a handful of expert programmers available. Training a newcomer in one of these is an expensive proposition. Last but not least, it is not necessarily so easy to interface them to standard software libraries. Making them talk to the outside world usually also breaks their internal consistency, by forcing the user to live with coexisting different paradigms.

For the majority of software engineers, the choice will be one of the “classic” procedural or object-oriented programming languages. In this group, we currently find (in approximate order of age): Pascal, C, Basic, Ada, Modula-2, C++, Eiffel, Oberon-2 and Java. Of course, all of the older ones have seen a lot of evolution, sometimes in the guise of a particular implementation – as with Pascal

---

<sup>3</sup> for some languages, entire books have been dedicated to this subject, in particular C/C++

in Delphi and Basic in Visual Basic – changing to the point where there is not much left over of the original language, except the paradigm and basic syntax. The common point is that they are all procedural programming languages, most of them with object oriented elements; even if they are fully object oriented, they still use procedural code to express algorithms, which distinguishes them from Smalltalk, where everything is an object and simple things can be very hard to express, not to mention Prolog.

C/C++, Visual Basic and even Delphi have enjoyed great popularity. Modula-2 and Ada have been used extensively, but in more restricted areas, such as in high-risk environments as nuclear power plants, NASA and aeroplane guidance systems. Given that Ada was originally developed for the US Department of Defence (DoD), it comes as no surprise that it should be used in military and space applications. Despite some major shortcomings, such as an excessive complexity and few quality compilers due to this complexity, it is possible to write solid and reliable code in Ada.

Eiffel is also a well-designed language, but again rather too large and complex. Just like Ada, it suffers from the kitchen-sink syndrome: “Everything but the kitchen-sink”. Let me still recommend the book “Object Oriented Software Construction” by the creator of Eiffel, Bertrand Meyer. It’s just like the language: good, but a bit too large. Some of Meyer’s insights are very valuable, in particular the principal of “design by contract”.

A lot is made of the greater availability of C/C++ programmers. The thinking goes: “C/C++ is so popular and everybody is learning it, it must be easier to find a competent C/C++ programmer for hire”. After 10 years of C++ hype, I have found this to be totally untrue. There are many people who have had some training in C/C++, but very few who have reached a high level of competency, given the difficulties and discipline that is required for learning and using it. According to the director of a large US IT company, the average period to train a C++ programmer is at least 3 years, “which is kind of embarrassing”, he added “as NASA only takes 18 months to train an astronaut”.

My personal experience with Oberon-2 is that the average programmer (or even non-programmer, say an engineer or mathematician) will learn the language in 5 - 10 days and will be able to write quite advanced GUI applications after 15 – 20 days. After 6 months, the newly trained Oberon-2 / Amadeus programmer will be able to produce professional code and advanced object-oriented applications. He will be able to dedicate most of his time and attention to learn what is really important for programming – algorithms, program design, methodologies etc. – rather than to the programming language and the basic tools and libraries.

Hence it will be much easier – and probably cheaper – to train new programmers in a simple, consistent, well designed language and environment, rather than to go hunting for the rare professional and affordable C++ programmer. And you still get to use (or reuse) all the standard programming libraries that are available on the market. Obviously, you can also re-train a C++ programmer to use Oberon-2 instead. If he’s willing, it will be excessively simple for him. The problem is that hardly any C/C++ programmers ever admit that their huge investment in learning the language was useless, a fact that’s even been recognised in *the C* programmer’s favourite magazine, “Dr.Dobb’s”.

### Services provided by Amadeus-3

Amadeus-3 supplies the following services in an OS-independent form. Some of these are wrappers around the corresponding OS/GUI specific elements, others are implemented 100% by Amadeus-3, based on nothing more than basic graphic/font elements.

- Graphical interface design tool
- Database editing utility
- Persistent objects
- Graphic device interface services with high-level resource management
- Font support
- Window and display object management
- Scrolling through any type of sequential data structures
- Intelligent child window management, including window drag&drop operations
- Layered display objects
- Cursor and Pointer manipulation
- Menus
- Event handling
- Command code management
- Dynamic moving and resizing of objects
- Simplified drag & drop support
- User Interface Objects (data entry fields, multi-line fields, buttons, drop-down lists, check boxes, scrollbars etc.), most of them GUI-independent
- Bitmaps and Icons (with installable bitmap loaders for extended bitmap format support)
- Resource files
- Installable termination procedures
- Stream handling, with one-way, positionable and buffered streams
- Standard file access as special case of positionable streams
- High-level file name and directory manipulation
- Data dictionary object
- Data type support and conversions (String, Date, Time, Numbers, Reals, Decimals etc.)
- Extensive string manipulation and conversion functions
- Generic database interface with automatic consistency checking and structure updating
- Btrieve / Pervasive.SQL support as standard instantiation of generic database interface
- ODBC Interface
- Code generation for interface variables and objects
- Projects for the management of groups of related objects and variables as units
- Generic list structure
- Generic hash table class
- Basic parsing support for free-form syntax
- Abstract sequential data source, with instantiations for lists, groups and database files
- Printer support
- Reporting language for the creation of complex documents on screen or paper
- ... many more ...

## Memory management

The only version of Amadeus-3 that does not yet support automatic garbage collection is the one with the Extacy compiler for Windows 3.x. Dynamic memory management had to be added to Amadeus-3 on this platform, too, as neither Windows nor the Extacy compiler (written for DOS) supplied this functionality.

Other platforms have their own memory management system, including garbage collection, which *should* always be present with Oberon-2. To be able to fully rely on garbage collection in a non-Oberon GUI environment<sup>4</sup>, object finalisation is required. XDS version 2.x and up provides this on all platforms. For other compilers, check corresponding notes.

## Interfacing with external libraries

In most cases, Amadeus-3 does more - much more - than just re-define the interface to the operating system or commercial libraries. A complete high-level and object-oriented wrapper is added, that not only shields the application from the underlying library, but also provides intelligence of its own.

You can interface with most commercial libraries and build wrappers yourself, by basing them on Amadeus-3 and fitting them into the framework. Examples of successful integrations include such diverse tools as Btrieve/Pervasive.SQL, AccuSoft Image Library, Victor Image Library, TX TextControl, HP Instrument Control Library and - last but not least - MS Windows itself.

The usual approach to including external libraries is to first convert the .H header files into Modula-2 style .DEF modules with a utility such as H2D.EXE (from XDS Inc. for free), then to ensure that the generated code actually compiles and finally to write a wrapper module, that makes the external library's functions available at a higher level of abstraction and integration into the framework. A good example is the ImgLib.OB2 module, which interfaces to graphics libraries that allow the loading and saving of various graphics file formats.

---

<sup>4</sup> In fact, Object finalisation is essential in any circumstance. For a discussion of the subject, read "Insight ETHOS", cf. Literature Index

## 2. OVERVIEW

In this chapter, a few basic concepts behind Amadeus-3 will be outlined. It is important that you understand these concepts. Amadeus-3 is trying to be very consistent, so that similar things are done in similar ways throughout the system. This should help with the learning process, because you can rely on this consistency. Please refer to the chapter on basic coding standards, which are also very important in this context.

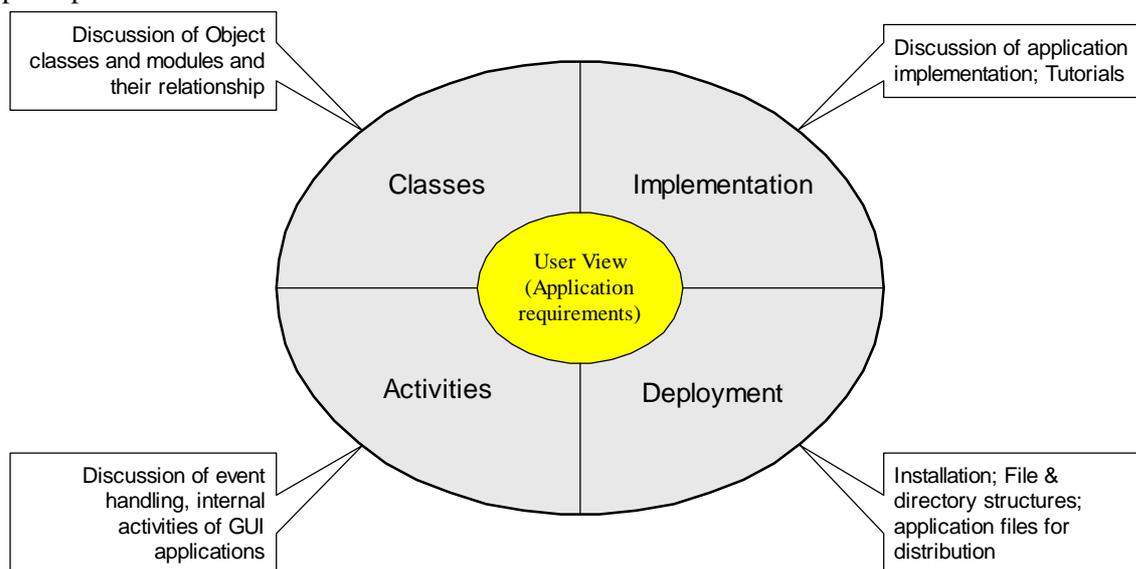
### 2.1 Concepts

Admittedly, the stuff behind Amadeus-3 is fairly complex, when you are not used to working with object-oriented and event driven programming. Although – compared to other products on the market (which are mostly aimed at the C++ or Visual Basic™ community) – Amadeus-3 is child's play, while being every bit as powerful as those other libraries.

The aim of this manual is not to teach you the complete semantics behind every element of the framework. You don't have to learn everything at once to make use of it. Some features you will probably never use, but you do have to understand a few fundamental concepts. Amadeus-3 tries to be very consistent, for the simple reason that it's easier to learn something that is similar to things that you have already learned before. Therefore, concepts are more important and more useful than a long list of features. The features you can always look up. The next few chapters will attempt to give you a broad overview of what you can expect of Amadeus-3 and how you can use it in your everyday projects.

#### 2.1.1 Context

In good UML tradition, let's first establish the context of Amadeus-3 developments and the principals around which this documentation was written:



**Figure 1 - Amadeus-3 Context**

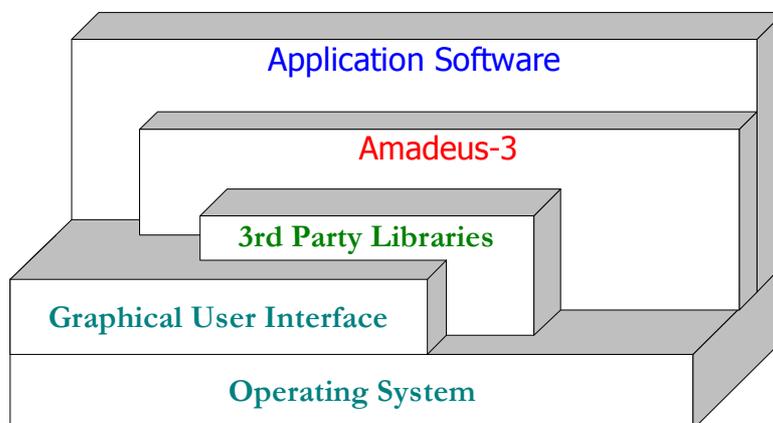
Application programs don't exist in a vacuum. Except where training and pure fun is involved (yes, programming *can* be fun), the first aim in writing a software application is to solve a real-world problem. Obviously, "real-world" may involve a problem strictly related to some computer-specific task. Given that computers have become integral parts of our everyday world, they are no less real than anything else.

The point is that the user of the application wants to solve some problem of information management and we'd like to solve the problem in an economic, efficient and expedient way. The first task is a thorough analysis of the problem and an evaluation of alternatives. This phase is not

what we will discuss in this manual. We are simply concerned with the implementation of the solution that has been selected by the analysis process. We'll assume that the output of the problem analysis is a document that describes use cases, constraints and available resources, as well as specific desires and preferences expressed by the user. Based on such input, we can start writing an application with Amadeus-3.

What is it that Amadeus-3 can do for you? In the first place, it manages the interface with the OS: it provides all the basic nuts and bolts that any application program requires, from string operations and number conversions to file access, directory management and path name parsing; all as far as possible in a OS independent manner. Then it provides a way to communicate with the GUI, which is the more complex part. It does so via a standardised access to basic drawing functions, font and window management and of course event handling.

The best part is that it doesn't just supply wrappers for GUI and OS functions, but that it provides *smart, object oriented* wrappers, which provide a lot of additional functionality. Beyond of these wrappers, Amadeus-3 also provides totally GUI- and OS-independent classes (on top of which a lot of the above mentioned are based). An example of such classes are the Persistent Objects and the concept of the Data Dictionary. They don't require a GUI and have a life of their own. Yet they greatly enhance the GUI.



**Figure 2 - The Layering of OS, GUI, Amadeus-3 and Application**

3<sup>rd</sup> Party Libraries are also shown in this diagram, as they are an important element in any complex software project. Try to do as much as possible with 3<sup>rd</sup> party libraries and tools, as anything you don't have to re-invent and re-code is time gained for your own software.

### 2.1.1.1 What's the OS, what's the GUI?

Obviously, the idea of OS/GUI layering depends on what you think is part of the GUI and what is part of the OS. Microsoft Windows® completely blurred this distinction. Hardly anyone still knows which part of Windows is just the user interface and what is part of the Operating System, which is funny, as the lowest layer of the OS still happens to be MS-DOS, for all but Windows NT. This muddle has been taken to great length, with the legal battle about whether the Internet Explorer is or is not integral part of Windows. The answer is: commercially speaking, anything that Microsoft decides to put into the box (or downloaded software package) is part of Microsoft Windows, that's just a matter of definition. On the other hand, Internet Explorer (or any other Internet Browser) most certainly is *not* a part of any operating system, as an OS that deserves the name is able to run a lot of useful applications without ever requiring the services of a browser. If you just want your computer to run an application that sorts names, all you need is whatever DOS provided.

Fortunately, Linux is here to bring a new awareness of the subject, through multiple user interface variants. You can run one single Linux OS and choose any of at least 5 different GUIs: KDE, GNOME, fvwm, Motif etc. (which in turn run on top of X-Windows, the engine for all the others). And obviously, whatever browser you decide to run on top of any of these GUIs is going to be just another application program.

So let's decide on a simple but sufficient definition for the concept of Operating System:

The Operating System is a collection of software applications and software services that are started right on top of the firmware (software integrated into the computer) and control the execution of any other program running on the same hardware. For Windows 3.x to Windows 95/98, the OS is really MS-DOS and the GUI (Windows) is just an application program. Any decent OS can run without a user interface or just with a command line interface.

Services that are definitely part of the OS, if provided<sup>5</sup>, are:

- Disk management, file system(s)
- IO-device management (keyboard, mouse, communication port(s), display hardware, SCSI etc.)
- Internal clock/calendar control
- Basic multi-tasking
- Network operations at the hardware and protocol level
- User access control (file and network access rights, login at the API level and in text mode)

Services that are part of the GUI:

- Display hardware access in graphical mode
- Basic graphics functions
- Window management
- Input device management in the graphical context (mouse, graphics tablet etc.)
- Multi-media device control
- Printer support

Additional software, that is neither part of the OS, nor of the GUI, include such utilities as:

- Administrative tools (user management, network device configuration, clock setting etc.)
- Internet tools (Browser, Email etc.)
- Multi-media players
- Editors, Paint programs

Is this separation into categories useful? Yes! First of all, a little taxonomy clears up the mind. It helps you to build your own little model of how computers work and then to build your module structure, which may turn out to be much more portable than if you did not break down things in this way. If you buy the Microsoft line that “Internet Explorer” is part of the OS, you will have a bad surprise if you ever have to port your code to OS/2, Linux or another reasonable operating system. Whereas if you consider any “shared modules” of that a browser makes available on any given platform as a 3<sup>rd</sup> party library, you’ll immediately encapsulate it correspondingly and will not suffer too much from having to adapt to a different layering model on another platform (OS & GUI & hardware).

---

<sup>5</sup> obviously, some of these services may also be added through 3<sup>rd</sup> party software, as network services under DOS

### 2.1.2 The interface between the OS and Amadeus-3

Given the above reflections on the role of the OS, the number of truly OS-dependent modules and functions are the following:

- File creation, destruction and access; access control
- Directory creation, destruction and access; access control
- Path name parsing and assembly, at least in as far as the rules are concerned
- Multi-tasking (not yet supported explicitly)
- Networking
- Basic Time / Date services

All of the above may be provided through low-level modules, that are relatively easy to keep platform independent; they may have to be re-written, but they are not very large and the essential services to be provided are very similar from OS to OS.

### 2.1.3 The interface between the GUI and Amadeus-3

The GUI interface is much harder to bring to a high level of portability and platform independence. This is where systems most differ in their internal operations, provided services, look and feel etc.

There are two possible approaches: A platform-independent development framework can try to include everything that any of the supported platforms does, which forces it to re-implement anything that is not supported on all platforms on those that don't have that feature. The other possibility is the minimalist approach, where only a small number of features that is common to all (or almost all) platforms is implemented and supported. If the application programmer needs some more platform-specific functions, he has to side-step the framework and implement it directly via the target platform's specific API.

Just like Java, Amadeus-3 takes the minimalist approach and tries to provide essential services that are common to all target platforms, while hiding the native API to 95% from the application programmer. This means that you don't really learn to program for Windows, but you learn to program for Amadeus-3. You will learn to appreciate this very much, if you ever should feel the need to work directly with the Microsoft Windows SDK.

To show you what we mean, here is an example of what typical MS SDK code looks like. In this sample of message handling code, the application program reacts to the pressing of the left mouse button and decides to perform some operation if the click occurred within a certain screen (window) area:

```
handler (HWND w, LPARAM lparam, MSG msg)
{
    switch (msg) {
        case MSG_LBUTTONDOWN:
            if ((LOWORD(lparam) < WinParam(w, WIN_XCOORD)) &&
                (HIWORD(lparam) < WinParam(w, WIN_YCOORD)) {
                /* handler code */
                break;
            }
        case MSG_LBUTTONUP :
            /* etc. */
            break;
        else {
        }
    }
}
```

It is obvious that the slightest change in the API definition will send code like this where it belongs: to the big bit bucket in never-never land. Just imagine what happens if the programmer mistakenly swaps `LOWORD` and `HIWORD`. This will introduce a fairly subtle and very hard to find bug, all the worse for the fact that the code will occasionally work as expected. Re-reading it won't help, as it's very hard to know which version is actually correct. Seeing this code sample also explains, why the

change from 16 to 32 bit Windows was so painful for most existing software, as each and every instance of message handling code had to be re-written.

**Amadeus-3** provides an elegant way out of this situation by encapsulating the entire interface with the GUI. All messages go through a single handler procedure, which starts by translating all the standard information contained in each message into a more accessible format, i.e. an Oberon-2 record. This happens in the module `Events` and the result is returned as a record of type `Events.Event`.

Next, the message receiver routine `WinEvent.MainWndProc` will proceed to check for messages to be rejected (e.g. mouse clicks outside a modal window or even a group of modal windows). Then, a range of standard messages are analysed by a basic handler, in particular any messages that imply changes to display objects, e.g. when windows are moved or resized. These changes are immediately acted upon and corresponding fields are set within the target object record, so the application never has to watch out for these messages or to use API calls to get the information. It's enough to read the proper fields from the actual display object. A window, for example, contains a sub-record that defines the current placement and extent of the associated window on the screen.

When the basic handler is done, **Amadeus-3** proceeds as follows:

- It tries to find any display object that may be in a position to handle the event, such as buttons, data entry fields or whatever else
- If the event hasn't been handled, the window's own handler method will be called
- Then, the window's dynamically assigned procedure (if any) will be tried
- If a default handler has been defined, this will be called for any event that has still not been flagged as "done"
- Finally the system makes sure that each message has either been acted upon or is passed on to the GUI's own default handler.

The above example which tries to interpret a mouse click based on window and mouse coordinates looks like this under **Amadeus-3**:

```
PROCEDURE Handler (w: WinMgr.Window; VAR ev: Events.Event);
BEGIN
  IF (ev.tp = Events.Mouse) & (ev.action = Events.Press) THEN
    IF (ev.button = Events.LButton) & (ev.px < w.pos.xl) & (ev.py < w.pos.yl)
      THEN
        (* here goes the handler code *)
      END;
    END;
  END Handler;
```

There are some major advantages to this representation, which is technically speaking very close to the C code:

- The code is much more readable; it would immediately strike you as odd if `ev.px` was compared to `w.pos.yl` (x / y coordinate confusion), whereas the C code is totally unintelligible.
- Tests are logically more consistent: you first determine that you want to handle *any mouse message* (and not just the `LBUTTONDOWN`), then that the expected action is a click (or button press) and finally you perform the coordinate-related test
- You don't have to use esoteric methods to extract mouse or window coordinates from the message or an object «handle». This information is properly prepared by the system for consumption by application code.
- The code becomes more secure, as all elements can be type-checked
- Changes to the API are not as sensitive, as only a few things in **Amadeus-3** may require changes; all the dependent application code just needs to be recompiled, in the worst case

### 2.1.4 Overall Amadeus-3 Program structure

The following is an overview of the standard main module functions and imports. These imports are required for any application that wants to provide the default Amadeus-3 functionality.

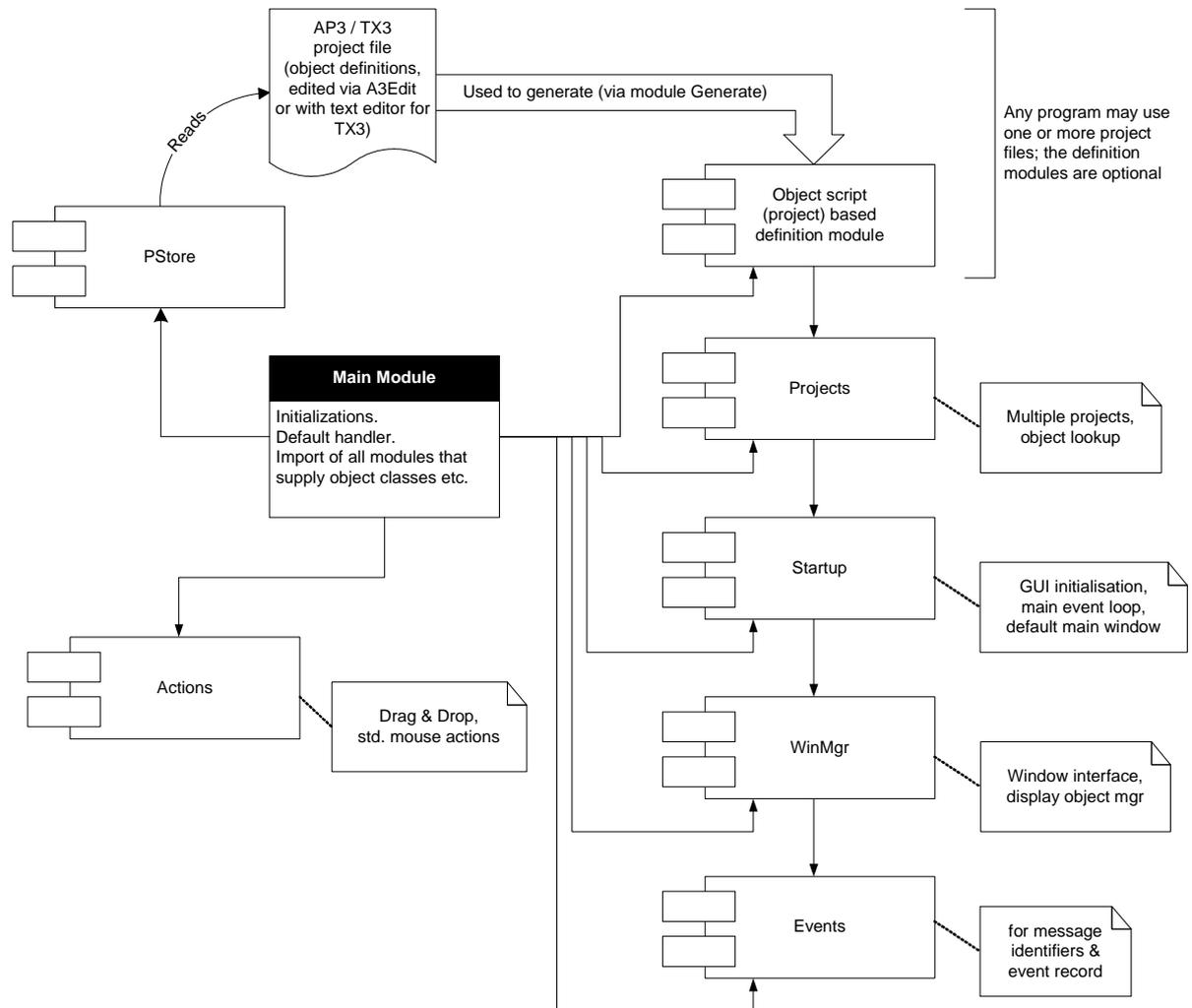


Figure 3 – Overall Amadeus-3 Program Structure

### 2.1.5 Execution Flow in an Amadeus-3 Application

The diagram «

Figure 4 – Execution flow in an Amadeus-3 application » shows some of the things going on inside the system. This is where you can start to relax: the application programmer doesn't really have to worry about all this. Look closely: all you have to do is manage the stuff in the right-most column.

For a more detailed discussion of the event handling process, please read the chapter « 2.8 Message handling ».

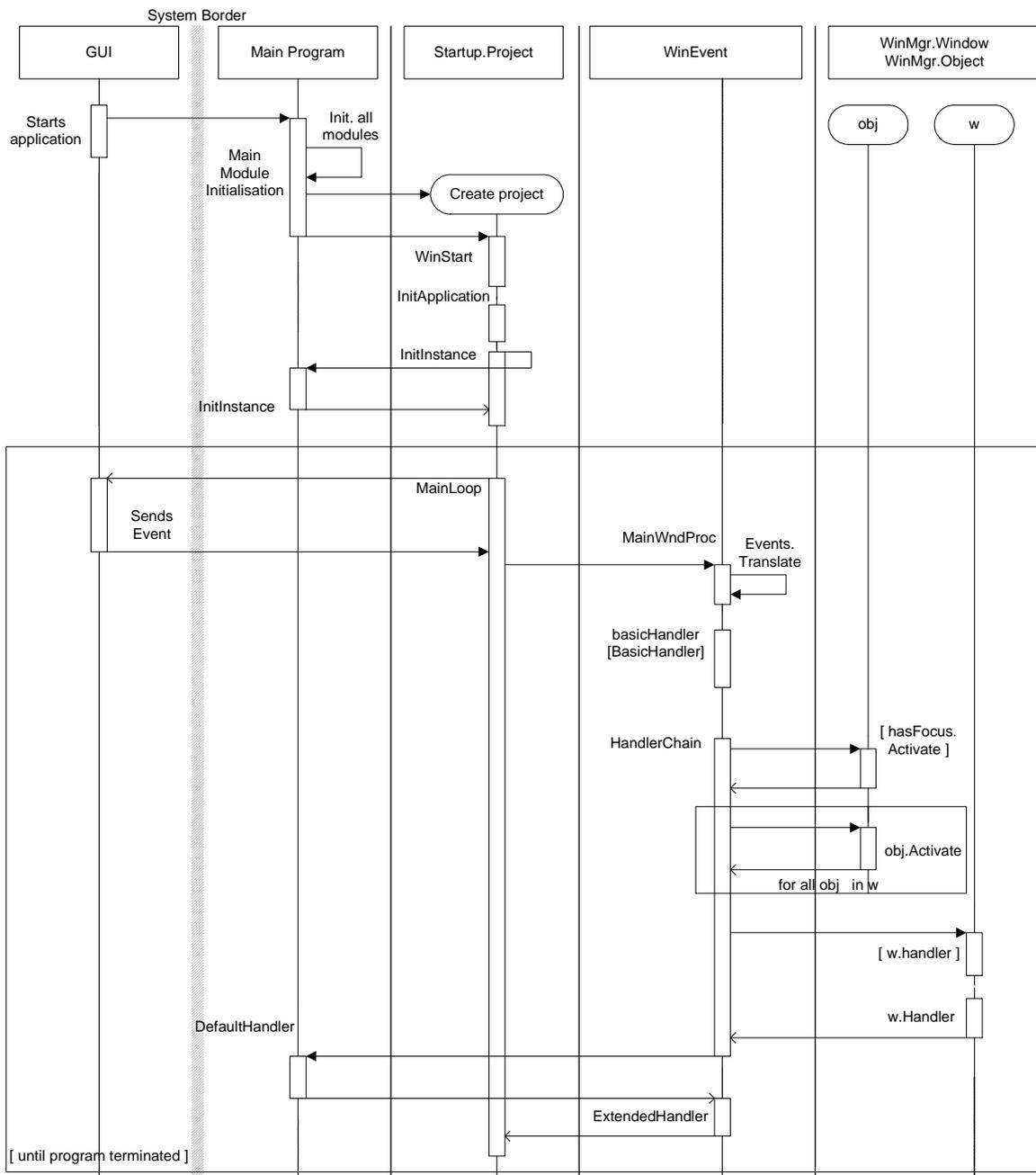


Figure 4 – Execution flow in an Amadeus-3 application

## 2.2 Object Oriented Programming

This documentation is not a detailed introduction to Object Oriented Programming (or OOP). If you are not already familiar with OOP concepts, you are invited to read one of the books listed in the bibliographic annex, in particular

*"Object Oriented Programming using Oberon-2"* by Prof. Moessenboeck, edited by Springer.

In this chapter, you will find a discussion of OOP design and techniques as used in Amadeus-3.

Typical objects used in Amadeus-3 are Windows, display objects, data dictionary elements, streams, files etc. They have one thing in common, they are all derived from the basic persistent object class `Persist.ObjectDesc`. Therefore, we'll have a closer look at the concept of persistent objects:

## 2.3 Object Persistence

Most objects appearing in the Amadeus-3 library are derived from the base class `Persist.ObjectDesc`, which supports object persistence, along with some other convenient features. What does "persistent" mean?

Objects of this class support encoding and decoding methods, which makes it possible to write a complete description of these objects to a data stream (e.g. a text or binary file, cf. the modules `Streams` and `Files`) and to recreate them by reading and decoding the same data stream.

To say it in other words, you can **save** and **load** complete objects to and from disk files (or any other external storage), giving the objects *persistence* between various executions of a program or even exchanging the object between different programs and computers. The lifetime of the object can thus be extended to more than just the execution time of a given program on a specific machine. When you load the object on another computer, it's state and attributes can be completely restored.

The encoding and decoding methods support text as well as binary streams:

- The advantage of the text format is that you can edit and modify the corresponding object script with a text editor or print the complete object definition on paper for documentation or transmission purposes.
- The advantage of binary encoded objects is that they are – marginally – faster to read and write and that they are more temper-proof, if you don't want someone who has no business doing so to modify the object definition simply by using a text editor.

Please note that each and every application can decide to read or encode objects in any of the two formats whenever desired. This has a major impact on what you can do with objects under Amadeus-3.

Example: you could prompt the user to enter object attributes as text, using the user's input to generate dynamic objects, set attributes for existing objects etc. You can do a lot of the things that would require a major development effort almost for free. It also becomes extremely easy to implement "macro" languages for your applications, as an important element – textual representation of objects – is already included in the basic system.

Obviously, object definitions may also be stored in a database in a very convenient way. The possibilities are endless. Which is why it's important that you fully understand the concept of persistent objects and their implementation under Amadeus-3.

### 2.3.1 Basic Persistent Object

Module Persist defines several classes that are related to the management of persistent objects in some way. Here is the basic declaration for the basic persistent object, with its methods:

```
ObjectDesc* = RECORD
  name-      : Str.PStr;
  note*      : Str.PStr;
  extClassId*: INTEGER;
  noFind-    : SHORTINT;
  finalize-  : BOOLEAN;
  PROCEDURE (po: Object) Init*;
  PROCEDURE (po: Object) Dispose*;
  PROCEDURE (po: Object) Finalize*;
  PROCEDURE (po: Object) ClassName*(VAR s: ARRAY OF CHAR; parent: INTEGER);
  PROCEDURE (po: Object) Name*(VAR s: ARRAY OF CHAR);
  PROCEDURE (po: Object) ClassId*(VAR id: INTEGER; test: BOOLEAN);
  PROCEDURE (po: Object) SetName*(s: ARRAY OF CHAR);
  PROCEDURE (po: Object) SetFindMode*(active: BOOLEAN);
  PROCEDURE (po: Object) Find*(VAR name: ARRAY OF CHAR; cid: INTEGER;
                               VAR owner, res: Object): BOOLEAN;

  PROCEDURE (po: Object) FindObj*(obj: Object; VAR owner: Object;
                                   VAR pl: Link): BOOLEAN;
  PROCEDURE (po: Object) EncodeStart*(ts: TokenStream; withObjTk: BOOLEAN);
  PROCEDURE (po: Object) Encode*(ts: TokenStream);
  PROCEDURE (po: Object) EncodeClose*(ts: TokenStream);
  PROCEDURE (po: Object) DecodeClose*(ts: TokenStream);
  PROCEDURE (po: Object) Decode*(ts: TokenStream);
  PROCEDURE (po: Object) CopyFrom*(src: Object; deep: INTEGER);
  PROCEDURE (po: Object) Declare*(VAR module, type, suffix: ARRAY OF CHAR);
  PROCEDURE (po: Object) Command*(cmd: ARRAY OF CHAR;
                                   VAR res: LONGINT; VAR answer: ARRAY OF CHAR):
    BOOLEAN;
  PROCEDURE (po: Object) SubstRef*(old, new: Object; VAR found: INTEGER);
  PROCEDURE (po: Object) Owns*(target: Object; VAR owner: Object): BOOLEAN;
END;
```

- Attribute 'name':  
The name field contains the local name of the object. The method Name, on the other hand, returns the extended name. A typical example would be an object representing a variable, based on the class Values.Value. If this variable is a member of a record structure (Values.Struct), then the local name is "fieldName" and the extend name is "recordName.fieldName". The method SetName will take care of setting the name correctly.
- Attribute 'note':  
The note field contains any sort of string, which may be interpreted as a simple comment, a note to be displayed as help message (for display objects) or a help reference. Further meanings may be assigned.
- Method Init and Dispose:  
These methods are the basic "Constructor" and "Destructor" methods of Amadeus-3 persistent objects. Init must fully initialize the object, placing it in a known state; Dispose should release any external resources held by the object, e.g. close open files etc.
- Method Finalize:  
Installs a finalizer procedure for a specific object instance. Only one finalizer will be installed for each object, even if you call Finalize several times (e.g. in sub-class initialization). The finalizer procedure will simply call the Dispose method of the object, which must de-allocate any external resources of the object. Only needs to be installed for objects that contain resources that are

external to the application's own memory space, such as GUI objects (window handles, bitmap data etc.).

- **Method ClassName and ClassId:**  
Return the class name or Id for a given object instance. You may also check if a class is a parent class of another by passing the target classe's ID and setting test = TRUE. If the given ID identifies a parent class of the tested class, it will return the object's own class ID, otherwise it will return Persist.NotRelated.
- **Methods Find and FindObj:**  
These methods allow you to look for a specific object by name or by it's pointer within another persistent object. Some objects may indeed contain references to other persistent objects, as would be the case with groups, but also others, as we will see.
- **Method CopyFrom:**  
May be used to copy attributes from another persistent object. May perform either a shallow or a deep copy, to the desired number of levels, if supported by the actual class.implementation.  
NB: May be used on objects from different classes, but obviously only attributes that are compatible will be copied. Typically, you may copy attributes from an object of a super-class to an object of a sub-class.
- **Method SetFindMode:**  
Allows to turn Find and FindObj on or off, layered, i.e. you may call SetFindMode (FALSE) several times and then need to call SetFindMode (TRUE) as many times to re-activate searching. Actually increments and decrements the attribute field 'noFind', which, when > 0, makes the object "invisible" for the Find and FindObj methods.
- **Methods EncodeStart, Encode, EncodeClose, Decode, DecodeClose :**  
Used to encode and decode persistent objects, cf. chapter "2.3.2 Internalisation and externalisation of persistent objects".
- **Method Command :**  
Very powerful method, allowing you to send commands to generic objects, e.g. within report scripts. The Command method will reply with TRUE / FALSE, a result code and a string answer, where appropriate. There are not many limits to what you can do with this.

### 2.3.1.1 Persistent Object Class

The first support class is a meta-class named "ObjectClass", that contains information on persistent object classes that are dynamically defined. Via this meta-class you can create instances of objects of that class by calling the attached 'make' procedure.

```
MakeProc*           = PROCEDURE (VAR Object);
ObjectClassDesc*   = RECORD (ObjectDesc)
  id*               : INTEGER;
  base*             : ObjectClass;
  make*             : MakeProc;
  owner*            : Object;
  module*           : Str.PStr;
  type*             : Str.PStr;
  suffix*           : Str.PStr;
END;
```

But why use such an indirect scheme to do what you could do directly?

- To be able to create an object by class name or class id

Why not use standard library functions for creating objects via the objects module and class name, without building a secondary class description?

- To allow for dynamic class-substitution; by replacing the 'make' procedure, an application can create instances of it's own implementation of a particular class, instead of the class that is associated with a particular class name or id. When loading object scripts, it is very handy to do

this, if you wish to re-define some aspects of – for example – the window class for the entire application.

- It also allows for dynamic substitution; the object script may specify a sub-class that is not known to the application that is loading the script, as is often the case with the application editor A3Edit. Yet you want to preserve the information defining the sub-class for applications that actually do know it. The Amadeus-3 object creation mechanism is sufficiently dynamic to support exactly this mechanism through composite class names, i.e. a string composed of “extended class name | base class name“. This is why each ObjectClass defines it’s base class.
- The attributes module, type and suffix definitions are mostly useful when generating code for a new module, such as is done in A3Edit.
- The owner attribute is used to link the class to an owner object, such as a project (Projects.Project).

### 2.3.1.2 Persistent Object Group

Another important class directly derived from and extending persistent objects is the `Group`, which in turn requires the `Link` class. Groups are persistent objects that may contain a list of other persistent objects, which obviously is a recursive definition and allows for recursive data structures.

```
Link*           = POINTER TO LinkDesc;
LinkDesc*      = RECORD
  next*: Link;
  po*   : Object;
END;
GroupDesc*     = RECORD (ObjectDesc)
  tbl*: TraceTbl;
  pl*  : Link;
  PROCEDURE (gr: Group) RemoveLink*(rem: Link): BOOLEAN;
  PROCEDURE (gr: Group) Remove*(po: Object);
  PROCEDURE (gr: Group) Add*(idx: INTEGER; po: Object; VAR ins: Link): BOOLEAN;
  PROCEDURE (gr: Group) Insert*(po,before: Object; VAR ins: Link): BOOLEAN;
  PROCEDURE (gr: Group) SetTraceTbl*(n: INTEGER);
  PROCEDURE (gr: Group) Delete*;
END;
```

The methods are fairly explicitly named:

- Method Add adds a link to the specified object in the group at a location specified as index number, i.e. before the element found at location number idx.
- Insert does the same, but specifies the location as the position between between 2 other objects.
- Remove removes any link to the specified object in the group, recursively
- RemoveLink removes a specific link from the group, recursively
- Delete kills all elements of the group by calling their Dispose function, recursively. Naturally, the actual objects will only be deleted when all other references to them have been destroyed.
- NB: When you call the Dispose function, the objects contained in the group will not be destroyed, merely the group pointers will be reset.

### 2.3.1.3 Persistent Object Containers

Containers are persistent objects themselves, containing a pointer to another persistent object. This allows a double-indirection, which can be very powerful. A typical use is demonstrated in the module DbReport, where the report script is able to refer to named objects and the objects they refer to, which may be assigned during the execution of a report script. The main particularity of containers is that any command that they receive and don’t understand themselves is passed on to the contained object.

### 2.3.2 Internalisation and externalisation of persistent objects

Note: In the Object Oriented Literature, you will find the terms Internalisation and Externalisation. The internalisation method for Amadeus-3 persistent objects is called [Decode](#); the externalisation method is [Encode](#).

Persistent objects may be encoded and decoded from both, text and binary data streams. The syntax for persistent objects is expandable, allowing for new object extensions to be smoothly described.

#### 2.3.2.1 Encoding a persistent object

Encoding of objects is a relatively simple process, as depicted in “Figure 5 - Object encoding”.

Each persistent object class first registers all of its keywords with the parsing service with [Persist.AddMap](#). This is required for encoding and decoding.

Before actually using either of these functions, you will also need to declare, initialise and open a [Parsing.TokenStream](#), which itself is derived from [Parsing.TokenStream](#) and must be associated with another stream, the actual output stream, that may be any stream type, but will usually be a file ([Files.File](#)). Note that this string is buffered (cf. module [Streams](#) for more details).

When this is done, start with a call to the [Persist.Object.EncodeStart](#) method, followed by [Persist.Object.Encode](#) and finally [Persist.Object.EncodeClose](#). The procedure [Persist.EncodeObj](#) will take care of calling all three methods in the right order. The reason for this multi-step calling mechanism is that it gives several layers of inheritance the chance to add their own information at the correct moment during the encoding process. The same sequence will be inverted for decoding.

If encoding to a binary stream, only a token, representing a symbol in an object-class specific keyword map, will be sent to the output stream.

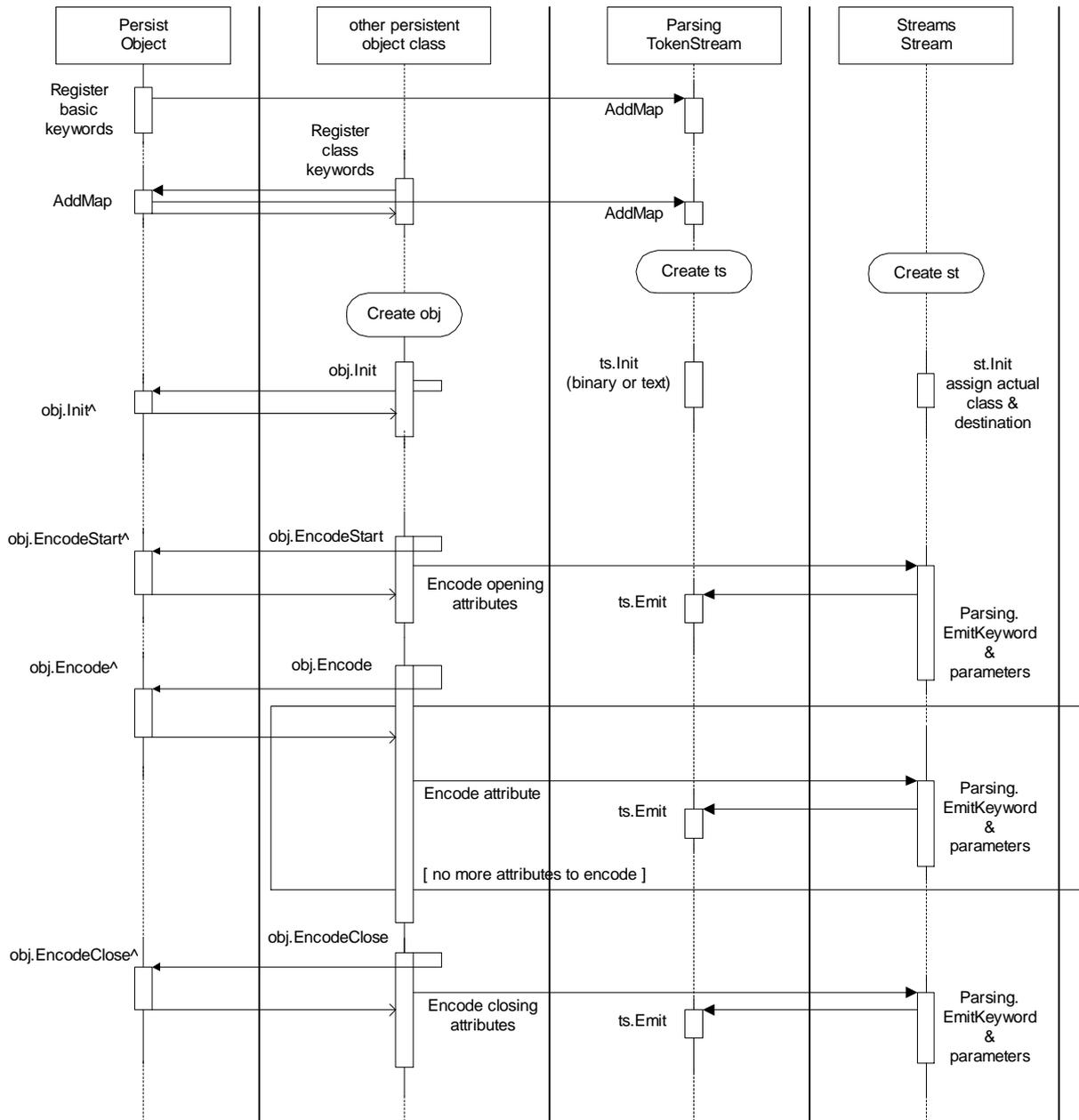
If encoding to a text file, the token will be replaced by the keyword itself. While it is possible to re-use non-standard keywords (not defined by module [Persist](#)), you are advised to use as diverse and specific keywords as possible, if you create your own persistent class. This will avoid potential name clashes and will keep the object script code more readable.

#### 2.3.2.2 Deep Encoding

If you call [Encode](#) for a [Persist.Group](#), the entire contents of the group will be saved. How other objects with embedded persistent objects react is implementation-dependent. You may either encode shallow structures, or you may perform “deep” encoding, where all of the embedded objects are encoded as well, within the data structure. An example of this handling are [WinMgr.Window](#) objects, that contain display objects. The display objects only exist within the window and should therefore be encoded along with the window.

For object references, i.e. pointers to other persistent objects that are not embedded in but outside of an object, you should only encode the reference, i.e. the name of the object or some other unique identification, so that the link may later be re-created. Obviously, reading such references from a token stream could cause major problems, when the referenced object was not re-created before being requested. Amadeus-3 provides mechanisms to overcome these problems. When saving projects, these are first sorted, so that referenced objects appear early in the list, before the references themselves. This also implies that you may not have any circular references, which is not so easy to check beforehand. If you get errors when reading a script, you should check immediately if this is not due to a circular reference.

A typical example of an object reference is the encoding of data entry fields, where the field refers to a variable, declared in a data dictionary of either the same project, or of a parent project.



**Figure 5 - Object encoding**

### 2.3.2.3 Decoding a persistent object

Decoding of an object stream is not quite as simple, but fortunately transparent to the application program. The process is basically the inverse of the encoding process, except that now the program must not only encode but also decode keywords.

Regarding the dynamic loading of object classes: the ETH Oberon system supports dynamic module loading, which is not so readily available on other systems. It could be emulated with DLL libraries, although this is far from equivalent. At this point, there is no such mechanism integrated into **Amadeus-3**, but it will be included at a later date, most likely via DLL libraries under Windows, despite their limitations.

### 2.3.3 Generating a new persistent object class

For a fully persistent object class, you have to add support for all the standard methods, i.e. as `Encode` and `Decode`, `ClassId` and `ClassName`. Others - such as method `Declare` - are optional. The method `CopyFrom` is desirable, as it supplies an easy way to make shallow and deep copies<sup>6</sup> of an object, depending on the “deep” parameter you pass. If deep is zero, the copy will be shallow, i.e. only the object itself and its field values will be copied. If deep is greater than zero, it will also copy attached data structures and objects, decrementing “deep” at every step until it reaches zero.

If the class was properly defined and registered, it will be available for use in object scripts and will behave in every way like a standard persistent object – in this case even as a standard display object.

Let’s have a look at a complete implementation of a persistent object class, the `Icons.Object` class, with the exception of some specific code that does not enhance the general understanding of the persistent object implementation. For the full version of this module, please read the actual source code distributed with Amadeus-3.

```

MODULE Icons;
IMPORT Persist,Projects,Parsing,Values,NumVal,Graphics,Colours,Events,Keys,
    Metrics,Fonts,Str,WinMgr,WinEvent,Win:=Windows,SYSTEM;
CONST
    (* Various constant declarations ... *)
TYPE
    Handle* = Win.HICON; (** GUI icon handle *)
    IconEntry* = RECORD (** Entry into IconTable, cf. below *)
        icn* : Handle; (** Icon handle *)
        name* : Str.PStr; (** Name of this entry in icon table *)
    END;
    IconTable* = POINTER TO ARRAY OF IconEntry; (** Icon table *)
    (** Basic window part icon object class ----- *)
    Object* = POINTER TO ObjectDesc;
    ObjectDesc* = RECORD (WinMgr.ObjectDesc)
        tbl-: IconTable; (** Actual table with icons and names *)
        is* : NumVal.Value; (** Associated value *)
        idx* : SHORTINT; (** Index of value *)
    END;

    (* GUI Icon handling code ... *)

VAR iconId-: INTEGER; (** Icon class ID *)

PROCEDURE (wp: Object) Init* ();
(** initialize data structure *)
BEGIN
    wp.Init^; wp.is := NIL; wp.idx := 0;
    NEW (wp.tbl, 1); wp.tbl [0].name := NIL; wp.tbl [0].icn := WinMgr.Null;
END Init;

    (* GUI Icon handling code ... *)

PROCEDURE (wp: Object) Dispose* ();
(** should be handled by finalizer, if available *)
VAR n: LONGINT;
BEGIN
    IF wp.tbl # NIL THEN
        FOR n := 0 TO LEN (wp.tbl^) - 1 DO
            Dispose (wp.tbl [n].icn); (* release GUI resources *)
        Str.Dispose (wp.tbl [n].name);
        END;
        SYSTEM.DISPOSE (wp.tbl);
    END;
    wp.Dispose^;
END Dispose;

PROCEDURE (wp: Object) SetName* (s: ARRAY OF CHAR);

```

<sup>6</sup> or even deep copies, implementation dependent

```

BEGIN wp.SetName^ (s); wp.AddToTable (s, 0);
END SetName;

PROCEDURE (wp: Object) Encode* (ts: Persist.TokenStream);
(* this is how an object may be encoded, independent of the output mode *)
VAR s: Str.WorkStr; n: LONGINT;
BEGIN
  wp.Encode^ (ts);
  IF wp.is # NIL THEN
wp.is.Name (s);
Parsing.EmitKeyword (ts, iconId, ValueTk); Parsing.EmitString (ts, s, FALSE);
  END;
  Parsing.EmitKeyword (ts, iconId, IndexTk);
  Parsing.EmitNumber (ts, Parsing.ShortInt, 10, wp.idx);
  IF NoReset IN wp.attrib THEN Parsing.EmitKeyword (ts, iconId, NoResetTk) END;
  IF wp.tbl # NIL THEN
    FOR n := 0 TO LEN (wp.tbl^) - 1 DO
      Parsing.EmitKeyword (ts, iconId, PictTk); Str.Get (wp.tbl [n].name, s);
      Parsing.EmitString (ts, s, TRUE);
    END;
  END; (* if tbl *)
  Parsing.EmitEndSection (ts);
END Encode;

PROCEDURE DecodeIcon (ts: Persist.TokenStream; wp: Persist.Object;
VAR ok: BOOLEAN): BOOLEAN;
VAR
  de: Values.DictEntry;
  l : LONGINT;
BEGIN
  WITH wp: Object DO
    CASE ts.cmd OF
      ValueTk : ok := Projects.NextIsDictEntry (ts,de) & (de IS NumVal.Value);
                IF ok THEN wp.is := de (NumVal.Value) END;
      | IndexTk : ok := Parsing.NextIsNumber (ts, Parsing.ShortInt);
                wp.idx := ts.sint;
      | NoResetTk: ok := wp.SetAttribute (NoReset, TRUE);
      | PictTk : ok := Parsing.NextIsString (ts); l := LEN (wp.tbl^);
                wp.AddToTable (ts.str^, SHORT (l));
    ELSE
      ts.again := TRUE; RETURN FALSE;
    END;
    RETURN TRUE;
  END; (* with *)
END DecodeIcon;

PROCEDURE (wp: Object) SubstRef* (old,new: Persist.Object; VAR found: INTEGER);
BEGIN
  wp.SubstRef^ (old, new, found);
  IF wp.is = old THEN INC (found);
    IF old # new THEN
      IF (new # NIL) & (new IS NumVal.Value) THEN wp.is := new (NumVal.Value);
        ELSE wp.is := NIL END;
    END;
  END; (* if wp.is *)
END SubstRef;

PROCEDURE (wp: Object) Declare* (VAR module,type,suffix: ARRAY OF CHAR);
BEGIN COPY ('Icons', module); COPY ('.Object', type); COPY ('Icn', suffix);
END Declare;

PROCEDURE (wp: Object) Decode* (ts: Persist.TokenStream);
BEGIN wp.Decode^ (ts); Persist.DecodeTokens (ts, wp, iconId, DecodeIcon);
END Decode;

PROCEDURE (wp: Object) ClassId* (VAR id: INTEGER; test: BOOLEAN);
BEGIN

```

```

    IF test & (id # iconId) THEN wp.ClassId^ (id, TRUE);
IF id = Persist.NotRelated THEN RETURN END;
    END;
    id := iconId;
END ClassId;

PROCEDURE (wp: Object) ClassName* (VAR s: ARRAY OF CHAR; parent: INTEGER);
BEGIN
IF parent <= 0 THEN COPY ('Icon', s)
ELSE wp.ClassName^ (s, parent - 1) END;
END ClassName;

PROCEDURE (wp: Object) CopyFrom* (src: Persist.Object; deep: INTEGER);
VAR
    s: Str.WorkStr;
    n: LONGINT;
BEGIN
wp.CopyFrom^ (src, deep);
IF src IS Object THEN
    wp.is := src (Object).is; wp.idx := src (Object).idx;
    FOR n := 0 TO LEN (src (Object).tbl^) - 1 DO
        Str.Get (src (Object).tbl [n].name, s); wp.AddToTable (s, SHORT (n));
    END;
END; (* if src *)
END CopyFrom;

PROCEDURE MakeIcon* (VAR po: Persist.Object);
VAR wp: Object;
BEGIN NEW (wp); po := wp;
END MakeIcon;

BEGIN
Persist.Register (MakeIcon, iconId);
Persist.AddMap (iconId, 'VALUE;INDEX;NORESET;PICT');
END Icons.

```

Important points to consider are:

- The class name and class id must be **UNIQUE!** The id will be assigned automatically by the call to `Persist.Register`, which at the same time registers the new object class with **Amadeus-3**
- Make sure the identifier map does not overlap with a parent class. Similar identifiers may occur in unrelated classes.
- The `Encode` method is usually simple to implement, but remember to call `Parsing.EmitEndSection` at the end. Remember to include spacing and newline information, so the ASCII version of the script will look nice.
- Decoding may be tricky. Be flexible and include some error handling code. Remember, scripts may be manually edited, so errors may occur.

So to sum it up:

- Declare the new object class, based on `Persist.Object` (or some derived class).
- Define a variable to hold the class id.
- Define the following required classes: `ClassId`, `ClassName`
- Define the procedure `Make"ClassName"` (here: `MakeIcon`), which simply creates an object of the new class and returns a pointer of type `Persist.Object` to it.
- Call `Persist.Register` with the “make” procedure and the class id.
- For persistence support, define constants for all required tokens and add the following call in the module initialisation section: `Persist.AddMap`, `Encode`, `Decode`
- Optionally define the following methods: `Init`, `Dispose`, `Declare`, `CopyFrom`

### 2.3.4 Object script basics

Each object starts with the reserved word "OBJ", which is followed by the objects class name (required), it's individual name in quotes (optional), the reserved word "ID" followed by the object's id number (optional and each object must be terminated with the reserved word "ENDOBJ").

Symbolically:

```
OBJ className [ "name" ] [ ID number ]
  { object specific attributes }
  [ NOTE "text" ]
ENDOBJ
```

The simplest acceptable object definition would therefore look like this:

```
OBJ Brush ENDOBJ
```

which just defines an object without any attributes. Further object-class specific attributes are added between the standard object header and the "ENDOBJ" keyword. Each object class may add it's own keywords. Unrelated classes (i.e. which do not inherit from each other) may even define the same keywords without causing name clashes.

To take a more concrete example:

```
OBJ Brush "dataEntry" 0xFFA10F ENDOBJ
```

defines an object of class `Brush`, as implemented by the module `Colours` (cf. that module for further information). Instead of adding new keywords, this object just assumes that the first string after the object header is an RGB value. Additional keywords may follow to define hatching attributes.

The `NOTE` section may be included with any object for documentation purposes. Display objects display their `NOTE` text, when the mouse cursor is placed over them for a brief period of time and if this feature is activated. Please refer to the module `WinEvent` for further details.

### 2.3.5 Ordering of keywords

Most objects accept keywords in any order, as long as all keywords belonging to the same class are grouped together. Example: An object of class `Toggles.ToggleDesc` is a checkbox associated with a `BOOLEAN` value. But `Toggles.ToggleDesc` descends from `Fields.FieldDesc`, which defines standard data entry entities. Therefore, `Toggles.Toggle` may add new keywords for it's own use, but all these keywords must follow any keyword defined by `Fields.FieldDesc`.

#### WRONG ordering

```
OBJ Toggle "Answer ok" ID 1
  ACCESS LEFT VALUE answerOk
ENDOBJ
```

While the term `ACCESS` is properly placed (defined by `WinMgr.ObjectDesc`), `VALUE answerOk` (defined by `Fields.FieldDesc`) should precede the term `LEFT` (defined by `Toggles.ToggleDesc`).

#### CORRECT ordering

```
OBJ Toggle "Answer ok" ID 1
  ACCESS VALUE answerOk LEFT
ENDOBJ
```

Occasionally, an object class may change this ordering requirement by inserting a group of keywords *before* those of it's base class, e.g. `Values.DataDict`. These instances will be clearly signalled.

Usually, you will not have to worry about these things, as objects know very well how to encode themselves. But you must watch out for this when you make manual changes to a script. It is indeed very unlikely that you should wish to write a full script from scratch.

### 2.3.6 A complete project encoded

Now let's have a look at a typical object script, in this case a project with elements you will find in most applications. This script was generated using the A3Edit application editor. Annotations are preceded by a "--" double dash.

```
-- The entry OBJ Prj defines the main entity, a project object
-- as defined by Projects.ProjectDesc:
OBJ Prj "Sample"
-- This object script has no parent script, so it specifies:
  REQUIRES NONE
-- You may define application-specific classes, with their name
-- and base class; these classes must be defined within the application
-- if objects that you declare to be of such a class should exhibit
-- the appropriate behavior
  CLASS TodoSrc DbView "" "" "" ENDOBJ
-- A project may define a table with command codes, cf. module Commands:
  OBJ Cmd Ok 1   Cancel 2   Abort 3   Retry 4   Ignore 5   Yes 6   No 7
    About 100   HelpIndex 101   HelpOnFocus 102   HelpOnHelp 103
    Exit 110   AddBook 1001   EditBook 1002   DelBook 1003
  ENDOBJ
-- A data dictionary may also be included in a project object definition
  OBJ DataDict "dataDict"
-- Data dictionaries may define a constant table, terminated by END
  CONST 27 10 NameLen 20   DescLen 40 END
-- The MEMBERS keyword signals the beginning of a list of
-- DictEntry objects must be matched by an ENDGRP statement;
-- this is defined by the base class Persist.GroupDesc,
-- from which Values.DataDictDesc descends
  MEMBERS
    OBJ Date "date" { DMY CENTURY } ENDOBJ
    OBJ Time "time" { MINUTES } ENDOBJ
-- A structure may use the RECORD keyword to start a list
-- of record fields, which must be terminated by an END statement
  OBJ Struct "book"
    RECORD
      OBJ Num "id" LONG LEN 0 ENDOBJ
      OBJ Str "author" LEN NameLen ENDOBJ
      OBJ Str "title" LEN NameLen ENDOBJ
      OBJ Str "description" LEN DescLen ENDOBJ
    END
  ENDOBJ
  ENDGRP
ENDOBJ
-- Here is the object group assigned to the project object,
-- which may contain any other type of persistent object
  OBJ Group "Demo"
    MEMBERS
-- The following object defines a menu with several pulldown menus.
-- Another form of menu often used is the PopUp menu
    OBJ Menu "main"
      ITEMS
        PULLDOWN "&File" OPTION Exit "&Exit" END
        PULLDOWN "&Help"
          OPTION HelpIndex "Help &Index"
          OPTION HelpOnFocus "Help on &Topic"
          OPTION HelpOnHelp "Help on &Help"
        BAR
          OPTION About "&About application"
        END
      END
    ENDOBJ
-- The Brush object defines a background colour with the name "edit"
  OBJ Brush "edit" 0x8000 ENDOBJ
-- The following objects instances of WinMgr.Window or derived classes,
-- containing objects derived from the class WinMgr.ObjectDesc
```

```

OBJ Win "MAIN" ICONIZE ZOOM SYSMENU FRAME THICK PARENT "MAIN"
  MENU "main"
  BRUSH "WorkArea" POS -4 , -4 1032 , 776
  SCROLL 1 , 1 TITLE "Demo V.1.0" ENDOBJ
OBJ Mask "bookEdit" ACTIVE MKCLIENT USE3D FRAME DIALOG
  PARENT "MAIN"
  BRUSH "edit" POS 74 , 103 394 , 175 SCROLL 6 , 16
  TITLE "Book Info"
  PARTS
-- Field is a data entry field linked to a variable defined in the
-- data dictionary (cf. above)
  OBJ Field "Author" ACCESS POS 50 , 32 300 , 56
    VALUE book.author OFFSET 40 ENDOBJ
  OBJ Field "Title" ACCESS POS 62 , 64 372 , 88
    VALUE book.title OFFSET 28 ENDOBJ
  OBJ Field "Description" ACCESS POS 22 , 96 372 , 120
    VALUE book.description OFFSET 68 ENDOBJ
-- A button is the typical user interface element, implemented
-- as Buttons.Button; the ID defines the command code returned
-- when the button is pressed, based on the command table (cf. above)
  OBJ Button "Ok" ID Ok ACCESS DEF POS 90 , 128 168,152 ENDOBJ
  OBJ Button "Cancel" ID Cancel ACCESS
    POS 180 , 128 258 , 152 ENDOBJ

  END
ENDOBJ
OBJ Win "bookTop" ACTIVE FRAME THICK PARENT "MAIN"
  BRUSH "Backgrnd"
  POS 30 , 74 326 , 435 SCROLL 8 , 10 TITLE "Books"
  PARTS
-- A ChildWin defines a child window which is always displayed together
-- with it's parent window; the PARENT keyword defines which
-- window owns another window and may be omitted for child windows, as
-- the relationship is implicit; NB: windows may also be owned by parent
-- windows without being child windows, in which case the PARENT keyword
-- is required!
  OBJ ChildWin POS 0 , 0 318 , 386
    ATTACH "" 0 FILL "" -25 0
    CHILD OBJ Scroll "bookList" ACTIVE DROPOK VSCR FRAME THIN
      PARENT "bookTop" BRUSH "LtGray" POS 0 , 0 318 , 386
      SCROLL 1 , 1
-- Scroll windows may also attach a mask and commands
-- used for standard data entry handling
    MASK "bookEdit" ADDCMD AddBook DELCMD DelBook
    EDITCMD EditBook ENDOBJ
  ENDOBJ
  OBJ ChildWin POS 0 , 386 318 , 411
-- Child windows may define a horizontal and a vertical attachment
-- This allows auto-managed groups of windows; don't worry too
-- much about the syntax here; use the A3Edit interface instead
    ATTACH "" 0 FILL "bookList" 0 FILL
    CHILD OBJ Win "bookCmd" ACTIVE AUTOFILL USE3D
      PARENT "bookTop"
      BRUSH "Gray" POS 0 , 386 318 , 25 SCROLL 8 , 10
      PARTS
        OBJ Button "AddBook" ID AddBook ACCESS
          POS 0 , 0 106 , 25 ENDOBJ
        OBJ Button "EditBook" ID EditBook ACCESS
          POS 106 , 0 212 , 25 ENDOBJ
        OBJ Button "DelBook" ID DelBook ACCESS
          POS 212 , 0 318 , 25 ENDOBJ

      END
    ENDOBJ
  ENDOBJ
END
ENDOBJ
-- The database object is a class derived from Persist.Group;
-- it may define the basic access path for the data files

```

```

    OBJ Db "main" PATH "data\"
      MEMBERS
-- Only at the level of each file do we decide which database
-- manager to use
    OBJ BtrFile "book" DATA "book"
      KEY 0 "id"
      SEG "id" AUTOINC END
      KEY 1 "title"
      SEG "title" DUP MOD NOCASE END
      KEY 2 "author"
      SEG "author" DUP MOD NOCASE END
    ENDOBJ
  ENDGRP
ENDOBJ
ENDGRP
ENDOBJ
ENDOBJ

```

Note that object scripts are generated exactly like this by the command

```
Persist.EncodeObj (project);
```

including the automatic indentation etc. – except for the annotations, which were added afterwards. For now, there is no direct way to add comments inside an object script file, except by adding comment objects.

Amadeus-3 also includes support for multi-layer projects, i.e. projects files that use information from other project files. In this way, you may use separate object script files for various sections of your application and yet share some elements between them, such as references to variables, windows and more. As usual, the restriction is that there may be no circular reference. The immense advantage is that you may create utility libraries that rely on the presence of certain objects, defined in their own object scripts, that can be included independently of the application interface elements.

The syntax for the inclusion of other object script files is:

```

OBJ Prj "Demo"
  REQUIRES "BASE"

```

On reading this instruction, the parser will check whether the requested script has already been loaded; if there is no project of the specified name, the parser will attempt to find an object script of the corresponding name, i.e. BASE.TX3 and read it, if found. If the requested project cannot be found either way, the parsing process is abandoned with an error message.

### 2.3.7 Amadeus-3 standard class names

Here is a list of predefined object classes, sorted first by *class name*, then by *exporting module*:

	Class name	Exporting Module	Exporting Module	Class name
1	Bitmap	Bitmaps.Object	Bitmaps.Brush	PatBrush
2	Bool	Toggles.Value	Bitmaps.Object	Bitmap
3	Brush	Colours.Brush	Btr.File	BtrFile
4	BtrFile	Btr.File	Buttons.Button	Button
5	Button	Buttons.Button	Buttons.Graphic	GButton
6	ChildWin	WinMgr.ChildWindow	Colours.Brush	Brush
7	Class	Persist.Class	Colours.Pen	Pen
8	Cmd	Commands.CmdTable	Commands.CmdTable	Cmd
9	DataDict	Values.Dictionary	DateVal.Value	Date
10	Date	DateVal.Value	Db.Database	Db
11	Db	Db.Database	Db.File	DbFile
12	DbFile	Db.File	DbSelect.Field	DbSelect
13	DbSelect	DbSelect.Field	DbView.Source	DbView
14	DbView	DbView.Source	DrawObj.Coloured	DrawColr
15	Draw	DrawObj.Object	DrawObj.Object	Draw
16	DrawColr	DrawObj.Coloured	DropList.Field	DropList
17	DropList	DropList.Field	DropList.OptionList	Options
18	External	Values.External	Fields.EditField	Field
19	Field	Fields.EditField	Fields.Label	FldLbl
20	FldLbl	Fields.Label	Fonts.Font	Font
21	Font	Fonts.Font	GrpView.Source	GrpView
22	GButton	Buttons.Graphic	Icons.Object	Icon
23	Group	Persist.Group	ImgFiles.ObjectList	Image
24	GrpView	GrpView.Source	ListView.Source	ListView
25	Icon	Icons.Object	Menus.Menu	Menu
26	Image	ImgFiles.Object	Menus.PopUp	PopUp
27	ListView	ListView.Source	NumToStr.Value	NumStr
28	Mask	Scroll.Mask	NumVal.Range	NumRange
29	Menu	Menus.Menu	NumVal.Value	Num
30	Num	NumVal.Value	Persist.Class	Class
31	NumRange	NumVal.Range	Persist.Group	Group
32	NumStr	NumToStr.Value	RealVal.Range	RealRange
33	Options	DropList.OptionList	RealVal.Value	Real
34	Outline	RptTbl.Outline	Reports.Report	Report
35	PatBrush	Bitmaps.Brush	RptTbl.Outline	Outline
36	Pen	Colours.Pen	RptTbl.Table	Table
37	PopUp	Menus.PopUp	Scroll.Mask	Mask
38	Prj	Startup.Project	Scroll.Window	Scroll
39	Real	RealVal.Value	Scroller.Field	Scroller
40	RealRange	RealVal.Range	Sequence.Source	Source
41	Report	Reports.Report	Sequence.Stepper	Stepper
42	Scroll	Scroll.Window	Startup.Project	Prj
43	Scroller	Scroller.Field	StrDisp.Object	StrDisp
44	Source	Sequence.Source	StrVal.Value	Str
45	Stepper	Sequence.Stepper	Tabbed.Control	Tabbed
46	Str	StrVal.Value	Tabbed.Tab	Tab

	<b>Class name</b>	<b>Exporting Module</b>	<b>Exporting Module</b>	<b>Class name</b>
47	StrDisp	StrDisp.Object	Tabbed.TabChild	TabChild
48	Struct	Values.Struct	Tabbed.TabWin	TabWin
49	Tab	Tabbed.Tab	TextEdit.Object	Text
50	TabChild	Tabbed.TabChild	TimeVal.Value	Time
51	TabWin	Tabbed.TabWin	Toggles.Group	TglGrp
52	Tabbed	Tabbed.Control	Toggles.Toggle	Toggle
53	Table	RptTbl.Table	Toggles.Value	Bool
54	Text	TextEdit.Object	ValueDsp.Object	ValueDsp
55	TglGrp	Toggles.Group	Values.Dictionary	DataDict
56	Time	TimeVal.Value	Values.External	External
57	Toggle	Toggles.Toggle	Values.Struct	Struct
58	ValueDsp	ValueDsp.Object	WinMgr.ChildWindow	ChildWin
59	Win	WinMgr.Window	WinMgr.Object	WinObj
60	WinObj	WinMgr.Object	WinMgr.Window	Win
61	Container	Persist.Container	Persist.Container	Container

## 2.4 The Data Dictionary

The data dictionary is a central element of Amadeus-3. It is implemented in module `Values`. It allows a program to create a logical mapping of Oberon-2 data structures, usable during program execution for tasks such as mapping internal variables to various output formats, in particular string representations. This representation is then used in many places, such as when information must be transferred between internal variables and data entry fields. In database applications, it is used for mapping variables to indexed files.

To map variables and their logical representation, the module `GenCode` may generate initialisation code that will perform such a mapping. Some Oberon systems use symbolic information supplied by the compiler to create this mapping, but this implies a great degree of dependence from the compiler.

Several object classes cooperate to implement the complete range of functionality:

Class	Description
<b>Dictionary</b>	Derived from <code>Persist.Group</code> , this is a group that only accepts dictionary objects. As special feature, it is able to search for objects with composite names, i.e. record names. A dictionary also has an attached constant table. This may be used to implement more flexible data definitions based on constant identifiers rather than fixed numbers for such elements as array or string length.
<b>DictEntry</b>	The basic dictionary entry, i.e. an abstract class of objects that may be added to a data dictionary group.
<b>Struct</b>	This is a mapping of the RECORD type. A <code>Struct</code> may contain a list of any other <code>DictEntry</code> object as members, obviously also other <code>Struct</code> elements (recursive definition).
<b>External</b>	A <code>DictEntry</code> object used as placeholder. It may be further defined by the user but has no specific functions or meaning, except unformatted information storage.
<b>Value</b>	A <code>DictEntry</code> class that defines what we usually think of as a simple variables, such as strings, integers, real- date- and time-values. All objects derived from this class should be able to externalise and internalise themselves as strings, making them suitable for data entry operations.

Module `Values` doesn't define specific instances of variable types. This is handled by the following standard modules:

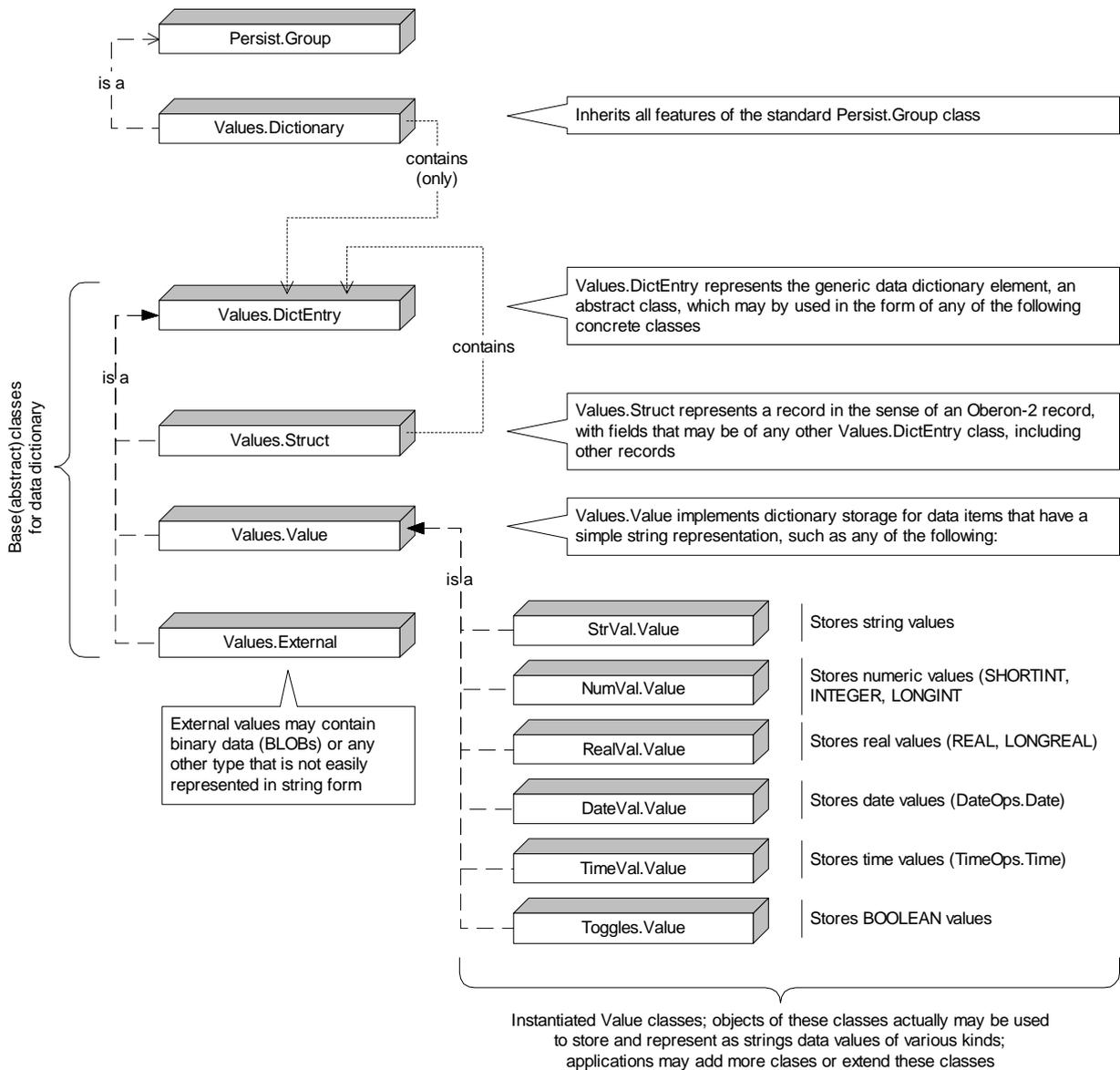
Module	Data type handled
<b>StrVal</b>	<b>Strings</b>
<b>NumVal</b>	<b>Numeric values</b> - <code>INTEGER</code> , <code>SHORTINT</code> , <code>LONGINT</code>
<b>RealVal</b>	<b>REAL values, real range</b>
<b>Toggles</b>	<b>BOOLEAN values (may also reference NumVal values)</b>
<b>DateVal</b>	<code>DateOps.Date</code> <b>values</b>
<b>TimeVal</b>	<code>TimeOps.Time</code> <b>values</b>
<b>DecimalVal</b>	<code>DecimalVal.Value</code> <b>values, decimals with high precision</b>

Many other modules are based on definitions from module `Values`. The generic database interface defined in module `Db`, for example, would be quite impossible without the concept of a data dictionary and the methods supplied in module `Values`.

And let's not forget the main reason for all this: the user interface. Values are behind every data entry field that accepts input that is easily transformed into the value's internal data storage format

(strings, numbers, Boolean, date and time etc.) and many other user interface features. For further information on these subjects, please read the documentation for the following modules, which all make direct and intense use of Values, in one form or another:

- Fields
- Toggles
- Scrollbars
- ValueDsp
- Icons
- Db

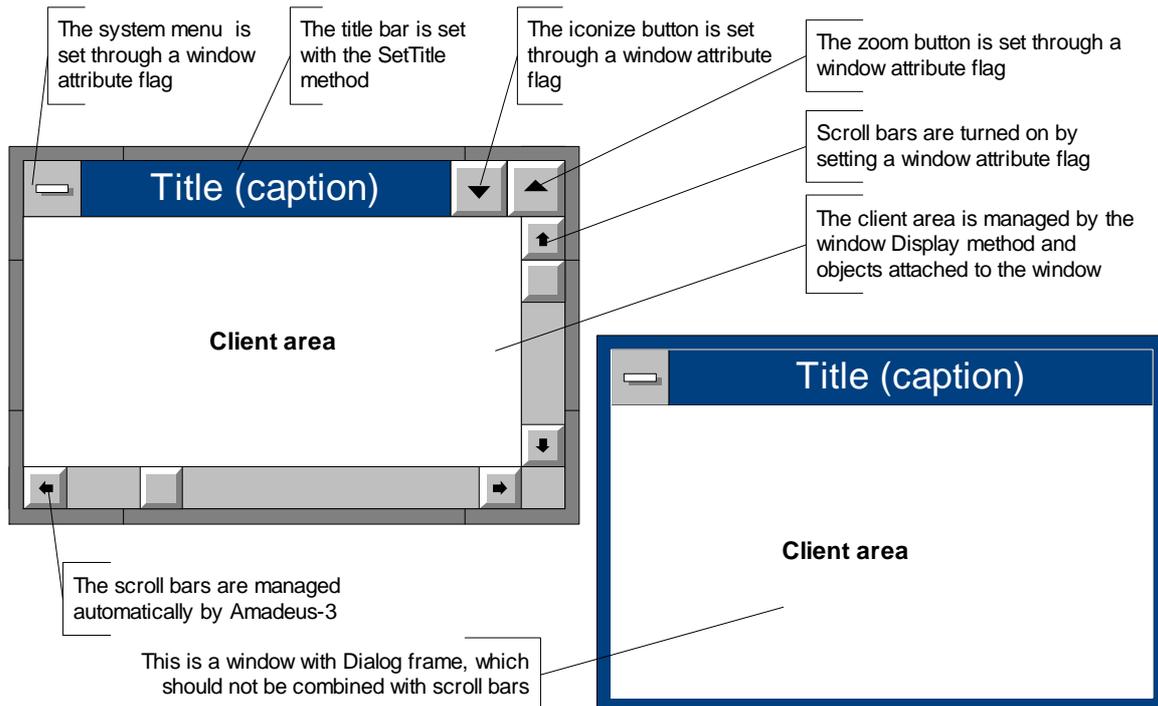


**Figure 6 - Data dictionary concept**

## 2.5 Windows and display

The entire user interface in Amadeus-3 is based on the window paradigm. As every program in a GUI environment needs at least one window to communicate with the user, Amadeus-3 supplies a default main window as `WinMgr.mainW`. This window is always created by the project initialisation method and is therefore available to the entire framework<sup>7</sup>.

An application may generate any number of windows (within the limits of the GUI). All standard window actions are monitored by Amadeus-3, allowing it to maintain status information on all windows, such as whether a window is displayed, its dimensions and the contents of the client (drawing) area. This information is made available to the application through the window object record, so the application does not have to inquire the GUI to determine the state of any window.



**Figure 7 - Basic window attributes**

Once a window is displayed, it will automatically receive various messages from the GUI. Some of them will request drawing operations, some will be other events, some related to system-management, some to user input. Event management will be treated in the next chapter, 2.7 Message handling. For now, let's have a look at all the methods that concern window display management. First, let's look at some of the basics:

### 2.5.1 Windows

In a system with overlapping windows, it is very important that every window knows how to draw its contents at any given moment, as efficiently as possible. Efficiency in speed, but also in memory requirements. This usually precludes the simple saving of the entire window contents as bitmap, which could then be used to regenerate the display at any moment. On one hand, this would be very inflexible, on the other it would consume huge amounts of memory. Instead, every window should have a method by which it can redraw its contents quickly, if possible even only in part, without completely repainting the entire window every time, as it often occurs that only part of a window is exposed.

<sup>7</sup> If you use an application resource file, you may override the aspect and settings for the default `WinMgr.mainW` object by including a window with the name « MAIN » in your resource file.

Amadeus-3 helps by supporting various methods to paint and manage a window's client area, which is the area not managed by the GUI system. System-managed areas include the title, the frame and any special icons, plus the scroll bars. Basically, you have two possibilities:

- You may either sub-class the window and perform drawing functions from the window's Background and Paint methods.
- Or you may add display objects ([WinMgr.Object](#)) to the window, which will then take care of painting their own contents when required.
- Of course you can freely combine these two methods as may suit your needs.

Here is a closer analysis of **window methods**:

- [Window.Background](#): is used to paint the window background before the window's Paint method is invoked. By default, this method simply uses the current window brush (set in the field [Window.backg](#)) to fill in the entire exposed area. If you assign the [Colours.nullBrush](#), this operation will not be performed, basically leaving the window « transparent ». Never directly call this method, leave it to the system to paint your window background.
- [Window.Paint](#): performs the actual painting of the window's client area. By default, it steps through all display objects assigned to the window through the [Window.Add](#) method and calls the [Object.Display](#) method for each object that falls into the exposed area. You should never directly call this method.
- [Window.Validation](#): you can use it to mark an area specifically as valid or invalid (and therefore requiring refreshing).
- [Window.Update](#): forces the painting of any invalid area of a window without waiting for a paint message to come in through the message queue.
- [Window.Refresh](#): request all window elements to refresh their current state to match any underlying data structure, e.g. data entry fields will be refreshed based on the value of attached variables. It will also update the window's current state - hidden or displayed - according to the [Window.displayed](#) flag. After the entire refreshing process, the Update method is called to force painting any changed areas.
- [Window.Create](#): Creates a matching GUI object, where possible, based on the parameters supplied with the window, but without displaying it.
- [Window.Show](#): requests that a window be displayed. The parameter allows the specification of the « Z-order », i.e. in what position the window should appear if all windows currently displayed are considered to be on a vertical stack.
  - ◊ If you specify [WinMgr.TopPos](#), the window will be shown at the top of its local stack, i.e. the stack formed by it's parent window - if any - at the bottom and it's sibling child windows.
  - ◊ If you specify [WinMgr.BottomPos](#), the window will be moved to the bottom of it's local stack.
  - ◊ If you specify [WinMgr.SamePos](#), no change will be made to the current « Z-order ».
- [Window.Hide](#): removes the window from the display list, but leaves the GUI object intact.
- [Window.Destroy](#): destroys the GUI object - after hiding it, if displayed -, but leaves the window object intact.

There are further window methods for managing specific functions:

- [Window.Move](#): moves a window to a specific location. By sub-classing, you may alter the way a window reacts to being moved around.
- [Window.Resize](#): Changes a window's extent. Again, you may alter the default behaviour by sub-classing the window.
- [Window.HandleScroll](#): Handles scrolling events by adjusting the window's offset or in any other required way.
- [Window.Handler](#): Receives all events directed at the window and not already handled; cf. next chapter « 2.8 Message handling ».

Then there are the functions required for **managing attached display objects**, defined as `WinMgr.Object`:

- `Window.Add`: Adds a display object to a window. The default co-ordinates of the top left corner are those currently set for the display object. The location within the list of objects already attached to the window is by default defined by topological sorting based on the top left corner of the object. If the `WinMgr.Object.pos.x2` is defined as `MIN (INTEGER)`, i.e. out of range, the extent of the object is considered unknown and will be recalculated. Otherwise, the extent is supposed to be the current extent.
- `Window.PlaceOfInsert`: Used to recalculated by the place of insertion and/or the co-ordinates of the an object being added to a window. Used by `Window.Add` method. Should not be called directly.
- `Window.Remove`: Removes a display object from a window, but keeps the object intact.

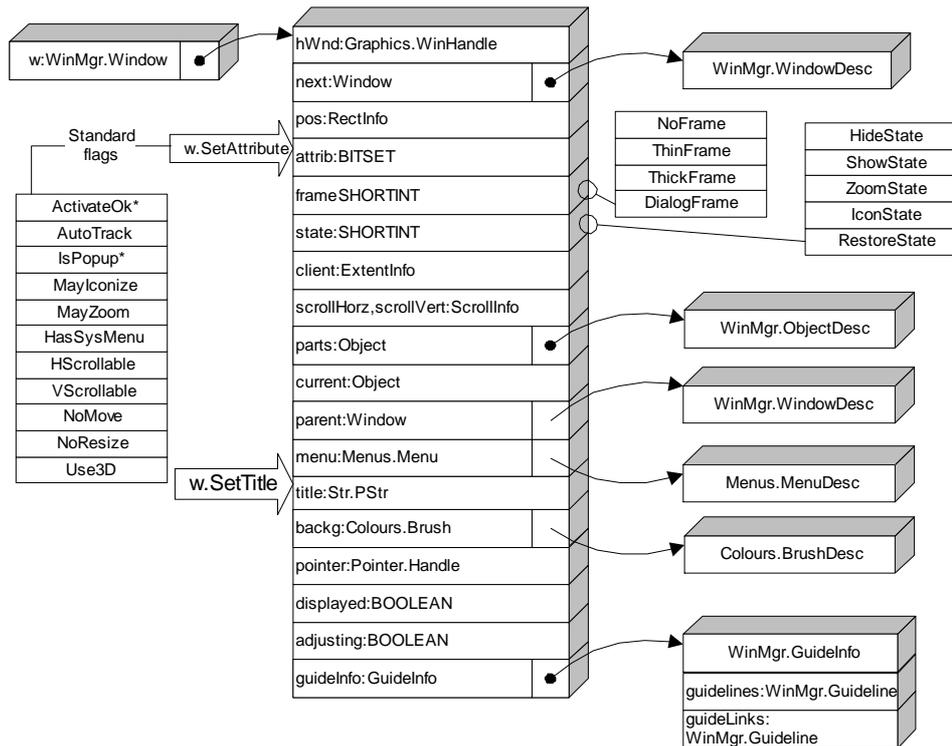
At this point, we must discuss an important concept: the **graphical context**. Most GUIs support graphical context in one form or another. The context contains information about the resolution of the output device, the colours that may be represented, the translation mode from logical to physical co-ordinates, the currently active font, foreground and background colours and more. No paint function could proceed without this type of information. Writing a string in a window, for example, would be meaningless if the active font and the colours to be used were not specified or if the available resolution was unknown. Where this graphical context is not included as parameter to a method, it may be retrieved by using the following methods:

- `Window.GetContext`: Returns the currently active graphical context. Only meaningful if the GUI object corresponding to the window object has already been created.
- `Window.ReleaseContext`: When `GetContext` was called, `ReleaseContext` *must* be used to free the reference to this context, as each such reference may use up GUI resources.

Another concept necessary for managing a user interface is the **Focus**. This term designates the window that receives all keyboard input. GUIs tend to handle this in different ways. Windows sets the focus independently of the mouse, whereas Motif assigns the focus to whatever window the mouse is pointing at. Amadeus-3 doesn't make any assumption one way or the other, but it supports focus handling through the following items:

- `WinMgr.hasFocus` is a global variable pointing to the display object that currently has the focus, if any.
- `WinMgr.hWndFocus` is the GUI handle (defined in module Graphics) that points to the window that currently has the focus, if any. It is possible that `hWndFocus` is set while `hasFocus` is `NIL`. The focus does not necessarily have to be attached to a display object.
- `WinMgr.SetFocusHandle`: sets `hWndFocus`, making any necessary adjustments. It will also set the activation state of the new focus owner and of all it's parent windows.
- `WinMgr.ReleaseFocusOwner`: may be used to clear the current focus owner. If `WinMgr.Null` is specified as parameter, any owner is cleared. Otherwise the focus is only released if the specified handle matches the current owner.
- `WinMgr.Object.SetFocus` is the method that allows each display object to acquire or release the focus. See the chapter on display object for more details.
- `Window.AcquireFocus` is a function that attempts to pass the focus to the first available display object in the specified window.

In addition to the information provided by all the fields in a Window record, some information is provided through **attribute flags**, which are grouped in `Window.attribute`. You cannot manipulate the flags in this field directly, you have to use the `Window.SetAttribute` method instead. This allows Amadeus-3 to perform additional checks and enforce certain rules governing these flags. Window sub-classes may add new flags at no cost in terms of memory (up to the maximum of 32, at which point a new window class would have to add a new field). Here are the standard attributes for windows and the meaning when the attribute is on:



**Figure 8 - Important window fields**

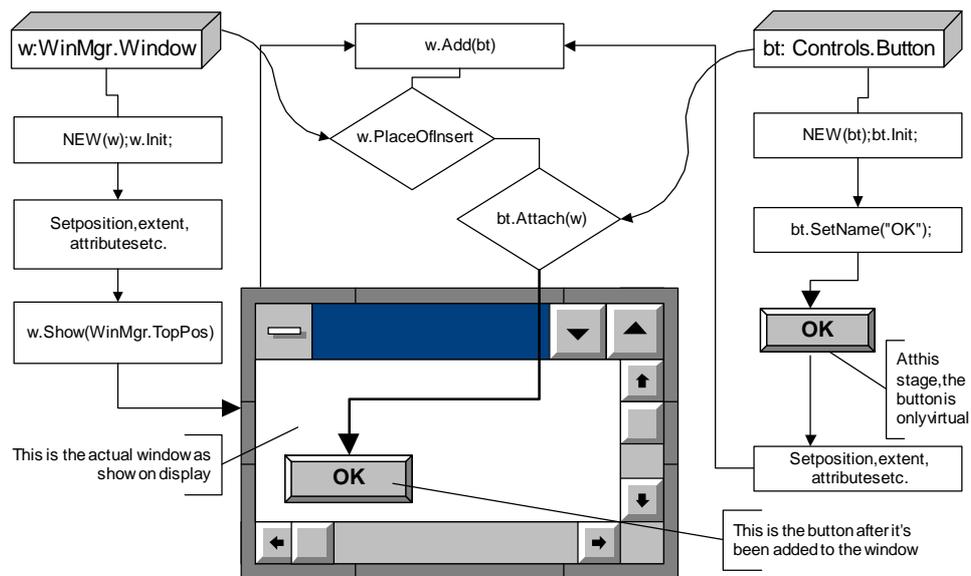
- ◇ **ActivateOk** : may be activated by a mouse click
- ◇ **AutoTrack** : will scroll to follow focus, if focus object outside visible area
- ◇ **AcceptsDrop** : accepts dropped objects
- ◇ **IsPopup** : is a popup window (special window style)
- ◇ **MayIconize** : has Iconize button in title bar
- ◇ **MayZoom** : has Zoom button in title bar
- ◇ **HasSysMenu** : has system menu and corresponding box in title bar
- ◇ **Hscrollable** : may display horizontal scroll bar
- ◇ **Vscrollable** : may display vertical scroll bar
- ◇ **NoMove** : may not be moved by user (even if it has a title bar)
- ◇ **NoResize** : may not be resized by user (even if it has a resizable frame)
- ◇ **NoReorder** : Z-order may not be changed, window remains at fixed z-order position
- ◇ **Use3D** : uses 3D look, where possible
- ◇ **MkClientSize**:create window based on client size, not frame dimension
- ◇ **MaxWinAttr** : Last attribute for this module; attributes for descendent classes after this

## 2.6 Display objects

Display objects are self-managed entities that may be attached to a window. Each display object knows how to attach itself to a window (if any special process is required), knows how to calculate its dimension for a given owner's graphical context and is able to paint itself. They may also react to user input and may acquire the

Display objects may be very simple or very complex. The major advantage of implementing a given painting algorithm through a display object's `Display` method rather than through a window's `Paint` method is flexibility. Properly implemented, the same painting algorithm may be used in various circumstances. As an example, consider that any display object may theoretically be used as a column in a scrolling window, which gives an incredible range of possibilities for formatting such windows.

The most important properties of display objects are their location and dimension. These allow the owner window to calculate the visibility of any object, whether it needs to be re-painted or if it should receive a given mouse message. When a window's `Add` method is called for a given display object, it will first be determined where - and if - this object should be inserted. This is done partly by the `Window.Add` method itself, partly by the `Window.PlaceOfInsert` method. Then, if a GUI representation and graphics context exist for the target window, the actual extent of the display object will be calculated, its `Attach` method is called and if all was successful, `w.Add` returns `TRUE`:



**Figure 9 - Adding a display object to a window**

Various attributes may be set for each display object, just as you can set attributes for windows. The field and method name are the same, too: `Object.attrib` and `Object.SetAttribute`.

### 2.6.1 Standard display objects

Here are some of the standard display objects supplied by Amadeus-3:

- **Controls.Control**

This is an abstract class which implements control objects, such as buttons, data entry fields, toggles and other standard user-interface control elements. The abstract class encapsulates a lot of the required interface code for generating standard interface objects as supported by the GUI. Under Motif, these elements are called "Widgets".

- **Buttons.Button**

This is the most standard user interface element: the button, which may be active or inactive, pressed or depressed and which reacts to mouse or keyboard input, sending command messages to the application when pressed. Module Buttons now implements these elements without any assistance from the GUI.



Standard button

The label displayed on the button is changed with the `SetName` method, inherited from `Persist.Object`. The command code generated by a button is the object's id, also inherited from `Persist.Object`.

- **Buttons.GraphicButton**

Basically the same as `Buttons.Button`, except that you have to supply 2-4 bitmaps to represent the various button states. As bitmaps are used to represent the various button states, you may apply absolutely any effect you desire, including fancy 3D effects etc.



Normal state



Pressed state etc. Owner-drawn button

Again, the button name is set through the `SetName` method, but in this case, the name is used to define the bitmaps that will be loaded for the various button states.

- name + '0' defines the bitmap for the normal state
- name + '1' defines the bitmap for the pressed state
- name + '2' defines the bitmap for the selected state
- name + '3' defines the bitmap for the inactive state

Example: A graphic button is named « buttons\user ». When the button is attached to a window, it will automatically attempt to load the following button bitmaps from the « buttons » subdirectory:

« user0.bmp », « user1.bmp », « user2.bmp », « user3.bmp »

If any other bitmap loading algorithms are installed, all the corresponding formats will be available for use, i.e. PCX, GIF or JPG formatted files could be loaded instead. Any unavailable bitmap will simply be ignored.

For further information on the use of bitmaps, please the entry on `Bitmaps.Object` below.

- **Fields.Field**

This is the standard data entry field, as abstract class. *Amadeus-3* links it directly to a data dictionary object and takes care of maintaining a tight relation between the display field and the dictionary object. The object's name is automatically interpreted as the field's label string, unless specified otherwise. The label is considered part of the display object and may be moved together with the field.

Some field types may be displayed with a 3D effect, where the corresponding flag is set for the owner window.

- **Fields.EditField**

This class assumes that your data will be entered as character data and is linked with standard GUI text entry fields.



Data entry field, no label



Data entry field with label

- **Fields.Label**

This object supplies a simple way of adding additional text labels with no further function in any window. The text displayed is static. It is changed through the `SetName` method, which is inherited from `Persist.Object`. This means that the name of a label object is equivalent to the string it displays.

**Window name**

Label string

- **StrDisp.Object**

Basically the same idea as `Fields.Label`, except that the string displayed is provided through a method, which makes this object class semi-static. It's intended for use as column object in scroll windows (cf. « 2.12 Scrolling »).

- **DropDown.Field**

This is a special form of data entry field based on `Fields.EditField`. It is usually represented by a field with an attached selection list (dropdown list).



Selection field with label

- **Toggles.Toggle**

This is a "check box" object. It is linked to a Boolean dictionary object. By clicking this object, the flag linked to it is turned on or off. When the flag is on, this is reflected by a « X » or check mark character being displayed

A groups of toggles may be combined to form a « multiple choice » input device. These toggles may be either mutually exclusive or may allow several options to be turned on simultaneously. The result may be returned as an `ARRAY OF BOOLEAN`, as a single integer value or as a `LONGINT` representing a `BITSET` (the latter in the case of non-exclusive multiple choices).



Checkbox; clicking this control item with the mouse turns the marker alternatively on and off.



Group of mutually exclusive toggles. Clicking one toggle of this group will turn on that toggle and will turn off all the others. The group frame and title were drawn using `DrawObj.Object` with attribute `Rectangle` (cf. entry about `DrawObj` display objects).

- **Scroller.Field**

This is another control element, linked exclusively to `NumVal.Value` variables. The value of the variable, which must be in a range between a specified minimum and maximum, is represented as the location of the scrolling box (« thumb »), proportionally to the specified range of values. Scrollbars may be either horizontally or vertically placed.



Any change in the position of the scrolling box will immediately be translated into a change in the variable. If the variable is changed, then the next call to the `Refresh` method of the window or the scrollbar will properly represent the new value on the scrollbar.

**Note:** If a numeric variable is simultaneously linked to a standard data entry field and a scrollbar in the same window, then any change in one will immediately be reflected in the other. This is valid for any number of elements linked to the same value in the same window, also for other types of widgets.

To cross-update widgets in other windows, you will have to extend the standard method `UpdateOthers` for the corresponding fields to suit your needs.

- **DrawObj.Object and DrawObj.Coloured**

Here you find a few basic graphical objects, such as lines, rectangles and ellipses, in association with a text string for the purpose of labelling. `DrawObj.Coloured` are colour-filled versions of the same objects. Box and line objects may be displayed with a 3D effect, where the corresponding flag is set for the owner window.



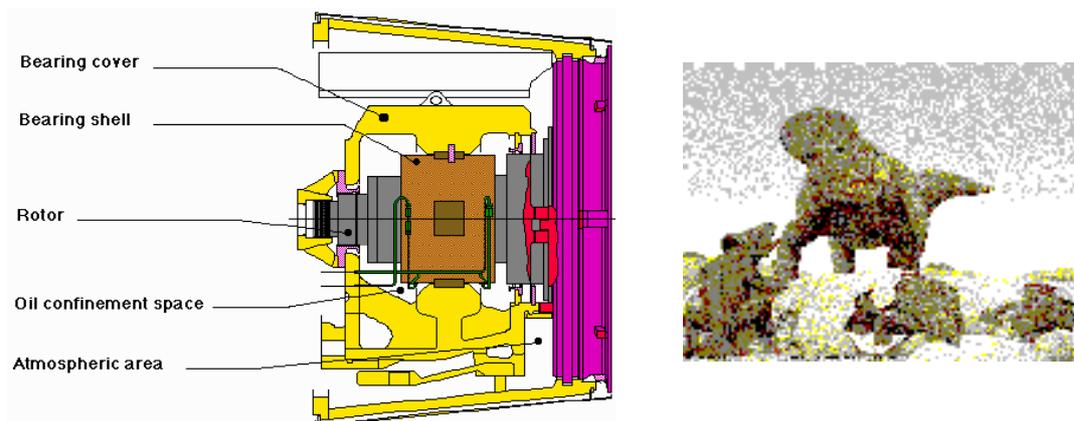
A few examples of basic shapes

You may attach a title to any of these objects through the `SetName` method. In addition, the title may be placed in different places (top, bottom, centre, on the left, on the right, centred), which is specified by a special attribute.

- **Bitmaps.Object**

This is the typical bitmap object. `Bitmaps` defines some basic bitmap manipulation methods, such as scaling, copying etc. Loading methods for various graphics file formats may be dynamically installed. The default GUI bitmap format is always support (BMP files for Windows). Others, such as PCX, GIF, JPG etc. may easily be added, as demonstrated by the module `PCXFiles`, which makes use of an external library to load PCX files, which remains totally transparent for the application program.

The name of a bitmap (defined through the `SetName` method) is used as the filename of the bitmap to be loaded. It may or may not include an extension (cf. above) or a pathname. The loading process normally takes place when the bitmap is attached to a window.



**Figure 10 – Sample Bitmaps**

If the attribute flag « Accessible » is set for a bitmap, it becomes an active user interface element. Clicking it with the mouse will generate a command code just as if it was a button.

Some of the possible manipulations on display objects and bitmaps will be discussed based on a sample application in the tutorial chapter.

Module Bitmaps exports many operations for the manipulation of actual bitmaps, such as Copy, Resize, BitBlt etc. These functions operate on the bitmap handle attached to a bitmap object, or alternatively you may declare bitmap handles independently of a display object.

NB: ImgLib.OB2 adds the possibility of loading and saving graphics files from many other standard formats, such as TIF, GIF, JPG etc., but requires you to buy a commercial library that will support those formats.

**Note (Windows only):** You may include any bitmap in the Windows resource file ‘Application.RC’. If you do this, the bitmap will always be available during execution time, as it is integrated into the .EXE file. Any pathname you specified will be ignored while attempting to read bitmaps from internal resources. For examples, please see the sample applications supplied with Amadeus-3.

- **Icons.Object**

Between a data entry field and a bitmap object, this class allows you to link one or more icons to a numeric value and to represent this value through one of the associated icons. A field of this type may receive the input focus and reacts to mouse clicks and to the space bar. You can also use it to just display a single icon, of course.



Icon

Icon size and number of colours are usually specific for each GUI system. They load faster and are easier to manipulate than bitmaps, but they are more restricted.

**Note (Windows only):** As for bitmaps, you may include any icon in the Windows resource file ‘Application.RC’. All the comments for bitmaps on this subject apply also for Icons.

- **Tabbed Control**

Tabbed Controls are composed of a parent window, a control window and one or more tab windows. These are arranged in such a way that only one of the tab windows is visible at any given moment. The control window displays “Tabs”, that allow the user to select one of the tab windows, which then becomes active and is moved to the top, to be visible. You have already seen examples of such tab windows in many applications. Since they allow you to easily distribute information over many sub-windows, while keeping them well-grouped and using only as much space as the largest of the individual tab windows, they are very popular.

This implementation is specific to Amadeus-3 and not based on the GUI, which means that this user interface element will consistently be available on all platforms, without any changes (including Windows 3.x).

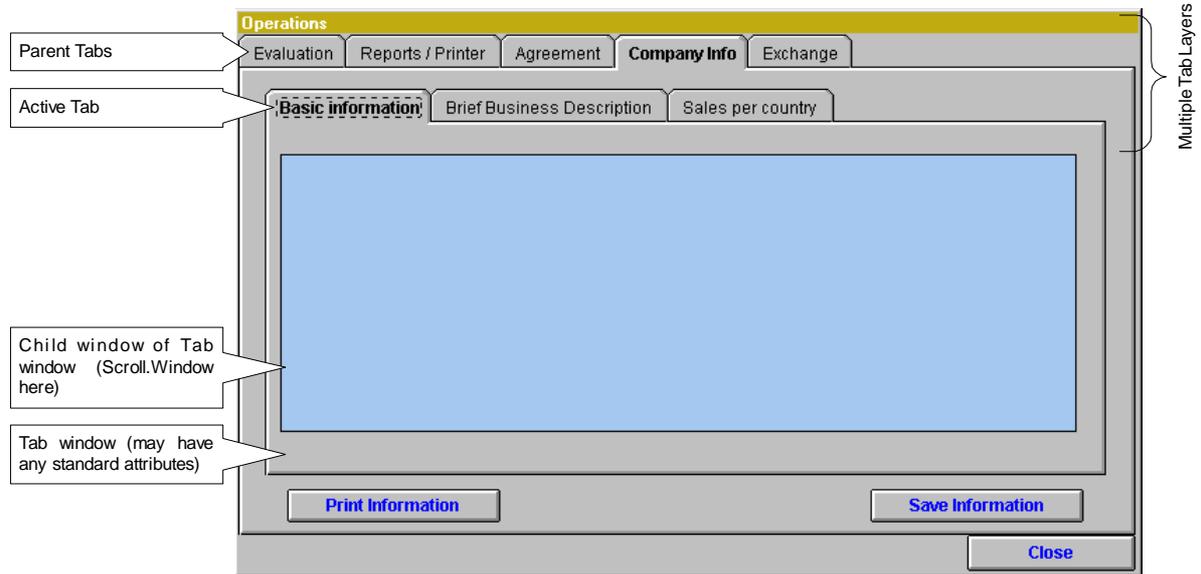
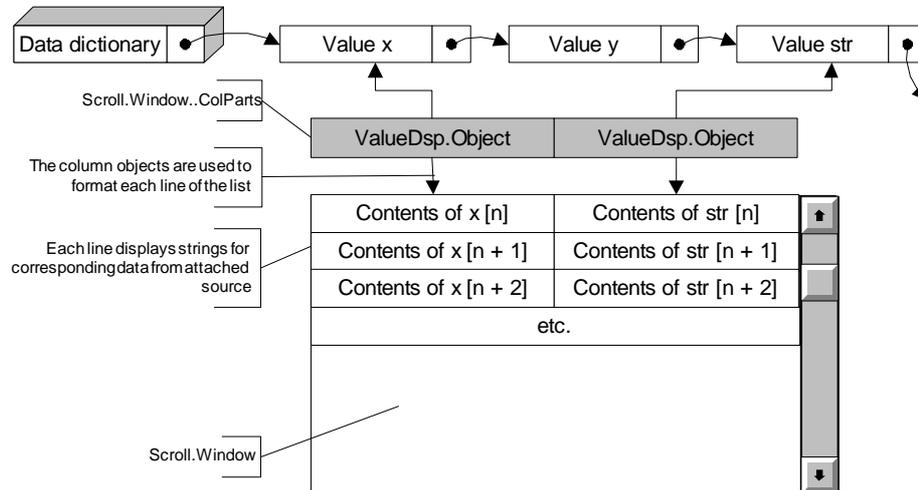


Figure 11 – Tab Control

- **ValueDsp.Object**

This is the most direct way to display values (cf. chapter « 2.4 The Data Dictionary » ) as strings but without any input mechanisms, whereas data entry fields are linked to values for data entry. It's very useful where you wish to display the contents of data dictionary variables. It's perfect for displaying variables in a scrolling list, cf. chapter « 2.12 Scrolling » where you will find a more detailed discussion of how to assemble scrolling windows. This is just a look at the specific use you may make of `ValueDsp` objects to display the contents of variables as strings in scrolling lists:



**Figure 12 - Use of `ValueDsp.Object` in scroll lists**

## 2.6.2 Creating your own display objects

Defining new display objects is a simple matter of creating a new object class with an appropriate `Display` and `Dimension`. The `Display` method takes care of drawing the object at its current coordinates. The `Dimension` method calculates the object's extent based on a given graphical context, which may be variable, as display objects can also be sent to a printer or some other device.

In addition, you may define an `Activate` method, which will handle input messages sent to the object (keyboard and mouse messages). If you want to accept keyboard input, be prepared to define the `SetFocus` method, which should take care of focus acquisition. Make sure the focus state is properly reflected by the `Display` method, so the user gets some feedback.

Please refer to the chapter « 2.3 Object Persistent » for more information on how to make new display objects fully persistent.

### Example:

In your Amadeus-3 distribution, you can find an example of an application that creates its own display objects to display projections of 3D objects. The application is called KS. The source code is split into the following files:

- KS.OB2 : Program initialisation, menu handler, termination
- Elements.OB2 : Classes of abstract 3D objects and projections of same
- BaseItem.OB2 : Implementation of 3 concrete sample element classes

The central module we are interested in right now is *Elements.OB2*. This is where the new display object class is defined.

The first new object class of this module, *Elements.Part*, is based on *Persist.Object*. This class represents the « physical » object with coordinates in 3 dimensions.

The display object class, *Elements.Projection*, refers to an object of class *Elements.Part*, which it will display according to the constraints of the owner window, which may specify the angle of projection and optionally a scaling factor.

Let's have a look at a graphical representation of the data structure behind these declarations. The little « clouds » are borrowed from the Booch method and represent object classes.

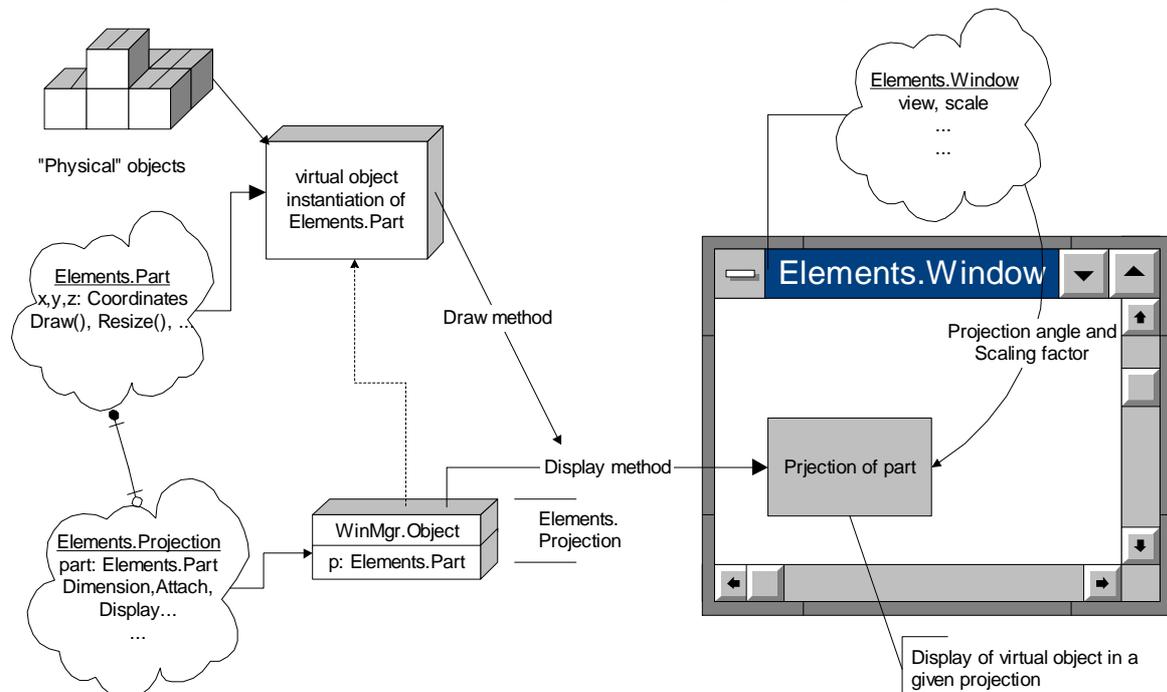


Figure 13 - New display object "Projection"

The following code excerpts include the object declarations and the important methods:

```

MODULE Elements;

IMPORT WinMgr, MemList, Graphics, Persist, Metrics, Str, SYSTEM;

(* ... constants etc. ... *)

TYPE
  Window* = POINTER TO WindowDesc;
  WindowDesc* = RECORD (WinMgr.WindowDesc)  view*: SHORTINT; scale*: LONGREAL;
END;
  Part* = POINTER TO PartDesc;
  PartDesc* = RECORD (Persist.ObjectDesc) x*,y*,z*: LONGREAL; END;
  Projection* = POINTER TO ProjectionDesc;
  ProjectionDesc* = RECORD (WinMgr.ObjectDesc)  of*: Part; END;

(* ... more code ... *)

PROCEDURE (p: Part) Init* ();
(* Initialization method or « Constructor » inherited from Persist.Object *)
BEGIN p.Init^; p.x := 0.0; p.y := 0.0; p.z := 0.0; END Init;

PROCEDURE (p: Part) Dimension* ( view: INTEGER; scale: LONGREAL; VAR w,h :
INTEGER);
(* Abstract method, should return element extent based on view and scale *)
BEGIN w := 0; h := 0; END Dimension;

PROCEDURE (p: Part) Draw* (VAR use: Graphics.PaintInfo;
view: INTEGER; scale: LONGREAL; atx,aty: INTEGER);
(* Abstract method that should draw the proper projection at coordinate atx, aty
for specified view and scale *)
BEGIN END Draw;

PROCEDURE (p: Part) Resize* (view: INTEGER; szx,szy: LONGREAL);
(* Abstract method used to resize an element in a given view *)
BEGIN END Resize;

PROCEDURE (p: Part) ClassName* (VAR s: ARRAY OF CHAR);
(* Standard method inherited from Persist.Object *)
BEGIN COPY ('KSPart', s); END ClassName;

PROCEDURE (pj: Projection) Dimension* (kind: INTEGER; VAR w,h: INTEGER);
(* Compute actual display dimensions of projection based on scale, view and
associated part *)
BEGIN
  IF (pj.of = NIL) OR (pj.owner = NIL) OR ~(pj.owner IS Window) THEN
    w := pj.pos.x2 - pj.pos.x1; h := pj.pos.y2 - pj.pos.y1;
  ELSE
    pj.of.Dimension (pj.owner (Window).view, pj.owner (Window).scale, w, h);
    pj.pos.x2 := pj.pos.x1 + w; pj.pos.y2 := pj.pos.y1 + h;
  END;
END Dimension;

PROCEDURE LocalCoord* (pj: Projection);
(* Compute projection coordinates if attached to view window *)
VAR
  sc : LONGREAL;
  w,h: INTEGER;
BEGIN
  IF pj.owner IS Window THEN
    pj.Invalidate (TRUE); sc := pj.owner (Window).scale;
    pj.pos.y1 := SHORT (ENTIER (sc * pj.of.y));
    IF pj.owner (Window).view = Front THEN pj.pos.x1 := SHORT (ENTIER (sc *
pj.of.x));
    ELSE pj.pos.x1 := SHORT (ENTIER (sc * pj.of.z)); END;
    pj.Dimension (WinMgr.InitialExtent, w, h); pj.Invalidate (TRUE);
  END;
END LocalCoord;

```

```

PROCEDURE (pj: Projection) Attach* (w: WinMgr.Window): BOOLEAN;
(* When attaching projection to view window, compute local coordinates *)
BEGIN IF pj.Attach^ (w) THEN LocalCoord (pj); END; RETURN TRUE;
END Attach;

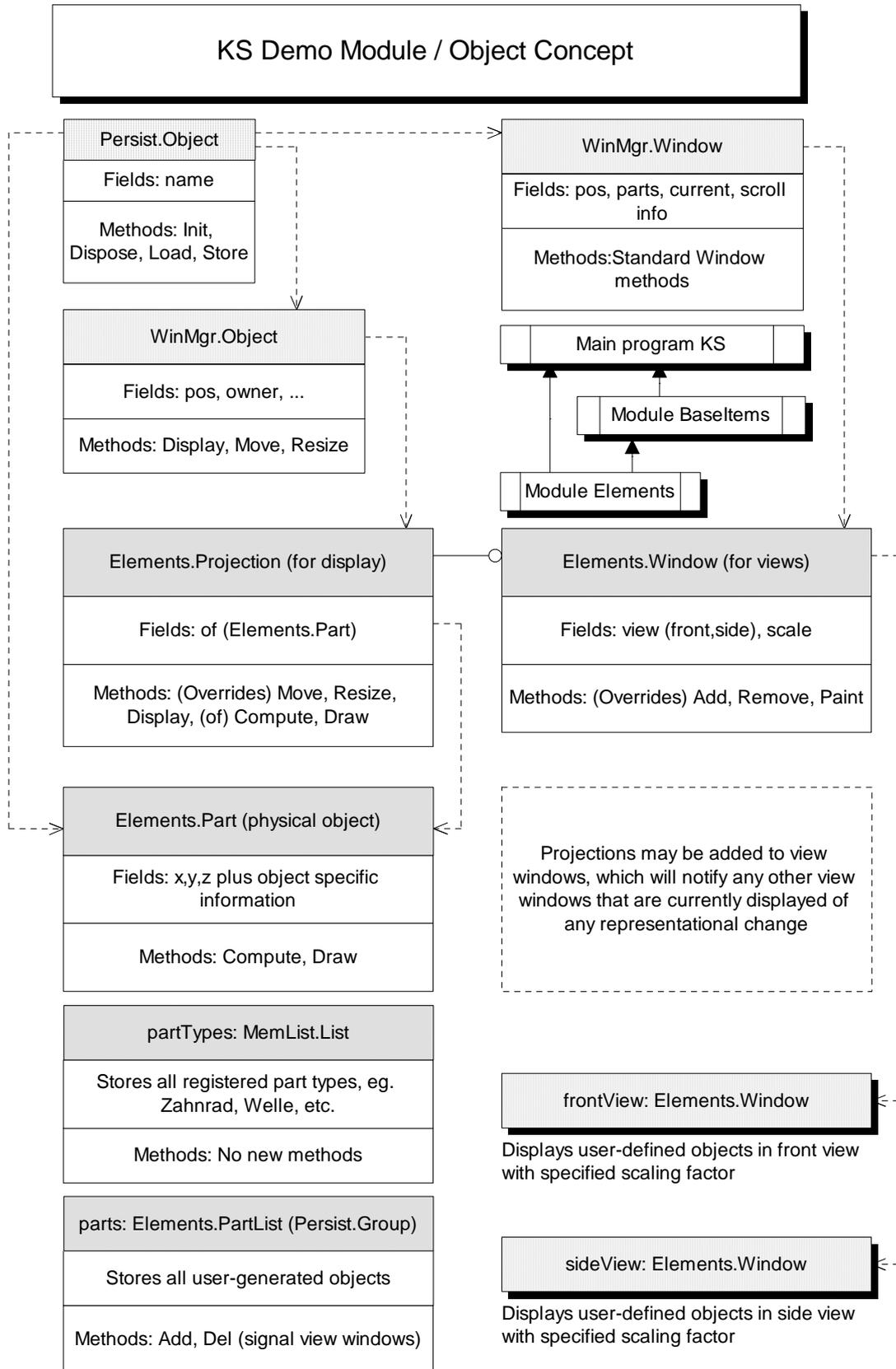
PROCEDURE (pj: Projection) Resize* (sizex,sizey: INTEGER);
(* Projection resize operation is applied to attached part, then copied in other
view windows *)
VAR szx,szy: LONGREAL;
BEGIN
  IF pj.owner IS Window THEN
    szx := sizex / pj.owner (Window).scale;   szy := sizey / pj.owner
(Window).scale;
    pj.of.Resize (pj.owner (Window).view, szx, szy);
    LocalCoord (pj); CopyInOtherView (pj);
  ELSE
    pj.Resize^ (sizex, sizey);
  END;
END Resize;

PROCEDURE (pj: Projection) Display* (VAR use: Graphics.PaintInfo);
(* Actual projection display is performed through attached part *)
VAR
  v: INTEGER;
  s: LONGREAL;
BEGIN
  IF pj.owner IS Window THEN (* assume this is Elements.Window *)
    v := pj.owner (Window).view; (* get window's view *)
    s := pj.owner (Window).scale; (* and scaling factor *)
  ELSE (* otherwise substitute default values *)
    v := Front; s := 1.0;
  END;
  pj.of.Draw (use, v, s, pj.pos.x1, pj.pos.y1); pj.Display^ (use);
END Display;

(* ... more code ... *)

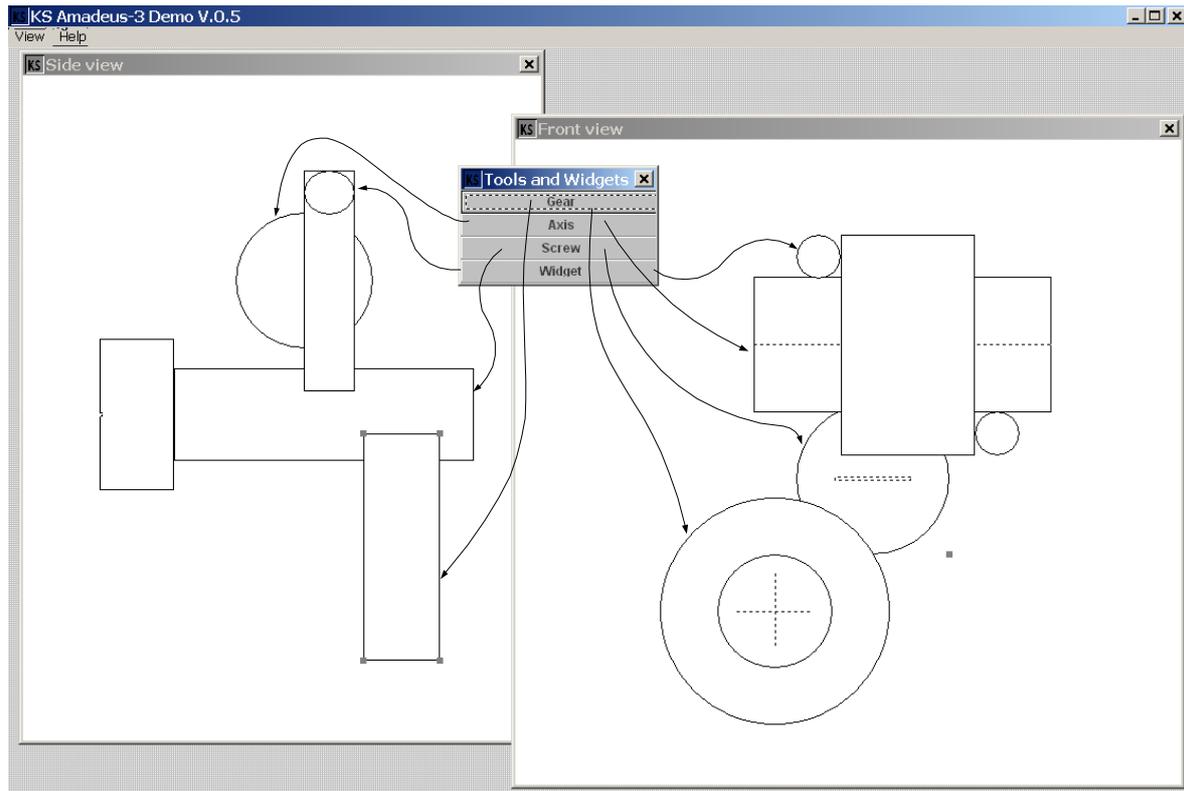
BEGIN Persist.Register (MakeWindow, windowId);
END Elements.

```



**Figure 14 - Module/Object concept for KS Demo**

Here is a screen copy showing the application running with two projections of the same objects.



**Figure 15 - Application-defined display objects**

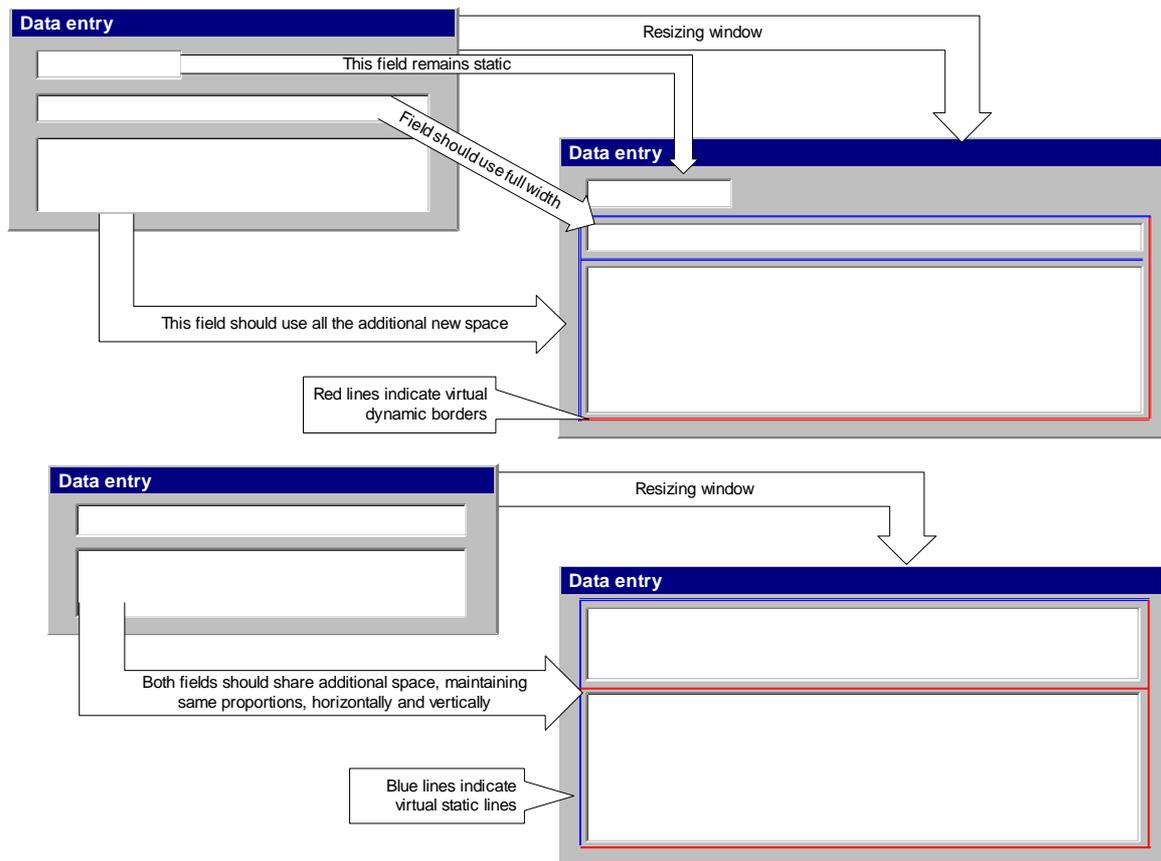
Manipulation of an object in one projection (i.e. resizing or moving the object) will immediately be mirrored by the equivalent change in the other projection window. The resize operation is object-class specific and constrained.

## 2.7 Guidelines: Advanced Display Object Placement

In addition to the absolute positioning of objects by window coordinates and child window attachments, Amadeus-3 knows another object placement and arrangement system: Guidelines.

Conceptually, the idea to allow for automatic resizing of all objects within a window when the window itself is resized. The idea appears in aswidget attachment, as found in Motif, for example, except that the Modif system is very complex and leads to some specific problems

Here are two simple cases, demonstrating horizontal and vertical attachments:



**Figure 16 - Guideline concept**

The guideline system uses precisely this idea of virtual static and dynamic lines, to which objects may attach. In fact, each object may attach to up to 4 virtual lines, which we will call “Guidelines”.

If one side of the object is attached to a guideline and not the other, then the opposite side will be moved to maintain the object dimension along the axis constant. If both sides are attached, then each will move with its guideline. If the guidelines are reverted (i.e. the one attached to the left or top side ends up to the right or the bottom of the opposite side’s attachment), then the resulting coordinates are simply switched.

Each guideline is either vertical or horizontal. You may only attach an object border to a guideline of the correct orientation, i.e. the left and right side may only attach to vertical guidelines and the top and bottom side may only attach to horizontal guidelines.

Guideline placement may be static or dynamic. The position of static guidelines is defined as their absolute offset from the left or top border. A dynamic position may be defined as a percentage offset from the left or top border *or* a fixed or percentage offset from the right or bottom border.

In addition, a guideline may also be attached to another guideline, which allows for a third form of relative placement.

The same system may be used for more complex windows with multiple guidelines, including windows containing child windows, tabbed windows etc. Up to 127 guidelines may be used. There are also 4 implicit or standard guidelines:

- -1 = Left border , offset 0 ; -2 = Top border, offset 0
- -3 = Right border, offset 0; -4 = Bottom border, offset 0

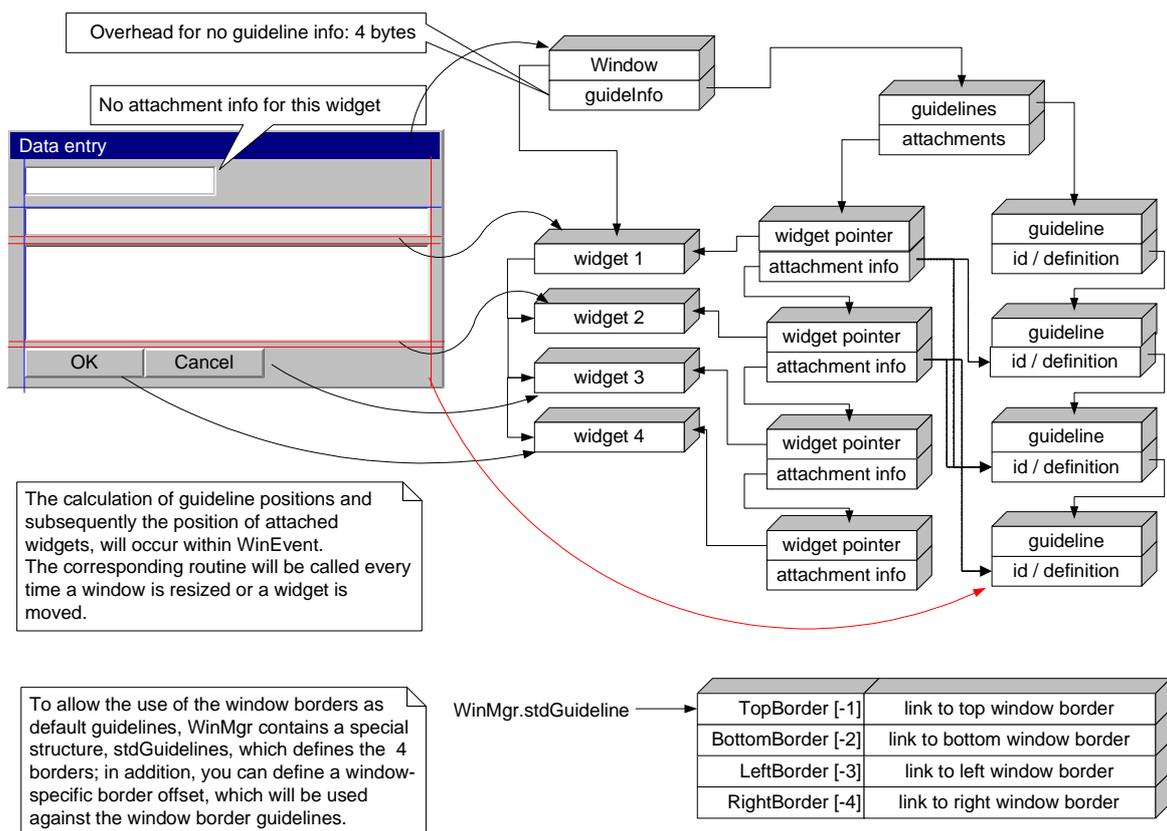
You may also center a line on the left or top line by specifying the attachment as “Center on line” for the right/bottom link.

You may define one border margin for each window which will apply to all borders, top, left, right and bottom.

### 2.7.1 Internal representation

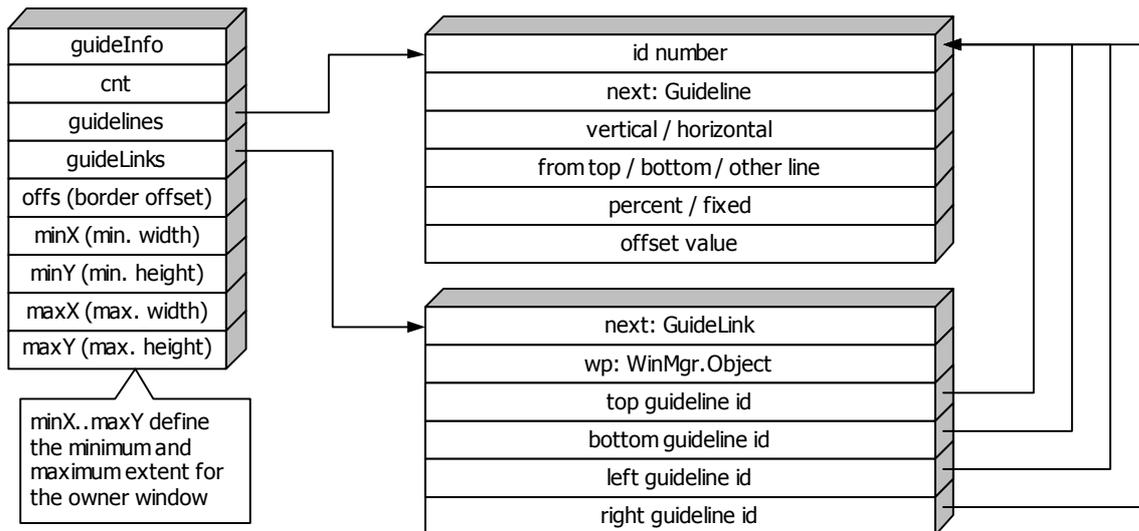
The following data structures are used to represent guidelines:

- Each window has a GuideInfo pointer, which is only used when any guideline or guide link information is attached.
- The GuideInfo object contains a counter, to track the number of actually defined guidelines, a list of guidelines and another list of Guidelinks.
- Each guideline has a unique number (0 – 127) assigned, which may then be used as reference to define attachments
- One guide link may be attached for any object in the window; this link defines one attachment for each side: Left, Top, Right and Bottom.



**Figure 17 - Guideline data structure**

Here are the detailed parameters for GuideInfo, Guideline and GuideLink:



**Figure 18 - Guideline record structure**

As you can see from the above diagram, the GuideInfo record also contains the fields minX..maxY, which are used to define the minimum and maximum size that the window may have. The user may not resize the window beyond the dimensions specified here.

As all object information, guideline information and links need to be encoded in object scripts. Here is the syntax used:

```
GuideInfo    := [ "[" minX | "?", minY | "?", maxX | "?", maxY | "?" "]" ]
BorderMargin := [ ! margin ]
Guideline   := ID [ "|" ] [ "<" | ">" ] offset [ "%" ] [ "@" attachToId ]
```

Use in Object Script Syntax:

```
OBJ Window [ window definition ]
  GLINE GuideInfo BorderMargin Guideline { ; Guideline } END
  PARTS
    { OBJ xyz [ object definition ] ENDOBJ
      [ GLINK leftID "," topID "," rightID "," bottomID ] }
  END
ENDOBJ
```

where ...ID is always either a guideline ID or "?". If no links are defined, no link record is created for the object. Guidelines may be defined even if not used.

SYNTAX EXPLAINED:

```
ID := unique number from 0 - 127 (unique for each window)
OR -1 for Top border, -2 for Bottom border,
   -3 for Left border, -4 for Right border
   -5 for Center on line; may only be used in right or bottom link
```

```
"|" := Vertical guideline; default is Horizontal
"<" := Window border or linked guideline is to the left / above this guideline
">" := Window border or linked guideline is to the right / below this guideline
offset := distance from the attachment (window border or other guideline)
"%" := relative offset, expressed as 1/10ths of a percentage of total width / height
"@ toID := attach this guideline to the guideline with the specified ID
```

Default offset style is in pixels, as absolute number. Default attachment is to the appropriate window border (according to orientation and side of attachment).

**Additional comments:**

- Decoding is handled by window, since outside the window context, the attachments have no meaning.
- By having each object directly followed by its attachment, there is no need to otherwise identify the object.
- Suppression of a guideline means that the others have to be renumbered, since the numbering should be continuous. If you edit an object script manually, you don't have to worry about this, since the decoder will take care of the renumbering automatically.

Guideline ID is SHORTINT, allowing for a maximum of 128 guidelines - far enough for any reasonable use, since it is unlikely that cutting a single window into more than 128 areas does not seem like a very good idea, from a user point of view. Besides, one could easily use multiple windows, among which to separate display objects.

What about alternate models of attachments:

Instead of attachment guidelines, one could use inter-widget attachments, as supported under Motif. Since one can create a huge number of attachment lines, there is no real advantage to inter-widget attachments, but many defects:

- one would need to identify the widget in a unique way, which would resist changes in the object script
- each attachment would have to specify to which side of the widget the attachment is made
- placements might be changed or completely invalidated when removing widgets or just by moving them around in the window

there is no apparent advantage to widget-based attachment as opposed to virtual guidelines.

Please read the chapter on “3.3.4 Viewing and editing window layers and objects”, to learn how to define guidelines and links interactively.

### 2.7.2 Scroll window columns

In scroll windows, you can attach columns (in fact the right column border) to a guideline. This column will then be dynamically resized according to the guideline constraints, except that columns must always maintain their order. If a guideline would move a column back before a predecessor column, then the attachment is ignored.

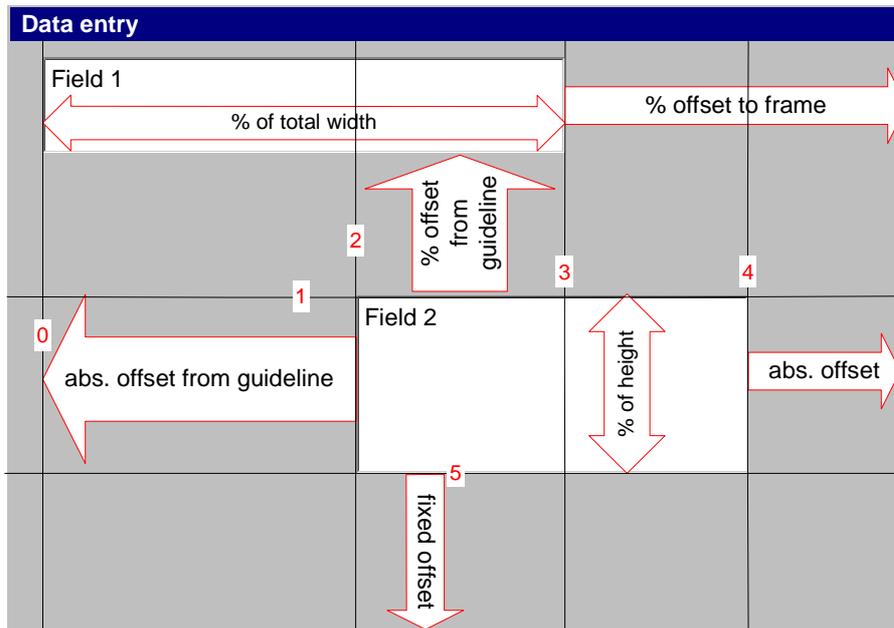
The syntax for column attachment is:

`GLCOL x`

Where “x” is the guideline index. This expression may appear within the syntax of a scroll column. The guideline must of course be specified for the window.

### 2.7.3 Guideline sample

And here is a sample of possible attachments:



**Figure 19 - Guideline options**

Guideline 0: Vertical, offset 10 (absolute) from left border

Guideline 1: Horizontal, offset 50 % (relative) from top border

Guideline 2: Vertical, offset 150 (absolute) attached to guideline 0

Guideline 3: Vertical, offset 40% (relative) from right border

Guideline 4: Vertical, offset 100 (absolute) from right border

Guideline 5: Horizontal, offset 120 (absolute) from bottom border

Attachments for Field 1: Left: 0, Top: none, Right: 3, Bottom: none

Attachments for Field 2: Left: 2, Top: 1, Right: 4, Bottom: 5

The corresponding encoding would be as follows (with unimportant details omitted):

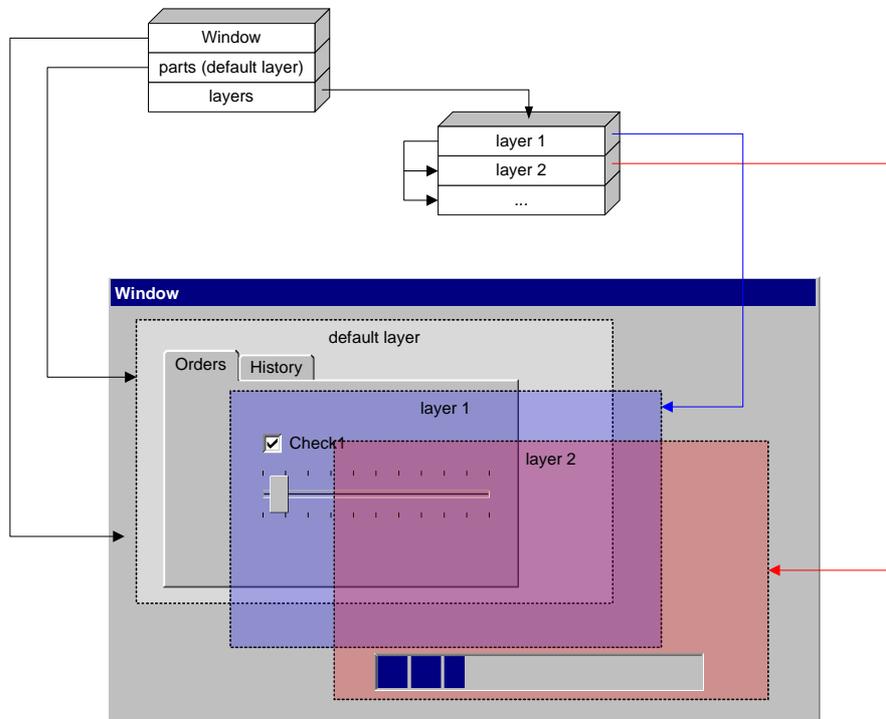
```

OBJ Window [ window definition ]
  GLINE 0 | < 10; 1 < 50%; 2 | < 150 @ 0; 3 | > 40%; 4 | > 100; 5 > 120 END
  PARTS
    OBJ Field "Field 1" [ object definition ] ENDOBJ
    GLINK 0, ?, 3, ?
    OBJ Field "Field 2" [ object definition ] ENDOBJ
    GLINK 2, 1, 4, 5
  END
ENDOBJ

```

## 2.8 Window layers

To further enhance your control over the display, Amadeus-3 provides window **LAYERS**. A layer is a list of display objects. Each layer is displayed successively, starting with the default layer, i.e. objects in successive layers will be displayed on top of the ones in lower layers, potentially obscuring them partially or entirely.



**Figure 20 - Conceptual view of layers**

The power of layers is that you can selectively activate, deactivate, display or hide them, i.e. the objects in one layer may be visible but may not react to user interface interactions or they may be temporarily hidden.

You may even activate one layer or the other, so that the same window can display an entirely different set of objects, yet still be known by the same name to the application.

NB: The elements of the default layer are always active and visible.

Layers can be edited dynamically in A3Edit, the object editor, cf. “3 THE APPLICATION EDITOR A3EDIT”.

### 2.8.1 Layer structures and methods

Layers are a new object class defined in WinMgr.ob2. They provide various methods and structures in their support: Each layer is attached to a window and may be numbered with a command code, by which they may later be activated. A layer points to a linked list of display objects (WinMgr.Object) called “parts” as in the window default layer. Each layer also may specify that it is either active or inactive, visible or invisible.

To access all the objects in a window sequentially would be quite daunting, as you’d have to go through all the layers every time, checking for access status etc. Instead, just use the following instruction sequence:

```
VAR st: WinMgr.Stepper;
...
IF w.FirstObject (TRUE, TRUE, st) THEN
  REPEAT
    (* do some processing on st.wp *)
  UNTIL ~w.NextObject (st);
END; (* if first *)
```

FirstObject accepts 2 boolean input parameters and returns the resulting stepper:

- If visible is TRUE, only objects from layers that are marked “visible” will be returned
- If active is TRUE, only objects from layers that are marked “active” will be returned
- If visible and active are FALSE, their status will be ignored
- If TRUE is returned, a first display object was found and is identified by st, cf. below
- If FALSE is returned, no display object was in a layer matching the criteria

The record st: Stepper contains the following information:

- wp : the current display object
- next : the next object that will be returned by NextObject
- prev : the previous object from the same layer
- gprev : global previous, the previous object from any layer; right after FirstObject, this contains the last accessible item from all layers in the window
- l : layer of current object
- pvL : layer of prev
- nxL : the layer of next
- active : TRUE if stepper returns only objects from active layers
- visible : TRUE if stepper returns only objects from visible layers

The following fields in WinMgr.Windows support layers:

- layers : a list of layers attached to this window
- activeLayer: the currently active layer; is used by the Window.Add and other methods

New methods for class WinMgr.Window:

- AddLayer : Adds a new layer
- RemoveLayer : Removes the specified layer and any objects it may contain
- ClearLayer : Clear all objects from a given layer; allows you to specify if the window should be updated after the layer was cleared and if cleared objects should be disposed of directly (e.g. where they consume external resources, such as bitmaps, files, database links etc.) instead of waiting for the garbage collector.

- `ActivateLayer` : Activates the specified layer by command code or by actual layer pointer; newly added elements will from then on go to the newly selected layer. The code Zero or NIL as layer pointer indicate that the default layer should be used.
- `CountObjects` : Returns the number of items of specified status (visible, active) and matches with `Object.MatchAttrib`.

New methods for class `WinMgr.Object`:

- `MatchAttrib` : If `matchAll` is `FALSE`, the `SET` parameter `attrib` must either be empty or contain at least one matching attribute or – if `matchAll` is `TRUE` - the `attrib` parameter and object's attribute must be both empty or the parameter `attribu` must be fully contained in the field attribute (but the object may have additional attributes).

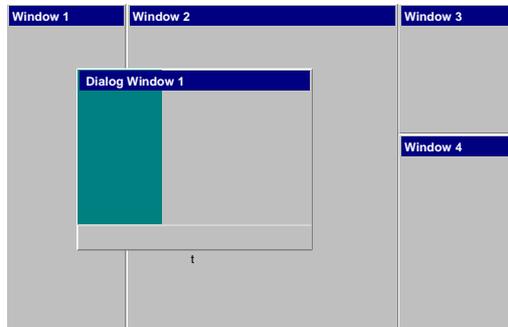
### **Object script syntax**

The optional sequence [ `PARTS` { Display object definition } `END` ] may be followed or replaced by one or more sequences of the form:

```
{ LAYER [ ID id ] [ INVISIBLE ] [ INACTIVE ] { Display object definition } END }
```

## 2.9 Window layouts

Another high-level concept for window manipulation are layouts. When several windows share a common parent window (e.g. the application main window), it is usually much more efficient to have a fixed – or at least structured – disposition for them, so that the user will always find the same information in the same location.



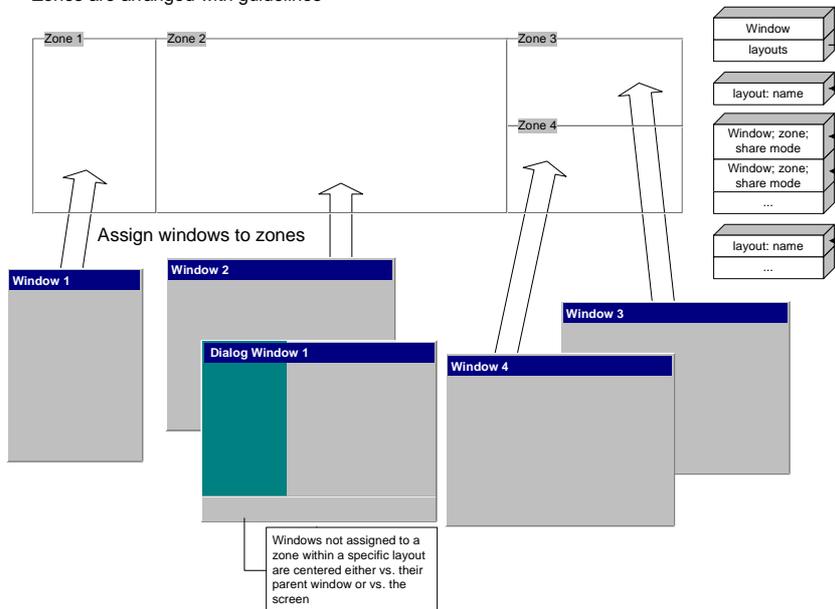
The aim of the layout concept is to achieve a dynamic, yet consistent arrangement of windows with a common parent window, e.g. the application main window.

A layout is a named set of windows in a specific arrangement for a given common parent window.

Windows can share zones with the option of overlay or vertical and horizontally split:

- the split option allows up to n windows to horizontally share a common zone, cycling through them as they are being brought to the top.
- the overlay option means each window will occupy the full zone, overlaying other windows in the same zone

Zones are arranged with guidelines



A zone is just a window object, usually invisible or - in design mode - drawn as a rectangle. May be attached to guidelines like any other object and may be named. A layout consists in assigning windows to zones of a common parent window

**Figure 21 - Window layout concept**

## 2.10 Message handling

GUIs are almost always built on the message passing paradigm, therefore you will have to understand it fairly well. You will have to know messages will be passed around, who will receive them and how you can respond to them.<sup>8</sup>

The basic structure of a message-based GUI program is the following:

```
Loop
  Get message
  Program finished ? YES: EXIT
  dispatch message to specified window
End
```

Windows may have a message handling procedure that may or may not be called directly by the GUI. Widgets under Motif always have to register a call-back procedure to notify the application program of their activity.

Amadeus-3 handles most of these aspects for you, and it also handles most messages that are not of interest to your application program. The dispatching procedure is embedded in the object class `Startup.Project`, under the method name `MainLoop`. The application can rely on the fact that all messages will be sent straight to the proper objects through their specific methods.

The dispatching of messages is quite simplified. In fact, all messages pass through `WinEvent.MainWndProc`. This procedure attempts to find the window for which a message was intended. It will then substitute the actual window object for whatever the GUI used to identify the window, typically just a handle. After calling `Events.Translate`, which returns a detailed event record, it also filters many standard messages, in particular those concerned with resizing and moving windows. This is done in `WinEvent.BasicHandler`. Then several other handlers get a chance to inspect the message. Informally, imagine the event record as a box being passed around between various services, as in this little informal representation in the next figure:

---

<sup>8</sup>Surprisingly (depending on what you expect), this is exactly the kind of general overview that is totally absent from Microsoft's Windows SDK. Every message is explained in much detail, but there is no general overview explaining how the messages are related and how they depend on each other. This omission can cause all kinds of trouble.

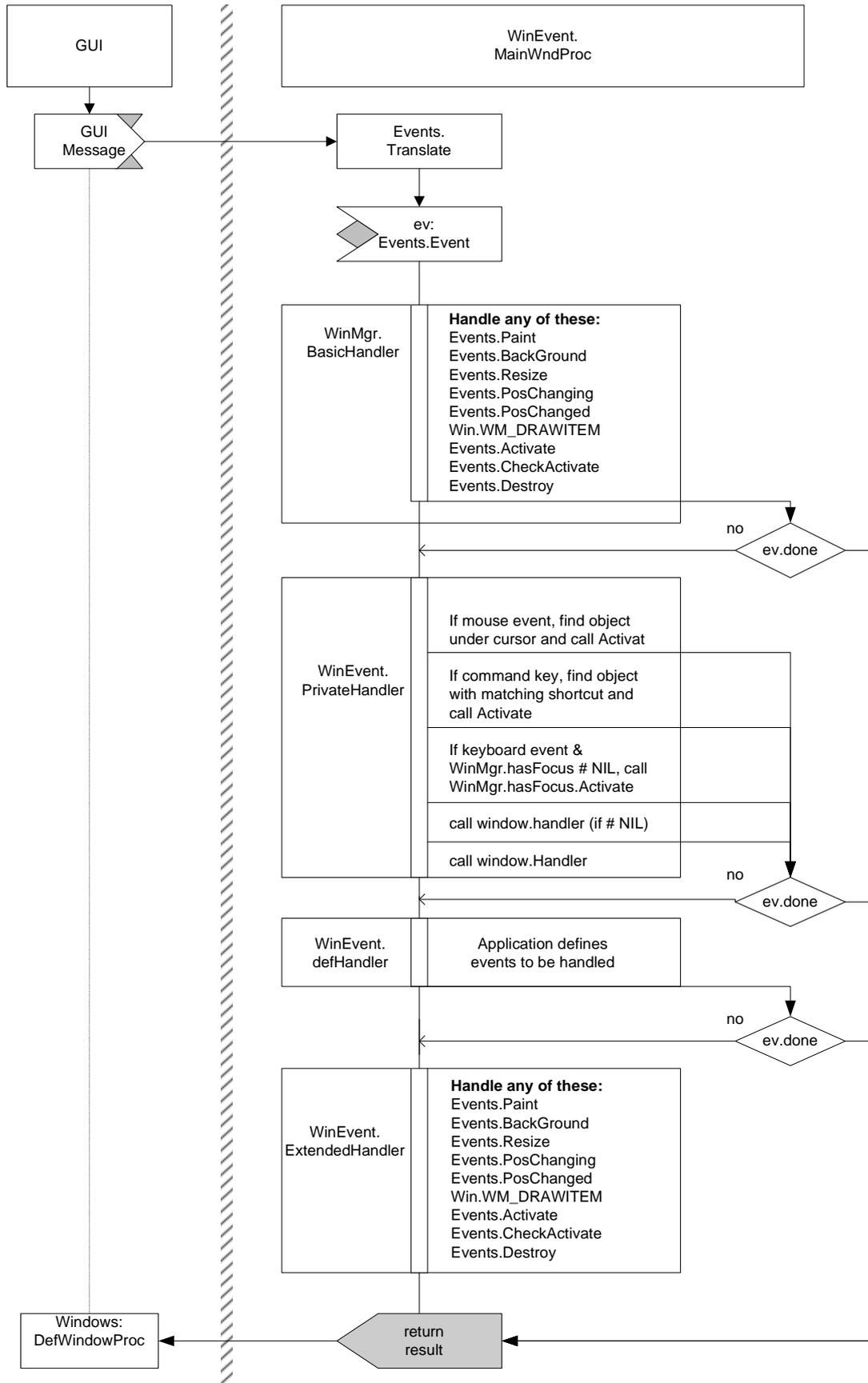


Figure 22 - Message handling

Let's express the contents of this diagram in words:

- The GUI sends a message to a window handler procedure. Amadeus-3 substitutes the procedure `WinEvent.MainWndProc` for this and finds out which window object is meant to receive this message.
- A number of basic messages is directly interpreted by `WinEvent.BasicHandler`, which may be intercepted by re-assigning the variable `WinEvent.basicHandler`. Remember though that even if you do re-assign this procedural variable, you always *must* call the procedure `WinEvent.BasicHandler` at the some point within your replacement routine, as otherwise a lot of services will not be provided. Some of the things that `BasicHandler` does is update fields contained in a window object after move or resize messages from the GUI.
- Then mouse- or keyboard-related messages are sent to any potential receiver display object, via the method defined as `WinMgr.Object.Activate`. Receivers may be: object(s) underneath the mouse cursor or - for keyboard events - the object that currently has the focus, if any; the focus owner object is defined by the variable `WinMgr.hasFocus`. The `Enter` key is sent to the first object with the attribute `Default`, or – if none found in the target window – to the focus owner. For mouse events, each object underneath the mouse cursor is activated, starting at the top-most one, until `event.done` is set to `TRUE`.
- Next the application-defined handler procedure, `WinMgr.Window.handler`, if any is assigned to the receiving window, gets it's turn. This handler may be empty or re-assigned during program execution and is not bound to the Window's class but to a specific instance of a window. It is used to check for command messages, such as those generated by menu selections or command buttons and other such event sources, as well as for any messages that should be handled by this specific owner window and not a whole class of windows. Obviously, several windows – even of different classes – may be assigned the same handler procedure.
- Then control is passed to the receiving window's handler method, which takes care of class-specific message handling. As a guideline, you should create a new window class whenever there are several windows showing similar behaviour within your application or you want to create a re-usable type of window with specific behaviour. A typical example is the class `Scroll.Window`. The behaviour of scrolling windows is very specific and easily made re-usable. You will also want to create a new window class, when you wish to alter it's way of handling display objects or it's own `Paint` method.
- The global default handler, found in `WinEvent.defHandler` and that may be set by the application, is next. It is used to handle system-wide commands and conditions, such as application termination and main menu selections.
- Finally, if no other handler consumed the message, `WinEvent.ExtendedHandler` will be called to clean up some things, such as reset the mouse cursor shape, if required.

Note for windows containing several child-windows:

If the message was a command (such as sent by a button or a menu) or a command key (« Alt- » key combinations), a child window will broadcast this command to all it's siblings as well as it's parent, until one of them consumes the command. This is particularly useful where groups of windows are combined so that one window is the actual receiver of messages, but the messages may be generated by buttons or menus attached to another window in the same group. It means these messages still get to the proper destination, but there is no need to dispatch them through the entire list of open windows.

### 2.10.1 What messages does your application have to handle?

As you may guess from the above diagram, you almost never have to handle anything else but keyboard input, mouse actions and commands - which makes your life pretty comfortable. The corresponding messages and defined fields in the event record are the following (see module Events):

Event type ( <code>ev.tp</code> )	Action ( <code>ev.action</code> )	Description	Other fields
<code>Events.Key</code>	<code>Events.Press</code> <code>Events.Release</code> <code>Events.Character</code>	Key was pressed Key was released Key encoded as character	<code>ev.key</code> : actual key code, cf. module Keys
<code>Events.Mouse</code>	<code>Events.Press</code> <code>Events.Release</code> <code>Events.Move</code>	Mouse button was pressed Mouse button was released Mouse moved	<code>ev.button</code> : <code>Events.LButton</code> <code>Events.RButton</code> <code>Events.MButton</code> <code>ev.px</code> and <code>ev.py</code> : Coordinates of mouse cursor
<code>Events.Command</code>	--	Command was sent (e.g. through a menu selection or a button)	<code>ev.code</code> : Command code, cf. Module Commands

### 2.10.2 Other messages that you may wish to handle occasionally:

The following events only need to be handled occasionally:

Event type ( <code>ev.tp</code> )	Description
<code>Events.Close</code>	The application will be terminated, except if you handle this message and set <code>ev.done</code> to <code>TRUE</code> . Cleanup functions, such as saving the application status etc. may be implemented by overriding the method <code>Project.Cleanup</code> . Any modules that are not application-specific should install their cleanup procedure with <code>Endup.Install</code> .
<code>Events.Resize</code>	A window was resized. You may wish to take special actions, such as re-arranging objects contained within this window. Often you do not have to implement special actions, as objects contained within the concerned window may automatically adjust to the new window dimensions
<code>Events.Move</code>	Window was moved. See comments on <code>Events.Resize</code>

### 2.10.3 Command codes

Command codes are one of the basic types of event that an application must handle. Each command code corresponds to a specific global, application-specific or context-specific event and is generated either through a user actions, such as selecting a menu option or pressing a button, or through a program, by sending the command code directly to an application's event queue.

Command codes are basically just numbers, which arrive together with a code that identifies the event type as being of type "Command". In Amadeus-3, a command event is represented by an `Events.Event` record, which has its type field "tp" set to `Events.Command` and the id of the command in the "code" field.

There are pre-defined GUI commands and application defined commands. The module `Commands.ob2` manages application-defined command codes:

- Each command code is a LONGINT number, associated with a string, i.e. each command is named and each has a unique identity, although several may define the same external value.
- There is a standard command table, `Commands.std`, which contains some standard codes that can be universally used, although they will have to be context-specific. These commands are always available. In the editor A3Edit, they are available in a separate window for all projects. Here is a list (might be expanded):
  - OK, Cancel, Abort (dialog masks, any data entry screen etc.)
  - Yes, No
  - Add, Edit, Delete, Insert
  - Print
  - Copy, Cut, Paste, Undo
  - Exit
  - Next, Prev, Up, Down
  - HelpIndex, HelpOnFocus, HelpOnHelp, HelpContents
- You can create multiple command tables, which can each be assigned a unique base code. Each table can define up to 10'000 unique codes, minus 100 standard codes (0-99), which are reserved for the standard command table. The base code is combined with the command number to form a number that is unique to the entire application. If your table's base code is 1, then the first definable command value is 10100.
- You can search for commands by name and find the name of a given command code.
- When you use the code generator, it will generate a constant for each command code. The name will be the command name plus the extension "Cmd". The value will be the command's unique number.
- Each project (cf. `Projects.Project`) has an associated command table, i.e. project can define its own local commands, which will not conflict with any other commands used inside an application, if each table is given its own unique base table ID.
- Inside the A3Edit user interface designer, you can use command codes to generate buttons and you can associate them with menu options, as IDs for display objects and more.

#### 2.10.3.1 Command ID vs. Command value

As specified, several commands can define the same external number, hence the number itself is not sufficient to identify a command uniquely. So to store a command code, we need an index into a table. Command codes are stored in a hash table, so the value that servers to represent a command code is a negative value, i.e.  $-(\text{command index into hash table} - \text{command table base} - 1)$ .

If you find a positive number, it's the command value; if it's a negative value, you first need to find the matching command table via the base number and then use `Commands.CmdTable.GetCommand` to transform it into its command value or use `CmdTable.ConstantToStr` to return the command name.

### 2.10.4 Timer Events

The timer represents a special category of event source. To receive timer event, you have to install a timer explicitly. Timers may be a restricted resource on some systems (in particular MS-Windows), so use them only where required.

To request timer events, your application must call the following function:

```
Events.StartTimer (wh: Graphics.WinHandle; id,unit: INTEGER): BOOLEAN;
```

Where `wh` is the receiver window handle,  
`id` is the number of the requested timer  
`unit` is the interval unit in microseconds

The window handle is usually `WinMgr.mainW.hWnd`. Only specify another handler if you wish processing of timer messages to be handled specifically by a certain window.

Timer events will be sent to the receiving window in the specified interval units of time with the event type (`ev.tp`) `Events.Timer` and the timer id in `ev.code`.

When you no longer wish to receive timer events, call:

```
Events.StopTimer (wh: Graphics.WinHandle; id: INTEGER);
```

You should always call this procedure for each timer that was started before leaving the application. Optionally, you may want to install a termination procedure using module `EndUp`.

Timer events may be useful in many situations. You may just wish to actually keep track of the time, or you may use them to implement repeating events.

To get a new, unique value for a timer ID, call:

```
Events.TimerID* (VAR id: INTEGER);
```

#### Example:

An application allows the user to drag an object within a window. He may not drop the object outside the target window, but the latter may have to be scrolled to be entirely visible. You can use timer events to implement a continuous scrolling effect as soon as the mouse reaches the edge of the target area. This type of operation is implement in module `Actions` and you may try it in `A3Edit`.

### 2.10.5 Power Management

In various circumstances, you may wish to react to power management events, especially when working with databases. When the system powers down or goes into standby mode, it is advisable to close open databases or disconnect from remote ones. There may be various other reasons, why you may wish to react to changes in the power state of the computer. If the parent OS supports this, you should receive one of the following messages, which you will probably want to intercept in your default message handler:

```
ev.tp = Events.PowerMngmnt message concerns power management function
```

Event action ( <code>ev.action</code> )	Description
<code>Events.Suspend</code>	System is being suspended (Standby, Hibernation...)
<code>Events.Resume</code>	System resumes operation
<code>Events.CrashResume</code>	System crashed without proper termination and is now resuming operation

## 2.11 Generic sequential data access

Module `Sequence` defines the underlying architecture for generic sequential data access, which is used for generic scrolling and other operations. It is often useful to have a generic method to access sequential data, without having to worry about the underlying data structure. The requirements are:

- the data is truly sequential
- each element can be identified with a unique value
- each element can be found by its unique identifier
- it is possible to find the preceding and following element of the sequence with any identifier

The following extensions will make a sequence even more useful:

- the total number of items in the sequence can be found quickly
- the index within the sequence can be determined quickly with precision

These functions are required for generating scroll bars, when displaying lists of elements in a window. “Quickly” is an important specification, as it is always possible to count the number of elements in a closed sequence and the index of an element can equally be found by stepping through the sequence, but in many circumstances, this is inefficient or at a totally prohibitive cost in terms of execution time. If your database has 500’000 elements and it is used on a network, you can’t use a sequential count to determine the number of its elements every time you need to refresh the scroll bar and the thumb<sup>9</sup> position.

A sequence may also provide a mechanism to tag elements, which must obviously be provided by the instantiation module. Tagging is useful when a variable number of elements needs to be selected from a larger set. A good example is the selection of files from a list for copying or deletion. There are many possibilities for marking elements. One possibility is to include the tag with the element itself. Another one is to use a dynamic array of SETs, which are projected onto the data structure.

Finally, you may want to use a sequential data structure from within your application without having to manage all the crummy details. The class `Sequence.Stepper` is made just for this. It allows you to step forward and backward through a sequence without having to declare the usually required temporary variables. You may also use steppers from within reports (cf. “”).

There are several standard sequential data structures with fully defined access methods including:

- Database file access ([Module DbView](#))
- Memory lists ([Module ListView](#))
- Persistent object groups ([Module GrpView](#))

---

<sup>9</sup> The little box that indicates the position of the current element within the total range

## 2.12 Scrolling

Scrolling through all sorts of data is of major importance, as one of the most important visualisation methods. To list the basic requirements for a decent scrolling system:

- Any data structure with some form of sequential access method may be used as basis for a scroll list. This access method will be made accessible via a class derived from `Sequence.Source`, cf. “2.11 Generic sequential data access”.
- Once the access method is defined, a display format for the data to be represented must be found. In text-only systems, this was fairly easy, as the only option was some kind of string representation. In a graphical user interface, this is no longer an acceptable restriction.
- Elements in a scroll list may be graphical objects, strings may use various fonts etc.
- The line height may be variable, as scrolling automatically adapts to individual line height
- At least a minimal range of editing functions is directly supported, as scrolling lists are often used to enter or modify data items found in the sequential data structure used to generate the scrolling display.

This is exactly the range of possibilities supported by **Amadeus-3**, as implemented in module `Scroll` and `Sequence`.

You may think that with all these possibilities, it’s not a piece of cake to produce a fully functional scrolling display. In fact, it is not nearly as hard as you think. First of all, **Amadeus-3** delivers most of the common data access methods that you are likely to use.

Secondly, the task of displaying variable height data items is very much simplified by the fact that each scrolling column uses a standard display object to perform the task of formatting and displaying properly formatted information. This means that you may use any bitmap, icon, value display or whatever other standard display object as column element in a scrolling window.

How do you go about assembling these different elements to produce a scrolling display? Here is an outline of the required steps, followed by a sample of actual code:

1. Create or import a window of type `Scroll.Window`
2. Create a data source of appropriate type, either based on one of the standard sources (cf. above) or application-defined.
3. Create and assign a data structure of the proper type to the data source.
4. Assign the data source to the scrolling window.
5. Create at least one display element and assign it to the scrolling window.

The window may now be displayed, it is ready for use, cf. below.

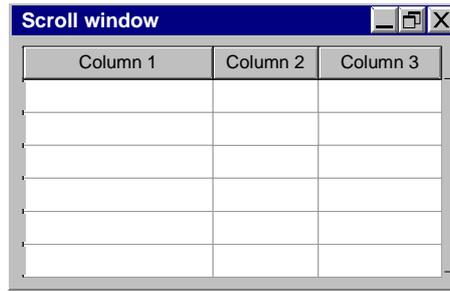
There are some extended scroll functions which are not explained here. Please refer to the corresponding chapters instead:

- Multiple line selection (7.32 `Sequence`)
- Drag & Drop from scroll lists (2.13 `Drag & Drop, Amadeus-Style`)

Module `Scroll` defines the following additional window attributes:

- ◇ `ContEdit` : continuous editing, i.e. after an element is added, prompting continues
- ◇ `HighlightOff` : Selected line is not highlighted
- ◇ `OnlyWithFocus` : Selected line is highlighted only if window has focus
- ◇ `HadFocusOnce` : Not persistent! Remembers if focus has been set to window before
- ◇ `TitleResize` : Allow the user to resize the height of the title line interactively
- ◇ `ColResize` : Allow the user to resize the width of columns interactively
- ◇ `KeepMask` : Attached editing mask is kept visible after add or update was performed
- ◇ `DontHide` : Attached editing mask is not hidden when scroll window is hidden

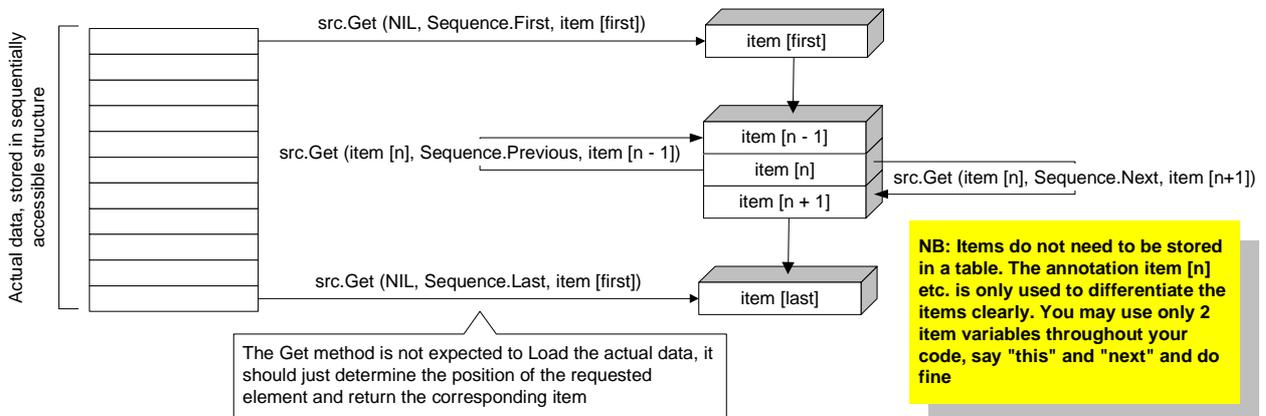
The typical application for a sequential source object is the use with a scroll window. Other uses are possible and desirable!



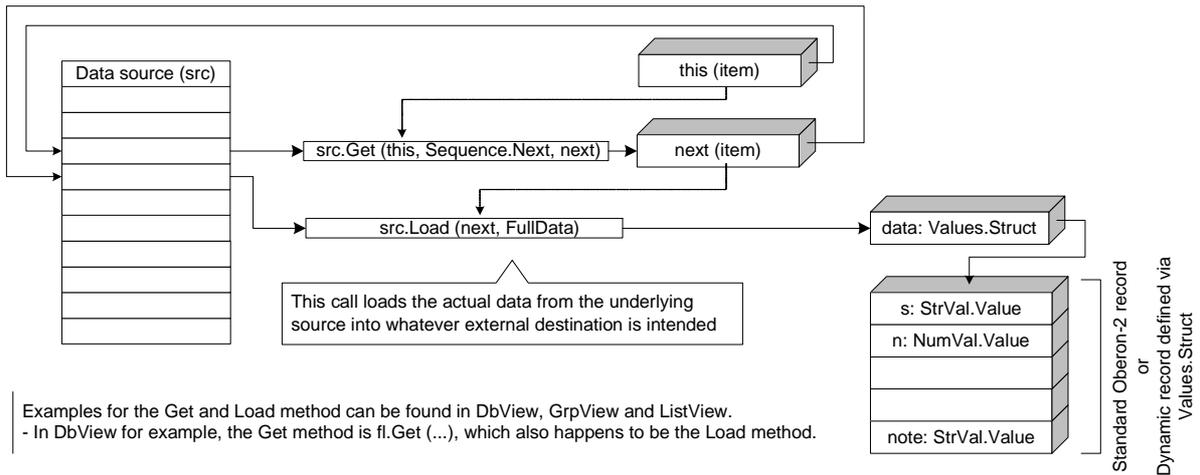
The contents of a scroll list is built based on data returned by the underlying data source in the form of a Sequence.Source

The basic concept behind Sequence.Source is to provide a unified interface to any form of sequential data structure, so that - based on some specific piece of information - exactly one element of the data structure is identified and that from such an item of information at least the next and the previous element may be found. It is also assumed that at least the first and usually also the last element may be accessed directly. Good candidates for such a data source are: database files with sorted index, sequential files, memory list, etc.

The data access source is a structure that identifies the actual source of the data and its state, that defines various access methods (SetStart, Get, Load, Store etc.) and that is associated with another object class, the Item, that stores the position of any element in the source data structure.



After the Get method used to locate data items, the next important method is Load, which transfers data from the underlying source into some other storage, such as global or local variables for further handling. A very typical use would be to load variables connected with Values.Value objects, which makes them easily accessible for use with Amadeus-3 user interface, e.g. in data entry, scrolling lists and many more.



Examples for the Get and Load method can be found in DbView, GrpView and ListView. - In DbView for example, the Get method is fl.Get (...), which also happens to be the Load method.

Figure 23 – Sequential Source and Scrolling explained

Here is a view of the execution flow during a scroll window update. You don't really have to understand this in detail, unless you want to modify low-level elements. Usually, it is quite enough to supply display elements for the columns and a data access source.

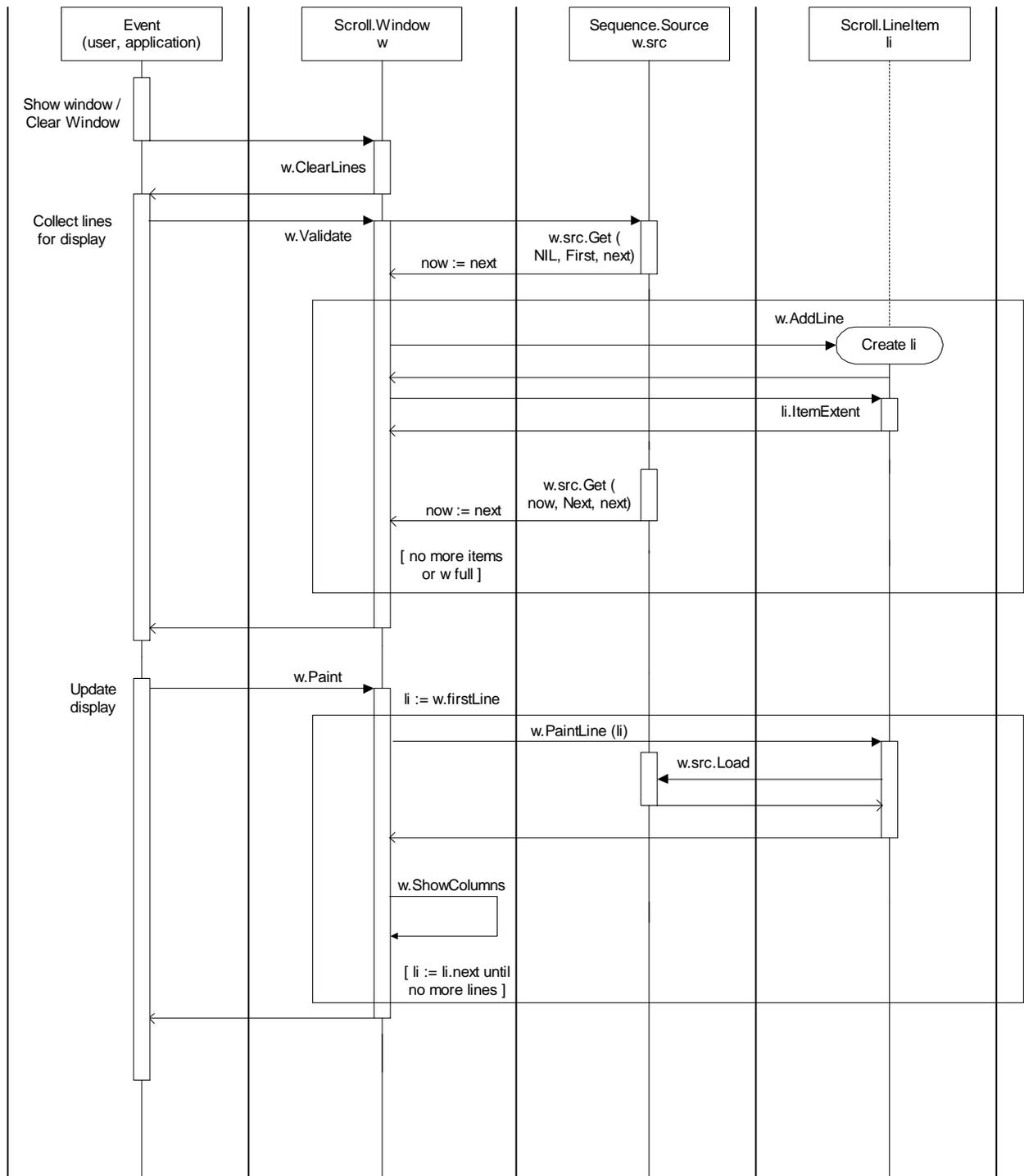


Figure 24 – Scroll window update mechanism

Here is a sample implementation of scrolling:

```

MODULE ScrollDemo;

IMPORT Scroll, MemList, ListView, StrDisp, WinMgr;

CONST
  MaxLen* = 80;
TYPE
  String* = ARRAY MaxLen OF CHAR;
  StrObj* = POINTER TO StrObjDesc;
  StrObjDesc* = RECORD (StrDisp.ObjectDesc) END;

VAR
  data : MemList.List;
  current: String;

PROCEDURE (sd: StrObj) GetString (VAR s: ARRAY OF CHAR);
(** Method assigns global list access variable to the display string *)
BEGIN COPY (current, s);
END GetString;

PROCEDURE Show* (w: Scroll.Window);
VAR
  src: ListView.Source;
  sd : StrDisp;
  b : BOOLEAN;

  PROCEDURE AppendData (s: ARRAY OF CHAR);
  BEGIN MemList.Append (data, s);
  END AppendData;

BEGIN (* Show *)
  IF w.src = NIL THEN (* scroll window not initialized *)
    IF data = NIL THEN (* data list needs to be created *)
      NEW (data); data.Init (MaxLen);
      (* add some initial data *)
      AppendData ('1st line'); AppendData ('2nd line');
      AppendData ('3rd line'); AppendData ('4th line');
    END;
    (* create the data source and assign list and data *)
    NEW (src); src.Init; src.Assign (data, current);
    (* assign the source to the window *)
    w.src := src;
    (* Now add a display object *)
    NEW (sd); sd.Init; sd.width := MaxLen;
    b := w.AddCol (sd, 0);
  END; (* if w.src *)
  w.ClearLines (NIL); (* reset to first line *)
  w.Show (WinMgr.TopPos); (* display window now *)
END Show;

BEGIN data := NIL; (* make sure it's initialized *)
END ScrollDemo;

```

### 2.12.1 Analysis of Module ScrollDemo

There are a lot of things implicit in the above code. You are invited to consult all the quoted modules to gain some deeper understanding of the ramifications. For now, let's just look at those features used in this module.

As you may notice, we pass the window to be used for the scrolling action is passed as a parameter from the outside. This means we assume that the window already exists and all its basic parameters, look and feel etc. are already defined. This is usually the case, as this will typically be window generated with the help of `A3Edit` (cf. related documentation). So we really can concentrate on the contents of the scroll list.

A `MemList.List` is a double-chained list, which stores any kind of data by simply moving it in and out of freely allocated buffers. This makes it suitable for any static data (such as simple types, strings etc.), but not for objects and other items containing pointers, at least when a garbage collector is used!

Usually, the receiving data buffer is passed as parameter when accessing the list. For the scrolling methods to work independently, we have to specify a fixed area of memory - here variable `current` - that will be used to access data during the scrolling process. We give the scrolling source this information on initialisation, i.e. in `src.Assign`.

Now to the implicit knowledge: Whenever there is a request to display a given element from the data source, this variable will be changed. When `StrObj.GetString` is called, variable « current » will contain the string stored at the line currently requested for display. To make sure that the variable - if used elsewhere - is not affected by this process, the data source object will automatically save and restore this variable whenever it has to access it. That means that if the variable « current » is used for data entry, for example, it will not be affected by the scrolling update process. It *will* be changed when there is a request to edit the corresponding variable (eg. bound to a `Fields.EditField`) through the method `Source.Load`.

The `StrObj` display object that we generate is now able to do its job, always returning the string currently requested by the data source. All the display operations are taken care of, all we have left to do is assign the result string. It's the only display column that we assign to the scrolling list.

We could also add a header by using the window's standard `w.Add` method with a static object, such as a `Fields.Label` string or a `Controls.Button`. This header would then be displayed statically over the assigned column and - if shown as an enabled button - could be used to select the sorting order of the scrolling display, as an example of possible ideas.

You may also notice the difference between `StrDisp.Object` and `Fields.Label`. The latter is really nothing else but a fixed string. The former may be changed dynamically according to any criteria or calculation on each call to `StrDisp.GetString`.

By now, you probably start seeing the possibilities:

- Instead of a simple list of strings in memory, the data source could be a database table
- The display objects could then be of type `ValueDsp.Object`, which allow the display of variables retrieved from a database file.
- Reading module Sequence, you will find that a scroll source may define filters and starting values, giving great flexibility as to the subset of data that will be included in a given scrolling view. This subset is actually defined by the data source structure, which may also be used in other places, such as reports etc.
- There is nothing to stop you from representing certain values found in the data source through bitmaps, icons or your own, application-defined display objects. As mentioned before, if multiple scrolling columns are defined, the elements do not have to be of the same size or height. Module scrolling is able to adjust for variable-height elements without any further help, even if the height varies from line to line.

More flexibility would be hard to get. Please observe that through sufficient analysis, we managed to break down the entire structure to a level where we need absolutely no multiple inheritance to

achieve this goal. We even gain a lot of flexibility by not having to stick with too extensive composite objects.

It is true though that it takes some getting used to program in this way, so we urge you to practice a few times. You might also guess that having to do this kind of thing all over all the time would require a lot of overhead, not to mention the time and effort required...

This is where you have to start realising that in a framework such as **Amadeus-3**, you don't just use ready-made elements. You can custom-tailor further tools that will suit your application needs precisely and over which you have full control. Once this is done, you may be able to reduce the creation of a new scrolling list which follows the application standards to no more than 1 or 2 lines of code. You will find examples of this in further demo code.

### **2.12.2 Enhancing Scroll Window Presentation**

You can enhance the scrolling window display in various ways.

- Adding column titles  
This is a standard feature: you may add titles to each column; you will find the necessary calls in Scroll.ob2 and of course, you can do it in A3Edit through simple drag&drop. Adding buttons is a good idea when you want to allow the user to sort by various columns. By allowing some buttons to be activated and making them "sticky", the one above the currently selected sorting column can be made to look like it is pressed.
- Adding standard display objects to the window's background  
A scrolling window is a standard window, which also supports the scrolling functions, so you can obviously add standard objects to it, which will be displayed and behave just as usual. Add a bitmap for an improved look or any other display elements, but remember to ensure that they don't conflict with what the user needs to see to properly operate the scroll list.
- Changing the Décor – the background of a scrolling window  
Each scroll window may be linked to a décor object, which will change the background for the table or for each line separately. The standard decor is a grid, which displays vertical and horizontal lines for each line and column. Another one, from ScrollDecor.ob2, is the Ledger Style, as commonly used in financial applications, with 2 alternating colours.
- Changing colour and font for individual lines  
You can attach a procedural variable via the field getColour to the window that is called before each line is displayed. This procedure may set the current brush, pen and font, allowing a fine-tuned adjustment for each line, according to such things as multiple selection etc.

### **2.12.3 Other functions**

More functions are available:

- The signal procedural variable is called each time a change occurs in the contents of the list, such as when a line is inserted, deleted or when the highlight is moved, allowing your application to take direct action.
- handleEdit is called when default editing functions are used, such as Add or Insert
- delOk is called before a line is going to be deleted.
- handleDel is called every time a line is actually deleted
- grab is called when the user initiates a Drag&Drop operation from within the list, such as Click the right mouse button, then drag the contents of the current line to some area where you can drop it.

### **2.12.4 Item tagging (selection)**

You may use multiple selection with Scroll, but your application has to supply a little assistance. The information about the selection status of each item has to be made available through the Sequence.Source that is attached to each scroll window.

The standard methods that an extended class may provide are:

- PROCEDURE (src: Source) SetTag\* (itm: Item; set: SHORTINT): BOOLEAN;  
Marks an item as tagged
- PROCEDURE (src: Source) GetTag\* (itm: Item): BOOLEAN;  
Returns the current tag status of an item
- PROCEDURE (src: Source) AnyTagged\* (): LONGINT;  
Returns TRUE if any item is currently tagged in this source
- PROCEDURE (src: Source) ClearTags\* ();  
Clears any tags for this source

DbView supplies a standard mechanism for tagging elements: if you call SetTagId with a LONGINT variable that should be used as Tag ID, then the current tag status of any item will be stored in a ARRAY OF SET, i.e. one bit per element.

NB: SetTagId needs to be called only ONCE in your application. From that point on, the scroll source will accept tagging of elements.

If your database is very large, you may still want to use a different system. Alternatively, you could define a boolean variable for each record element, which would have to be updated each time you change a tag. Such a system would be persistent, slow and independent of the database volume.

Module Scroll supports the use of Ctrl+Left mouse click to tag/untag multiple lines. You may also use Alt+Space to tag/untag lines.

### 2.12.5 Example of GetColour procedure

The following example should illustrate how an application may use the Scroll.Window.getColour procedure to assign colours dynamically. In this case, only the text colour is being changed. In the same way, the background colour could be changed as well. The sample code should be quite understandable:

```
PROCEDURE GetDocColour (w: Scroll.Window; li: Scroll.LineItem;
  VAR br: Colours.Brush; VAR fg,bg: Graphics.RGB; VAR mode: Fonts.BkgMode);
BEGIN
  Scroll.StdColour (w, li, br, fg, bg, mode);
  IF App.doc.state = App.StateExpiredOpt THEN fg := Colours.Blue;
  ELSIF App.doc.state = App.StateRevokedOpt THEN fg := Colours.DkGray;
  ELSIF App.doc.state = App.StateMissingOpt THEN fg := Colours.Red END;
END GetDocColour;
```

## 2.13 Drag & Drop, Amadeus-Style

Amadeus-3 supports a special flavour of Drag & Drop operations, which are not bound to the operating-system. They are excellent for supporting application-specific D&D operations and are much easier to implement than what you get with most GUI systems. In addition, you may drag any object, even large bitmaps, full-fledged windows or any other valid display object. So far, only NextStep could do this!

What's even better is the fact that D&D operations are almost easier to implement than any other form of sophisticated interface. You may or may not have to specify a way of grabbing objects (in the case of standard display objects, it's just a matter of setting a flag) and a acceptance procedure (again, for standard display objects you just have to set a flag on the receiving window).

Your application can generate draggable objects on any kind of user action. The user may then drop the corresponding object(s) on any window that accepts dropped objects. From there on, control reverts to the standard window method for accepting objects. In this way, the drag&drop operation can be completely decomposed into 2 separate components:

- Generating an object
- Accepting an object

There is no implicit or explicit mention of the fact that there is any drag&drop operation to be performed.

This allows your application to generate objects and send them directly to the receiving method without any user interaction, i.e. the D&D process is entirely managed by Amadeus-3 and is totally transparent for your application.

### 2.13.1 How it works for the user

The user should always find a familiar interface. Therefore, all D&D operations should be initiated in the same way. The Amadeus-3 standard was inspired by the OS/2 Workplace shell system.

To drag an object:

- grab an object with the right mouse button to start dragging it
- dragging the mouse while keeping the right button pressed will drag across the screen
- release the mouse button when the object is on top of its destination

To grab a window:

- You can grab a window like any other object, but it may not always be accessible, because it can be covered up by display objects; therefore, you can grab a window by pressing Ctrl+Shift+Right-Click, even if the click occurs over a display object

Modifiers may be applied for special effects, in particular:

- pressing the « Ctrl » key while performing the grab operation will make a copy of the object to be dragged instead of moving the original object (where allowed)
- pressing the « Shift » key while performing the drag operation will avoid the alignment to the destination window's grid, allowing pixel-precise insertion of the dropped object.

### 2.13.2 Other standard mouse operations

By simply setting an appropriate flag on a display object, you also make it available for move and/or resize operations. Initially, the object to be manipulated must be selected (another set of flags, here: `WinMgr.Selectable` and `WinMgr.Selected`), except for drag&drop operations. This usually implies that the user clicks the left mouse button on top of the desired object. When selected, an object's corners are highlighted with grey rectangles.

Usually, whenever a new object is selected, any already selected object is de-selected. To select multiple objects, the user may do either of the following:

- Press the « Ctrl » key while clicking an object with the left mouse button; this way, any objects already selected will not be de-selected.
- Press the left mouse button while the mouse cursor is outside the area of any display object, i.e. over the window's background, then press and hold the left mouse button and start dragging the mouse in any direction. This will display a « rubber band » box, which you can stretch to include any objects you wish to select. When you release the mouse button, any object *fully* contained in the outlined area is marked as selected.
- Whenever more than one object is already selected, clicking additional objects will *not* de-select the previously selected ones but will add to the number of selected objects.

When multiple objects are selected, they may be manipulated as a group for the following operations:

- Moving
- Drag & drop

The Resize operation is still only applied for one object at a time.

When moving the mouse over a selected object, the mouse cursor changes its shape to indicate available actions. Over the edges and corners, a double error indicates the possibility to resize the object. Over the central part, a crossed double arrow indicates that the object may be moved. To perform any of these actions, the user must simply click and hold the left mouse button and start moving the mouse in the desired direction.

### 2.13.3 Internal architecture for standard mouse operations and Drag & Drop

Here is an overview of how Amadeus-3 supports standard mouse interaction such as moving and resizing objects, selecting multiple objects and drag & drop operations.

Actions.OB2 is the central module. It makes use of standard display objects (WinMgr.Object) and windows (WinMgr.Window), along with events (Events.Event) to achieve the desired results, i.e. interactive display object manipulations.

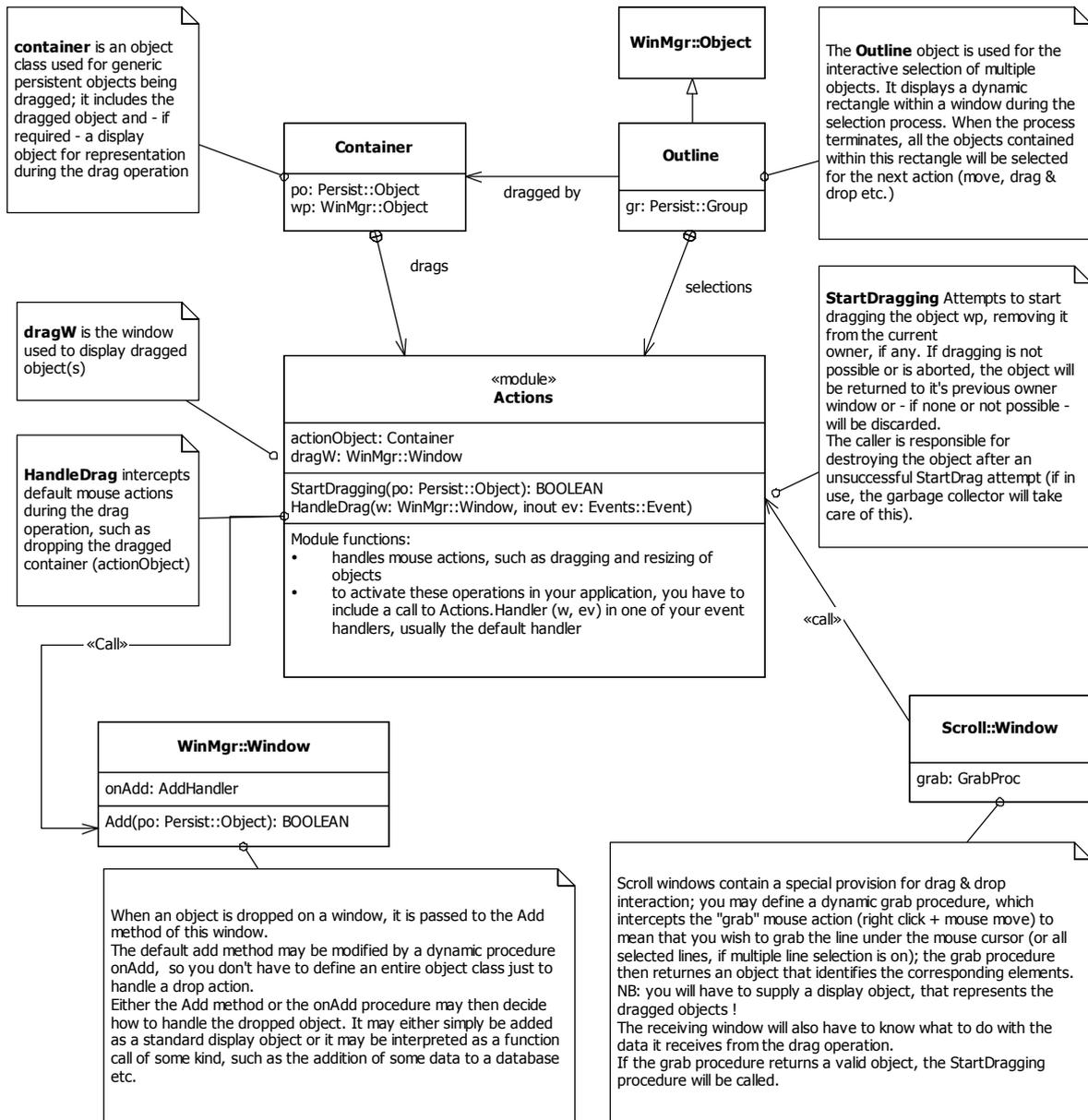


Figure 25 - Drag and Drop object structure

## 2.14 Databases

There are several major database models, including Transactional, Relational and Object Oriented databases.

- Transactional databases allow straightforward access to data that is stored in specially formatted and indexed files (we won't consider flat format data structures without indexing or other fast search structures as databases). Amadeus Software has supported Btrieve – later renamed to Pervasive.SQL - ever since the MS-DOS days in 1986. That database has evolved enormously and now includes both, transactional and relational programming. Amadeus-3 supports it in both modes.
- Relational databases are the most common, including Oracle, Sybase, DB2, MS SQL, Access etc. These databases use abstract tables to store your data. There is no specific definition to how they are implemented, but they all support some variant of the SQL data definition and access language. We won't go into any subtle aspects of database and SQL programming here. Let's just mention that there is a fairly well supported programming interface to access most relational databases, which is ODBC. Through this interface, an application can send SQL commands to the actual database and retrieve data. Amadeus-3 supports ODBC data access.
- Object Oriented databases are still experimental, usually proprietary and tied into a specific environment. For now, Amadeus-3 does not support any OO Database, as there are none that are really of interest at this point.

### 2.14.1 Basic interface

The standard database interface as defined in module `Db` is based on a minimal, transactional database engine. The functions that absolutely need to be supported are the following:

- Indexed file management (create, open, close indexed files)
- Record insertion, updating and deletion
- Ordered access to records (get first, last, next, previous element)
- Index-based search functions, with support for segmented keys in some form

Any database engine supplying at least these functions should be usable with Amadeus-3. Of course it would be nice to have support for a few more functions, such as record locking, but this is optional. Where a function is non-essential, it can usually simply be ignored.

Additional functions may be supported, but they will be database-specific, i.e. you must make use of information that is exported by the module implementing the interface to the desired database API.

Amadeus-3 maintains definitions for the various parts of a database, i.e. information about files (tables), keys, key segments and attached data dictionary objects etc. Based on this information, operations are performed using the standard database API.

### 2.14.2 Major features of the Amadeus-3 database interface

There are many features that distinguish the Amadeus-3 database interface, although Amadeus-3 is not really specialising in database handling. You will be surprised how easy it can be to manage fairly complex databases with Amadeus-3, even though you don't work at a very high level of abstraction (SQL or similar tools). By properly using the object-oriented possibilities of Oberon-2 and the design of the Amadeus-3 framework, you will be able to do things that would require a lot more work on other systems and might cause quite a few inconsistencies.

- **Close coupling with the Amadeus-3 framework**

By coupling Amadeus-3 information closely with the database, the application becomes much more consistent. You get a lot of implicit information that you usually don't have. Let's take the most impressive Amadeus-3 feature, where database operations are concerned:

- **Link between a database and display / print functions is implicit**

Another interesting aspect of the tight coupling of Amadeus-3 and database information is that you don't have to perform any additional operation to move data from one place to the other. When you read a record from the database, it's immediately available for display through one and the same data dictionary object that's used to generate data entry fields and report.

This is made easy by the fact that in Amadeus-3, development takes a data-centric approach. When you design a data input mask, you start by defining the variables that you will place into this mask and not - like in many other tools - by designing the mask and then linking fields to your application. When you define your variables and record structures, you already specify 80% of the information needed to create database tables. Amadeus-3 makes it easy to build those tables based on the already-defined variables.

- **SQL / ODBC definitions for database files are generated and updated automatically**

Once you created the table definitions with Amadeus-3, you just need to add a Data Source on the target system and pass the name of that data source to your Amadeus-3 application. The SQL commands to generate compatible data dictionary entries for your tables will be generated and executed automatically. At that point, you can use the Amadeus-3 ODBC interface to access your database as well as the native interface you may be using (e.g. Pervasive transactional).

- **Automatic database update**

Let's assume, you create and distribute an application. At some point, you make a modification to a database file (you add or remove fields, you change their type or their size etc.). Now you send a copy to users who already have a previous copy of your application. What happens? Usually, you would have to write code to import the old data into the new file format. But this only works the first time, i.e. from the latest distributed version to the new version. Not so with Amadeus, which automatically saves a full description of every database file's contents (fields, keys, attributes) in a text file which resides in the same directory as the data file itself. When the database is opened, the current definition of data dictionary and file objects - as found in the application resource file - is compared to the stored format. Any differences are resolved in the best possible way, including type changes etc.<sup>10</sup>

---

<sup>10</sup> At this point, Amadeus-3 does not try to enforce or represent inter-file dependencies, such as referential integrity. Such support will be added in the future, either directly or by using features of other database managers.

- **Database table definitions are stored in readable text files**

To make maintenance and documentation even easier, you can actually read and print the files used to define database tables. These files are stored in the same directory as the database files and have the extension « **.DFI** » (for **D**ata **F**ile **I**nformation). They contain nothing else but the encoded description of the data record linked to the database table and the table elements, i.e. keys, segments and attributes, in the standard expandable object script syntax. That means that each specific table format may add it's own relevant specifications.

Please note that you get all of this in a library that supports 3rd generation programming with all the corresponding flexibility and power. This is not a 4th generation tool that shines with sluggish performance and limited possibilities!

### 2.14.3 Basic Database Objects

The standard database interface of Amadeus-3 defines the following object classes in module `Db` to implement the functions specified above:

Class	Description
<b>Manager</b>	Must be installed by actual implementation of database manager. Initialisation must be called in or after <code>Project.InitInstance</code> , as the manager may require the GUI interface to be initialised.
<b>Database</b>	Descends from <code>Persist.Group</code> . Simplifies manipulation of a group of files. Allows specification of default path for all member files.
<b>File</b>	Defines the actual operations on database files, i.e. key access, insert, update and delete. Has link to an object of type <code>Values.Struct</code> , which is the record used in conjunction with above operations. Owns a list of keys.
<b>Key</b>	Define logical search access paths. Maximum number of keys depends on database manager, e.g. Btrieve V.6 allows up to 127 key segments or at most 127 independent keys. Owns list of segments. A key may have one or more segments.
<b>Segment</b>	Defines the actual record field - a <code>Values.DictEntry</code> object - for a particular key segment. This supplies the key's type and length. Also defines specific attributes, such as normal or descending sorting order etc.

#### **2.14.4 About the “Manager” Object Class**

The manager object defines all specific limits that apply for a specific instance of a database system. You may install several different managers to be used simultaneously. You may also share data from different managers in the same application and perform transactions involving more than one manager (where this feature is supported by the corresponding managers).

The database manager is a high-level abstraction. It does not make many assumptions about the database engines it supports. How well the integration will work depends very much on the actual database engine you try to integrate.

Pervasive.SQL/Btrieve is one specific manager, which is supported by Amadeus-3 right from the start and is now explained in more detail:

#### **2.14.5 Pervasive.SQL Requirements**

Pervasive.SQL (Btrieve) is the first manager / engine supported by Amadeus-3. It has all the required features, it is fast, supports the full client-server architecture on several network platforms and operates currently under DOS, OS/2, Windows 3.x, Windows 95, Windows NT/2000 and Linux.

You should use the standard installation utility when setting up a database. Amadeus-3 only provides those files required for development.

For development, you will need the following files:

- Btr.ob2
- Btrieve.def (in A3Lib)
- Wbtrv32.lib

NB: When you use Pervasive.SQL, you may access your data files simultaneously through the transactional and the relational interface. Amadeus-3 will see to it that compatibility is maintained, down to the actual file / table and index format.

#### **2.14.6 ODBC**

ODBC programming requires the following files:

- SQLDb.ob2
- SQL.def (integrates several standard .h files)
- ODBCInst.def
- ODBC32.lib
- ODBCCP32.lib

### 2.14.7 Transactional File Location

Where will transactional files be opened? This is determined by the following elements:

- The field fname of the database file object may contain a full file or path name, including embedded variables, which is the files's basic name
- If fname is empty, then the file object name field will be used as basic name
- If the file is member of a database group, then the database may have a global path, which may in turn contain embedded variables and which will be combined with the basic name to form the final path
- The basic name may contain relative path information, such as “.\” or “..\”, to indicate it's location relative to the global database path.

#### Example:

Let's assume you rapplication defines an Amadeus variable named “dbPath”. Before opening your database, you set the actual path through this variable, for instance through the user.

The path of your database may then be defined as “@[dbPath]”, which will translate into the actual path.

Some files may be placed in a different directory, such as “..\Shared”. They will be created in a directory named “Shared”, that is at the same level as the directory specified in dbPath.

### 2.14.8 Default database

You may have a default database, which contains some or all of the files for a new database. These may contain standard data for the application, such as a table with country names, zip codes, unit conversion etc.

When the application creates a new database, it will automatically start by copying the default database to the location specified for the new database, with subdirectories and all non-database files in those directories as well.

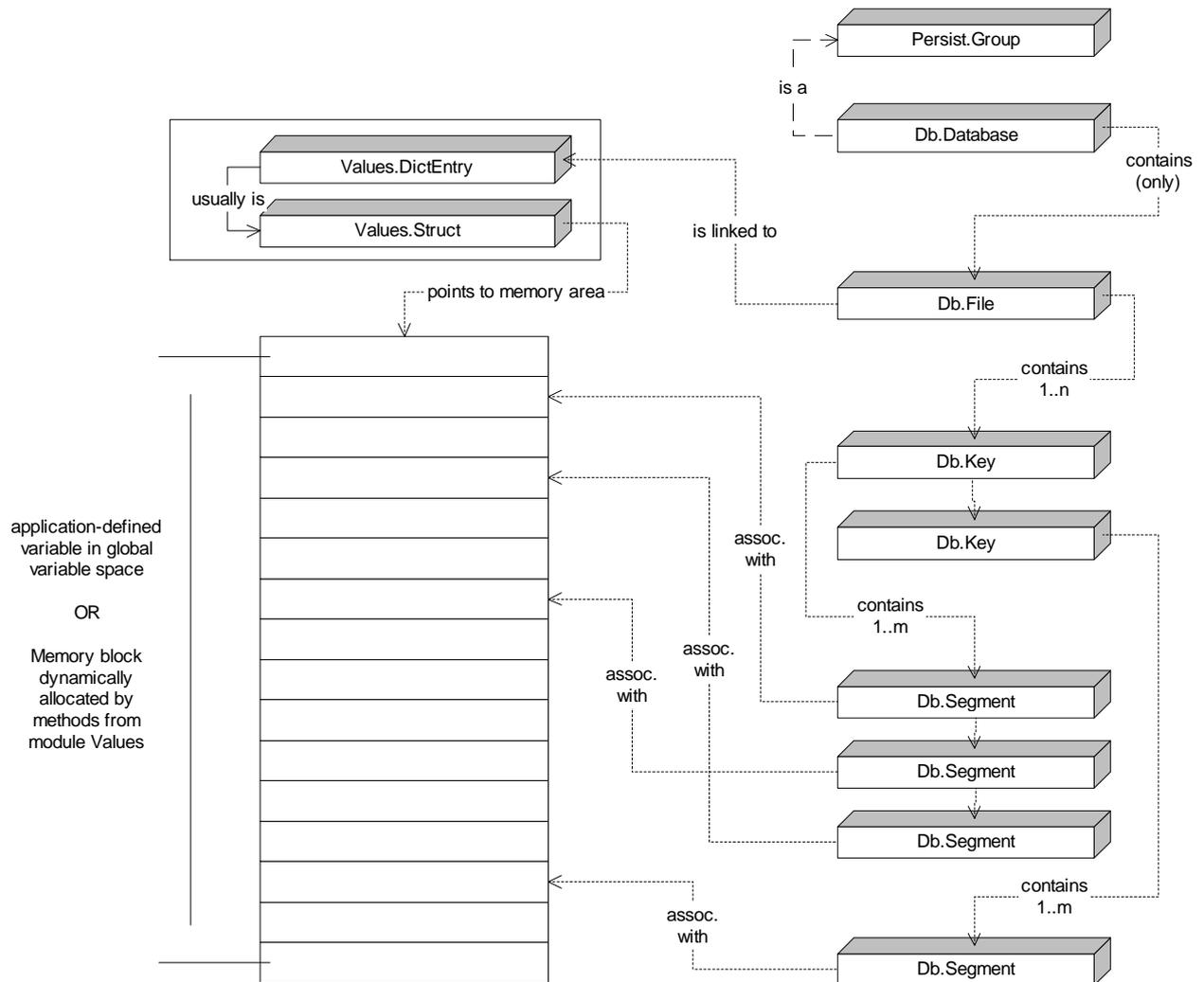
**Beware:** Make sure you do specify a default directory when using this function and that it is not a recursive definition.

### 2.14.9 Relational table location

ODBC tables are accessed via DSN entries or through proprietary mechanisms. They may be created by Amadeus-3 based on the Db.Database locaton information. If so, then the same rules apply as for transactional database files.

### 2.14.10 Structure of database information

In the following diagram, you will find a representation of how the contents of memory might look like, when a certain database structure is defined:



**Figure 26 – Database internal structure**

A database contains one or more files. Each file is linked to a dictionary entry, which is usually instantiated as a record (**Values.Struct**). The database file contains zero to n keys, where the maximum value of n depends on the database manager; in the case of Btrieve, it's 119 *segments*; given that each key may contain 1 or more segments, the maximum number of keys would therefore be 119, less if some keys define more than one segment. NB: the Amadeus-3 definition contains no such limitations; any file may contain an unlimited number of keys and each key an unlimited number of segments. Each interface to a database engine must define it's own limitations.

The segments, finally, establish the actual fields of the data record, which is linked to the database file, and that compose the complete key.

## 2.15 Transactional database programming

Here is an introduction to transactional database programming with Amadeus-3. the principles are universal. You won't need to read this if you are familiar with the concepts. Otherwise, even if you intend to use relational access methods only, you should study this chapter, since it explains some ideas you should help you write better programs even when relying on a large database engine.

### 2.15.1 Composite Indexes (segmented keys) explained

Sometimes, it is necessary to define indexes (keys) based on more than one field. All modern database tools support such a feature in one way or another. If you don't know what this is all about, don't worry, it's really quite simple:

Comparing and sorting numbers is elementary: they may simply be sorted by value, as the relation "greater than", "smaller than" and "equal to" are well defined for numbers, be it integer or real numbers.

Characters also have a sorting order, the most basic and best known one being the Alphabet. We all learned the sorting of alphabetic characters:  $A < B < C \dots < Z$ . There may be a few questions about how to sort uppercase and lowercase characters, not to mention accented or other special character. To deal with these issues, various standard character sets have been defined. They simply assign a numeric value to each character and then compare the numeric values of the characters to be compared. The best known character set is ASCII. Occasionally, the order specified by whatever character set you use may not be appropriate and may require a different mapping, but the principal remains the same; you simply create an intermediate character table, that re-defines the relationship of characters within a given set, which is another feature most database engines supply. How to treat upper- and lower-case characters is also up to the individual application; should lowercase "a" follow uppercase "A" or should it follow uppercase "Z"? Again, mechanisms within the database usually allow you to define for each index, how to deal with this issue.

When you sort strings, you compare character values one by one, starting at the first character in each string. We know a string "A" is "greater" than a string "B", when we reach a character from string "A" that is greater than the character at the matching location in string "B".

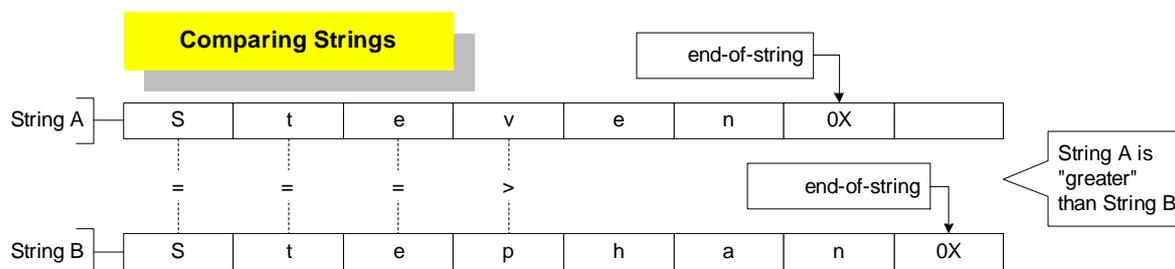


Figure 27 – Sorting strings

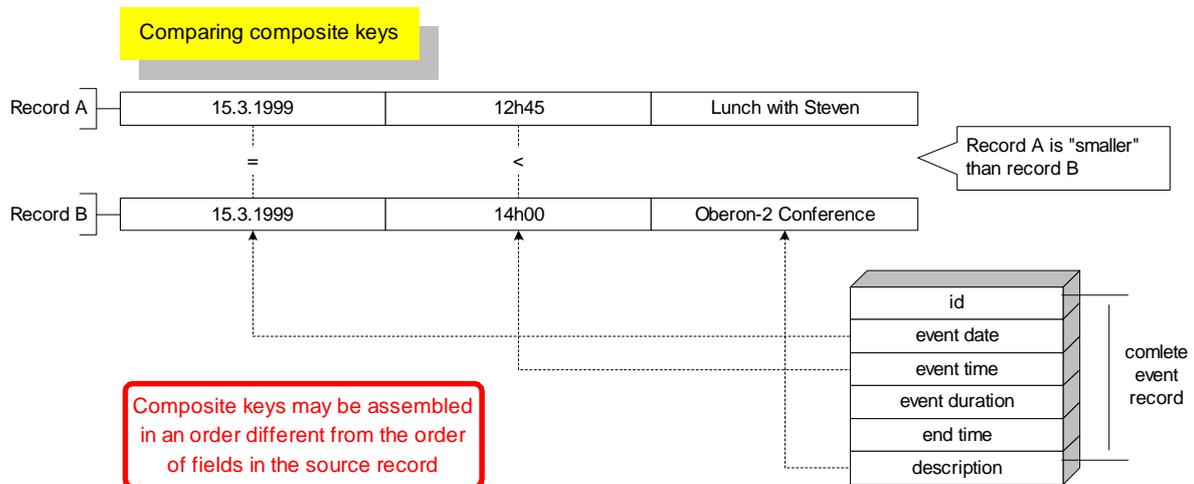
In the object script definition of key segments, use the following keyword to define the properties of a string key:

- NOCASE: consider uppercase and lowercase characters as the same

Some of the attributes specified for composite keys may also be applied for string keys, except for AUTOINC, which is strictly limited to numeric key values.

Sorting composite indexes – segmented keys – the principal is exactly the same: you sort each field that is a member of the composite index separately, as if it were individual characters in a string; Thus, when you have a composite key consisting of a date, a number and a string, you first compare the date value in record "a", establish if it is greater equal or smaller than the corresponding date in record "b", then you compare the numbers and finally the strings. The result (greater, equal, smaller)

is established whenever you reach the first element that is different in the two records or the last field; the result of the latest comparison is the result for the entire record.



**Figure 28 – Sorting composite indexes (segmented keys)**

For all types of key segments, you may further refine the conditions for their use with the following keywords:

- **DUP:** a given key may appear more than once; for composite keys that allows duplicates, this keyword must be repeated for *each segment*; the key is only considered “duplicate”, if *all* of it’s components are equal; A3Edit does this automatically for you; don’t forget it, if you write or modify the object script manually.
- **MOD:** a given key may be modified; for composite keys that allow modification, this keyword must be repeated for *each segment* in a key, even if only one of the segments may be modified.
- **NOSEGNUL:** if any single segment of the key is NULL (empty, according to type-specific definition), the record will not be referenced in *this* index (though it may be found by some other index).
- **NONULLKEY:** the complete assembled key must not consist of only NULL values (again, according to definition of NULL for each segment type).
- **AUTOINC:** the database engine will automatically set the value of this segment to the highest existing value of this segment plus one, when a new value is inserted; may only be used for segments linked to numeric variables; usually used for unique id fields.

## 2.15.2 Sample database construction

You could obviously write Oberon-2 code to create your database objects, but this would be very slow and require a lot of coding. A better way of doing it is to create the objects with the application editor A3Edit (cf. « 3 THE APPLICATION EDITOR A3EDIT »). Here is an example of the design of a section of a database for an appointment scheduler and its corresponding object definition.

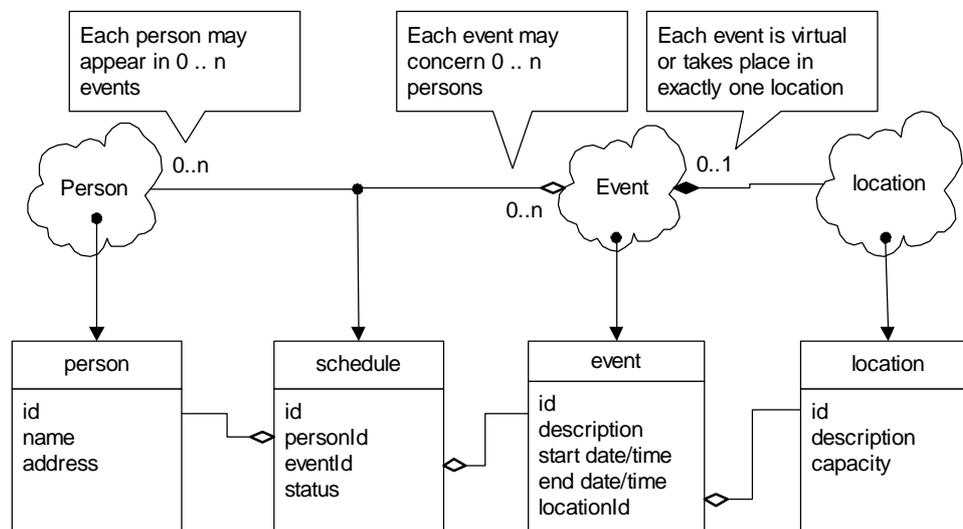
We want to define a database that will hold data about persons, who will schedule events, that will take place in certain locations. As data elements, we have:

- Persons
- Events
- Locations

In addition, we have a many-to-many relationship between persons and events. Each such relationship will contain additional information about its status, such as confirmed or unconfirmed. We will call the relationship:

- Schedule

Here is the corresponding diagram:



**Figure 29 - Sample database design**

This diagram identifies the data objects we need, i.e. the data records to hold all the various pieces of information, the corresponding database tables and the keys that establish the relationship between the various database tables. The `id` fields always define unique, internal identification codes, assigned by the database.

We haven't defined the other access paths, or indexes, which we will use to find specific entries in our database. First, let's define what kind of search operations we will want to perform on our database:

Operation	File	Key Number
<i>Find any record by it's internal id</i>	all	0
<i>Find a person by name</i>	person	1
<i>Find an event by it's description</i>	event	1
<i>Find an event by it's starting date &amp; time</i>	event	2
<i>Find a location by it's name</i>	location	1
<i>Find a location by it's capacity</i>	location	2
<i>Find all schedule entries for a person's id</i>	schedule	1
<i>Find all schedule entries for an event's id</i>	schedule	2

Note that some of these search operations concern more than one data field at a time. This will require the creation of "segmented keys" or composite indexes, which are described in the chapter "2.15 Transactional database programming".

The requirements formulated above require at least the following index definitions:

File / Record	Key 0	Key 1	Key 2	Key 3
person id name address	id	name		
schedule id personId eventId status	id	personId   eventId	eventId   personId	
event id description start date/time end date/time locationId	id	start date   start time	description	locationId
location id description capacity	id	description	capacity	

Figure 30 - Index structure table

### 2.15.3 Database sample object script

The script defining the records and tables required to implement this design would be the following:

```

OBJ Prj "DEMO"
REQUIRES NONE
OBJ Cmd Ok 1   Cancel 2   Abort 3   Retry 4
      Ignore 5   Yes 6    No 7    Add 8    Edit 9   Del 10   Ins 11   Exit 12
      Next 13   Prev 14   Up 15   Down 16   Print 17   Setup 18   About 19
      HelpIndex 20   HelpOnFocus 21   HelpOnHelp 22
ENDOBJ
OBJ DataDict "DbDemoData"
CONST 27 43
      AdrLen 41   CityLen 31   NameLen 31   FirstLen 16
      InfoLen 40   ZipLen 11
END
MEMBERS
  OBJ Struct "person"
    RECORD
      OBJ Num "id" LONG LEN 6 ENDOBJ
      OBJ Str "name" LEN NameLen ASCII UPPER ENDOBJ
      OBJ Str "first" LEN FirstLen CAPS ENDOBJ
      OBJ Str "company" LEN InfoLen CAPS ENDOBJ
    END
  ENDOBJ
  OBJ Struct "event"
    RECORD
      OBJ Num "id" LONG LEN 5 ENDOBJ
      OBJ Str "name" LEN InfoLen ENDOBJ
      OBJ Date "date" { DMY CENTURY } ENDOBJ
      OBJ Time "time" { MINUTES } ENDOBJ
      OBJ Time "until" { MINUTES } ENDOBJ
      OBJ Num "locationId" LONG LEN 5 ENDOBJ
    END
  ENDOBJ
  OBJ Struct "location"
    RECORD
      OBJ Num "id" LONG LEN 5 ENDOBJ
      OBJ Str "name" LEN InfoLen ENDOBJ
      OBJ Str "area" LEN InfoLen ENDOBJ
      OBJ Num "capacity" INT LEN 5 ENDOBJ
    END
  ENDOBJ
  OBJ Struct "sched"
    RECORD
      OBJ Num "id" LONG LEN 5 ENDOBJ
      OBJ Num "personId" LONG LEN 5 ENDOBJ
      OBJ Num "eventId" LONG LEN 5 ENDOBJ
      OBJ Num "status" SHORT LEN 5 ENDOBJ
    END
  ENDOBJ
ENDGRP
ENDOBJ
OBJ Group "DbDemoMain"
MEMBERS
  OBJ Db "main" PATH "data\"
    MEMBERS
      OBJ BtrFile "person" DATA "person"
        KEY 0 "id"
        SEG "id" AUTOINC END
        KEY 1 "name"
        SEG "name" DUP MOD NOCASE END
        SEG "first" DUP MOD NOCASE END
      ENDOBJ
      OBJ BtrFile "event" DATA "event"
        KEY 0 "id"
        SEG "id" AUTOINC END
        KEY 1 "name"
        SEG "name" DUP MOD NOCASE END
        KEY 2 "date"
        SEG "date" DUP MOD NOSEGNUL END
    END
  ENDOBJ

```

```

    SEG "time" DUP MOD NOSEGNUL END
    SEG "locationId" DUP MOD END
    KEY 3 "location"
    SEG "locationId" DUP MOD NOSEGNUL END
  ENDOBJ
  OBJ BtrFile "sched" DATA "sched"
    KEY 0 "id"
    SEG "id" AUTOINC END
    KEY 1 "person"
    SEG "personId" MOD NOSEGNUL END
    SEG "eventId" MOD NOSEGNUL END
    KEY 2 "event"
    SEG "eventId" MOD NOSEGNUL END
    SEG "personId" MOD NOSEGNUL END
  ENDOBJ
  OBJ BtrFile "location" DATA "location"
    KEY 0 "id"
    SEG "id" AUTOINC END
    KEY 1 "name"
    SEG "name" MOD NOSEGNUL NOCASE END
    KEY 2 "capacity"
    SEG "capacity" DUP MOD END
  ENDOBJ
  ENDGRP
  ENDOBJ
  ENDGRP
  ENDOBJ
  ENDOBJ

```

You should have no trouble identifying the various elements. And as you see, the definition is quite straight forward.

From these definitions you can generate the following Oberon-2 declarations, for easy reference from within your program (cf. « 2.23 Code generation »):

```

MODULE DemoApp;
IMPORT NumVal, StrVal, DateVal, DateOps, TimeVal, TimeOps, Persist, Projects, Dialogs,
    Values, SYSTEM, Db, Btr;

CONST
  PersonIdKey* = 0;
  PersonNameKey* = 1;

  EventIdKey* = 0;
  EventNameKey* = 1;
  EventDateKey* = 2;
  EventLocationKey* = 3;

  SchedIdKey* = 0;
  SchedPersonKey* = 1;
  SchedEventKey* = 2;

  LocationIdKey* = 0;
  LocationNameKey* = 1;

  AdrLen* = 41;
  CityLen* = 31;
  NameLen* = 31;
  FirstLen* = 16;
  InfoLen* = 40;
  ZipLen* = 11;

  CmdBase* = 0;

TYPE
  PtPersonDesc* = POINTER TO PersonDesc;
  PersonDesc* = RECORD

```

```

        id*: LONGINT;
        name*: ARRAY NameLen OF CHAR;
        first*: ARRAY FirstLen OF CHAR;
        company*: ARRAY InfoLen OF CHAR;
    END;
    PtEventDesc* = POINTER TO EventDesc;
    EventDesc* = RECORD
        id*: LONGINT;
        name*: ARRAY InfoLen OF CHAR;
        date*: DateOps.Date;
        time*: TimeOps.Time;
        until*: TimeOps.Time;
        locationId*: LONGINT;
    END;
    PtLocationDesc* = POINTER TO LocationDesc;
    LocationDesc* = RECORD
        id*: LONGINT;
        name*: ARRAY InfoLen OF CHAR;
        area*: ARRAY InfoLen OF CHAR;
        capacity*: INTEGER;
    END;
    PtSchedDesc* = POINTER TO SchedDesc;
    SchedDesc* = RECORD
        id*: LONGINT;
        personId*: LONGINT;
        eventId*: LONGINT;
    END;

VAR
    thisProject*: Projects.Project;
    mainDb*: Db.Database;
    personFl*: Btr.File;
    eventFl*: Btr.File;
    schedFl*: Btr.File;
    locationFl*: Btr.File;
    person*: PtPersonDesc;
    event*: PtEventDesc;
    location*: PtLocationDesc;
    sched*: PtSchedDesc;

    __ok:BOOLEAN; __po:Persist.Object;

PROCEDURE Find(name:ARRAY OF CHAR;id:INTEGER);
VAR ow:Persist.Object;
BEGIN
    IF ~thisProject.Find(name,id,ow,__po) & __ok THEN
        Dialogs.Message ("NotFound", name); __ok := FALSE;
    END;
END Find;
PROCEDURE Asgn(name:ARRAY OF CHAR;VAR data:ARRAY OF SYSTEM.BYTE);
BEGIN Values.AssignByName(thisProject.dict,name,SYSTEM.ADR(data),__ok) END Asgn;
PROCEDURE Init*(prjAsParam:Projects.Project;makeCurrent:BOOLEAN):BOOLEAN;
BEGIN
    __ok:=TRUE; thisProject:=prjAsParam;
    IF makeCurrent THEN Projects.current:=prjAsParam END;
    Find("main",Db.databaseId);mainDb:=__po(Db.Database);
    Find("person",Btr.fileId);personFl:=__po(Btr.File);
    Find("event",Btr.fileId);eventFl:=__po(Btr.File);
    Find("sched",Btr.fileId);schedFl:=__po(Btr.File);
    Find("location",Btr.fileId);locationFl:=__po(Btr.File);
    Asgn("person",person^);
    Asgn("event",event^);
    Asgn("location",location^);
    Asgn("sched",sched^);
    RETURN __ok;
END Init;

```

```
BEGIN
  NEW(person);
  NEW(event);
  NEW(location);
  NEW(sched);
END DemoApp.
```

### 2.15.4 Code for database access

When you wish to use these objects from your program, the code might look like this:

*Sample for reading all « person » records and writing the persons' name and first name to a text file, sorted by name and first name:*

#### Sequential access to database information

```
...
IMPORT Db,Btr,Files,App:=DemoSampleApp;
...
PROCEDURE WriteNames;
VAR f: Files.File;
BEGIN
  IF Files.OpenNew (f, 'Names.TXT', Files.WriteMode, TRUE, TRUE) THEN
    App.personFl.GetWithKey (Db.Lowest, App.PersonNameKey);
    WHILE App.personFl.result = Db.OK DO
      f.WriteString (App.person.name); f.WriteString (' ', ' ');
      f.WriteString (App.person.first); f.WriteEOL;
      App.personFl.Get (Db.Next);
    END; (* while personFl *)
    f.Close (TRUE);
  END; (* if open f *)
END WriteNames;
...
PROCEDURE (p: Project) InitInstance* (): BOOLEAN;
BEGIN
  ...
  Btr.mgr.Install;
  IF Btr.mgr.ready & App.mainDb.Open (TRUE) THEN
    WriteNames;
    RETURN TRUE;
  END; (* if mainDb.Open *)
  RETURN FALSE;
END InitInstance;
```

If you wish to read one particular record based on a known key, you do the following:

```
App.person.id := App.sched.personId;
App.personFl.GetWithKey (Db.Equal, App.PersonIdKey);
IF App.personFl.result = Db.OK THEN
  (* operation was successful *)
END;
```

In the case of a composite key, you may perform “joint” searches, even on partial keys. In the following sample, we'll try to find all the schedule records associated with a given event:

```
App.schedFl.Reset; (* clear information in file and record *)
App.sched.eventId := App.event.id;
App.schedFl.GetWithKey (Db.GtEqual, App.SchedEventKey);
WHILE (App.schedFl.result = Db.OK) &
  (App.sched.eventId = App.event.id)
DO
  (* perform desired operation on this element *)
  App.schedFl.Get (Db.Next);
END; (* while schedFl *)
```

Note the double condition for the loop: we keep reading, until we've either reached the end of the file (error condition) or until the record that was returned no longer matches the condition, i.e. the schedule record is no longer linked to the current event id.

To add a new record, you just assign all the proper field values and call the Insert method. All keys will automatically be built and inserted in all key paths:

```
App.sched.event.id := App.event.id;
App.sched.person.id := App.person.id;
App.sched.id := 0; (* this key is auto-incremented *)
App.schedFl.Insert;
```

To perform an update, you must first read the target record, modify it, then call the update method:

```
App.person.id := App.sched.personId;
App.personFl.GetWithKey (Db.Equal, App.PersonIdKey);
IF App.personFl.result = Db.OK THEN
  App.person.first [0] := 0X; (* clear first name *)
  App.personFl.Update;
END; (* if found *)
```

If you do not wish to update but to delete a record that you have found by the above manoeuvre, then you simply replace the call to method Update with a call to method Delete:

```
App.person.id := App.sched.personId;
App.personFl.GetWithKey (Db.Equal, App.PersonIdKey);
IF App.personFl.result = Db.OK THEN App.personFl.Delete END;
```

In any case, errors will be report in the field `fl.result`. If you want to be sure that an operation was properly performed or that a search was successful, check this field for `Db.OK`. If it's any other value, there was an error and you may use the value of `fl.result` to determine the exact cause of the error. For example:

- Key not found in a file access operation
- Trying to insert a record with duplicate key value where this is not allowed
- Trying to modify a field that is used in a non-modifiable key
- Record already locked by another user, when working on a network

## 2.16 ODBC Data Access

Here are the specifics on database access through ODBC with Amadeus-3. There are two possibilities: you want to access an existing database with Amadeus-3 or you want to create ODBC-compliant definitions for your database.

For the latter, you first have to create a DSN entry and the required database system files. For this, please follow your rdatabase administration information.

Once your database is ready for use, you may create an Amadeus-3 connection through objects found in SQLDb.ob2. First, you will need a connection:

```
VAR cn: SQLDb.Connection;
...
NEW (cn); cn.Init; Str.Assign ("connection string for database", cn.src);
IF cn.Connect ("user name", "password") THEN
  IF cn.CreateDb (associatedDb, TRUE, TRUE) THEN
    (* this creates and executes SQL statements to create tables *)
    ...
    END; (* if create *)
    cn.Close;
  END; (* if connect *)
```

Once the connection is established, you may use it to access any defined SQL table, view etc.

The Connection.CreateDb command creates all tables that are missing or changed with SQL statements. Here is a typical sample script, as generated automatically:

```
SET TRUENULLCREATE=OFF;
  ▪ For now, Amadeus-3 does not support true null keys, since they require a
  data record change
DROP TABLE cityTbl IN DICTIONARY;
CREATE TABLE cityTbl IN DICTIONARY USING '..\shared\city.dat'
  ▪ you can choose to create a table only in the dictionary (file exists)
  ▪ or you can have ODBC actually create it physically
(cityId IDENTITY CONSTRAINT cityIdKey UNIQUE PRIMARY KEY,
  ▪ This is the default behavior for auto-incremented keys
  ▪ They are considered primary keys and unique non-duplicate
cityDateMake DATE NOT NULL,
cityTimeMake TIME NOT NULL,
cityDateStamp DATE NOT NULL,
cityTimeStamp TIME NOT NULL,
cityName VARCHAR(49) NOT NULL CASE,
cityZip VARCHAR(15) CASE,
  ▪ CASE is added if at least one key segment for this field has the CASE
  attribute
cityCntry VARCHAR(20) NOT NULL);
CREATE INDEX cityNameKey IN DICTIONARY ON cityTbl ( cityName, cityZip );
CREATE INDEX cityZipKey IN DICTIONARY ON cityTbl ( cityZip );
```

Reading data through statements:

```
VAR st: SQLDb.Statement;
...
cn.NewStatement ("SELECT * FROM client", st);
IF st.Execute (FALSE) THEN
  ...
  END; (* if execute *)
```

Note that Amadeus-3 adds a lot of functions: the SQL Statement may contain references to variables as input parameters and output columns.

### 2.16.1 Sample code for ODBC access

Here is a sample procedure that produces a statistics report based on a number of user parameters. The database contains shipment data about articles that belong to various product groups and brands; the shipments specify the destination country, the end user and the shipping agent. All the shipment records contain a date field. The output is to be formatted as a table with columns that specify the quantities shipped per month and – in the first column – the total per line.

The rows may vary: the user can select the fields he wants to include and in which order he wants them sorted. He may for example specify that he wants the shipments per brand and country. This will produce a table with lines per country and summary totals per brand and year, plus an overall total.

The code here concentrates on the central procedures only. The user interface definition and code was not included and the generation of the output was omitted as well, though it is a very short procedure, which generates simply a few strings that can be used by the report syntax (cf. below).

sortList contains “meta” records, which specify the name of each possible field and is sorted in the order the user wanted the output. Other variables should be fairly obvious.

Based on the user specifications, the procedure generates an SQL statement that will return rows with totals for the lowest level of the table already sorted, per month and the other sorting criteria.

```

PROCEDURE Report* ();
VAR
  now: DateOps.Date;
  fs : Paths.Specifier;
  nm : Paths.FullStr;
  f  : Streams.Stream;
  ac : Accumulator;
  b  : BOOLEAN;

PROCEDURE GenStat (): BOOLEAN;
VAR s : Str.PStr;

PROCEDURE AddFields (param: BOOLEAN);
BEGIN
  Rapp.sortList.Reset;
  WHILE Rapp.sortList.Step (Rapp.meta^) DO
    IF (Rapp.meta.name = "brand" ) & Rapp.statParam.inclBrand THEN
      Str.AppendC (s^, ",brandTbl.name" );
      IF param THEN Str.AppendC (s^, "@[stat.brand] AS Brand" ) END END;
    IF (Rapp.meta.name = "product") & Rapp.statParam.inclProduct THEN
      Str.AppendC (s^, ",productTbl.name");
      IF param THEN Str.AppendC (s^, "@[stat.product] AS Product" ) END END;
    IF (Rapp.meta.name = "article") & Rapp.statParam.inclArticle THEN
      Str.AppendC (s^, ",articleTbl.code");
      IF param THEN Str.AppendC (s^, "@[stat.article] AS Article" ) END END;
    IF (Rapp.meta.name = "cntry" ) & Rapp.statParam.inclCntry THEN
      Str.AppendC (s^, ",cntryTbl.name" );
      IF param THEN Str.AppendC (s^, "@[stat.cntry] AS Country" ) END END;
    IF (Rapp.meta.name = "endUser") & Rapp.statParam.inclEndUser THEN
      Str.AppendC (s^, ",endUserTbl.name");
      IF param THEN Str.AppendC (s^, '@[stat.endUser] AS "End user"' ) END END;
    IF (Rapp.meta.name = "agent" ) & Rapp.statParam.inclAgent THEN
      Str.AppendC (s^, ",agentTbl.name" );
      IF param THEN Str.AppendC (s^, "@[stat.agent] AS Agent" ) END END;
  END; (* while *)
END AddFields;

BEGIN (* GenStat *)
  NEW (s, 1000);
  COPY ("SELECT YEAR(shipTbl.date0)@[stat.year] AS
Year,MONTH(shipTbl.date0)@[stat.month] AS Month", s^);
  AddFields (TRUE);
  Str.AppendC (s^, ",SUM(boxTbl.qty)@[stat.qty] AS Shipped"); Str.AppendEol (s^, TRUE);
  Str.AppendC (s^, " FROM shipTbl,boxTbl,articleTbl");
  IF Rapp.statParam.inclBrand THEN Str.AppendC (s^, ",brandTbl" ) END;
  IF Rapp.statParam.inclProduct THEN Str.AppendC (s^, ",productTbl" ) END;
  IF Rapp.statParam.inclEndUser OR Rapp.statParam.inclCntry THEN

```

```

    Str.AppendC (s^, ",orderTbl,endUserTbl") END;
IF Rapp.statParam.inclCntry THEN Str.AppendC (s^, ",cntryTbl" ) END;
IF Rapp.statParam.inclAgent THEN Str.AppendC (s^, ",agentTbl" ) END;
Str.AppendEol (s^, TRUE);
Str.AppendC (s^, " WHERE (YEAR(shipTbl.date0)>");
    Convert.AppendNum (s^, 0X, Rapp.statParam.year [0], FALSE);
Str.AppendC (s^, " OR YEAR(shipTbl.date0)=" );
    Convert.AppendNum (s^, 0X, Rapp.statParam.year [0], FALSE);
Str.AppendC (s^, " AND MONTH(shipTbl.date0)>=" );
    Convert.AppendNum (s^, 0X, Rapp.statParam.month [0], FALSE);
Str.AppendC (s^, ") AND (YEAR(shipTbl.date0)<=");
    Convert.AppendNum (s^, 0X, Rapp.statParam.year [1], FALSE);
Str.AppendC (s^, " OR YEAR(shipTbl.date0)=" );
    Convert.AppendNum (s^, 0X, Rapp.statParam.year [1], FALSE);
Str.AppendC (s^, " AND MONTH(shipTbl.date0)<=" );
    Convert.AppendNum (s^, 0X, Rapp.statParam.month [1], FALSE);
Str.AppendC (s^, ") AND boxTbl.shipId=shipTbl.id AND articleTbl.id=boxTbl.articleId");
Str.AppendEol (s^, TRUE);
IF Rapp.statParam.inclBrand THEN
    Str.AppendC (s^, " AND brandTbl.id=articleTbl.brandId");
    IF Rapp.statParam.brand [0] # 0X THEN
        Str.AppendC (s^, " AND brandTbl.name='");
        Str.AppendC (s^, Rapp.statParam.brand);
        Str.AppendC (s^, "'");
    END;
END;
END;
IF Rapp.statParam.inclProduct THEN
    Str.AppendC (s^, " AND productTbl.id=articleTbl.productId");
    IF Rapp.statParam.product [0] # 0X THEN Str.AppendC (s^, " AND productTbl.name='");
        Str.AppendC (s^, Rapp.statParam.product); Str.AppendC (s^, "'") END; END;
    IF Rapp.statParam.inclEndUser OR Rapp.statParam.inclCntry THEN
        Str.AppendC (s^, " AND orderTbl.id=shipTbl.orderId AND
endUserTbl.id=orderTbl.endUserId");
        IF Rapp.statParam.endUser [0] # 0X THEN
            Str.AppendC (s^, " AND endUserTbl.name='");
            Str.AppendC (s^, Rapp.statParam.endUser); Str.AppendC (s^, "'") END END;
    END;
    IF Rapp.statParam.inclCntry THEN
        Str.AppendC (s^, " AND cntryTbl.id=endUserTbl.cntryId" );
        IF Rapp.statParam.cntry [0] # 0X THEN
            Str.AppendC (s^, " AND cntryTbl.name='");
            Str.AppendC (s^, Rapp.statParam.cntry); Str.AppendC (s^, "'") END END;
    END;
    IF Rapp.statParam.inclAgent THEN
        Str.AppendC (s^, " AND agentTbl.id=shipTbl.agentId");
        IF Rapp.statParam.agent [0] # 0X THEN Str.AppendC (s^, " AND agentTbl.name='");
            Str.AppendC (s^, Rapp.statParam.agent); Str.AppendC (s^, "'") END;
    END;
END;
Str.AppendEol (s^, TRUE);
Str.AppendC (s^, " GROUP BY YEAR(shipTbl.date0)"); AddFields (FALSE);
Str.AppendC (s^, ",MONTH(shipTbl.date0)");
Str.AppendEol (s^, TRUE);
Str.AppendC (s^, " ORDER BY YEAR(shipTbl.date0)"); AddFields (FALSE);
Str.AppendC (s^, ",MONTH(shipTbl.date0)");
SQLDb.NewStatement (GlobalDb.gdb.cn, s^, stm);
RETURN stm.Execute (FALSE);
END GenStat;

BEGIN (* Report *)
IF (GlobalDb.gdb.cn = NIL) OR (GlobalDb.gdb.cn.dbc = NIL) THEN
    Dialogs.Error ("", "NoConnection"); RETURN END;
FillSort;
IF Rapp.statParam.year [0] = 0 THEN
    DateOps.Current (now); DateOps.Trimester (now, FALSE);
    Rapp.statParam.month [0] := now.month; Rapp.statParam.year [0] := now.year;
    DateOps.Trimester (now, TRUE );
    Rapp.statParam.month [1] := now.month; Rapp.statParam.year [1] := now.year;
END; (* if *)
LOOP
    IF ~WinEvent.StdModal (Rapp.statParamWin) THEN RETURN END;
    IF Files.OpenBuffered (f, "Stat.inc", Files.ShareReadWrite, TRUE, TRUE) THEN EXIT END;
    Dialogs.Message ("", 'OutputFileErr');
END; (* loop *)
SaveSort;
Pointer.CaptureWait; b := GenStat ();
IF b THEN GenOutput (TRUE) END;
f.Close; Pointer.ReleaseWait;
IF b THEN Rpt.Run ('Stat', DateOps.NoRange) END;
END Report;

```

## 2.17 Database Editor DbViewer

Amadeus-3 comes with a database editor utility called DbViewer. This utility allows you to view a database file's definition and contents and even to edit the data.

For use, you must have installed a supported database driver (per default Pervasive.SQL). Simply launch the DbViewer.exe utility in the A3\DbViewer directory and the following window will show up:

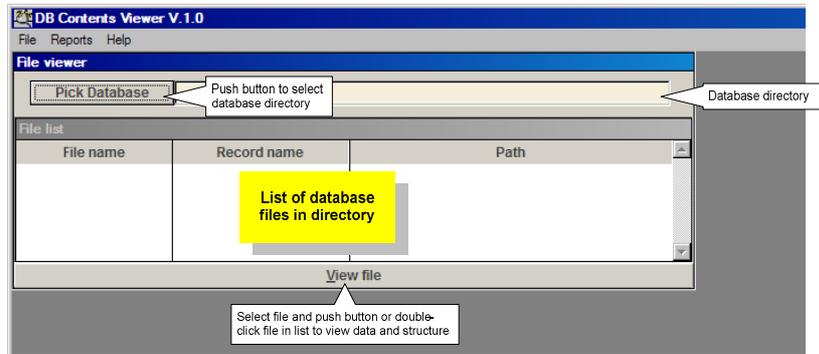


Figure 31 - Select database in DbViewer

Once the database directory is selected, you can open any one or more of the available tables. Each table you open will show up in the “Open file list” window below the database directory. You can then switch to the view of that table by double-clicking it in the open file list.

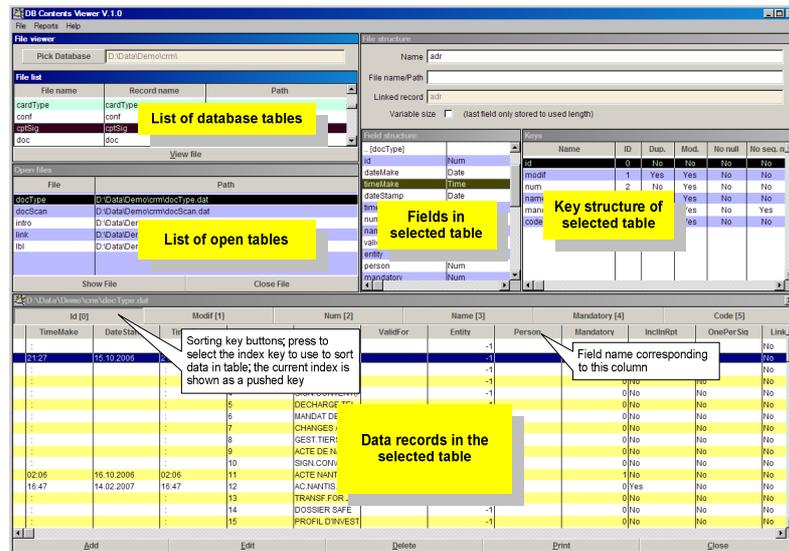
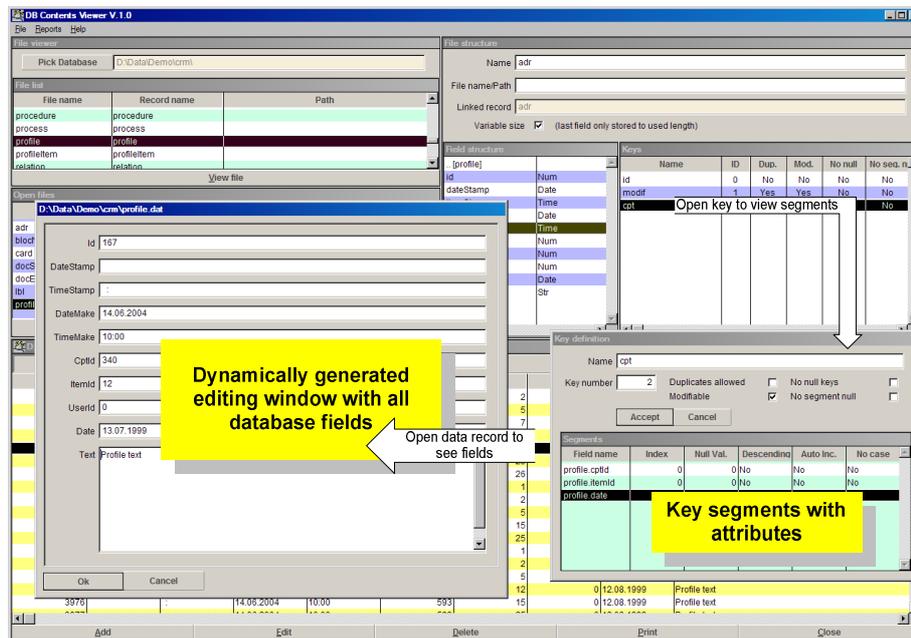


Figure 32 - Viewing selected file

### 2.17.1 Editing features:

- You can edit the database contents by double-clicking a line in the data viewer
- You can delete any entry by pressing “DEL” or the “Delete” button
- You can change the sorting order of the data by clicking on one of the index key buttons in the header of the data table.
- You can view the segments of a key by clicking the key name in the key list in the table definition.

Note that the editing window for the database contents is fully dynamic and adapts completely to the actual contents of the database. It supports all basic types, including multi-line string fields, options as dropdown lists and represented as strings, Booleans as check boxes etc.



**Figure 33 - Editing database contents and definition**

NB: You cannot change the database structure here. To do so, you should use A3Edit, redefine the table, recompile your application to ensure that it is compatible with the new definition and then start your application in admin mode (+admin launch parameter) to convert an existing database.

### 2.17.2 Copying records

In the button menu of the “Open File” window, you’ll find the “Copy to” option, which allows you to copy data from the currently active file (cf. open database in bottom window) to the file selected in the “Open File” window.

The copy is performed field by field, based on identical field names in the two records. Where the data format is different, the destination field will be appropriately transformed, if possible.

## 2.18 Reports

The reporting feature of Amadeus-3 is very powerful and flexible. Via a simple scripting language, you can generate sophisticated screen and printer output as well as various formatted text formats for data export. As it is parsed dynamically, you can easily edit and change it during program execution, to generate the precise output format that you desire without compilation or other complex procedures, a definite plus, as anyone who ever has had to generate reports for end-user applications will know.

The scripting language has several ways for interacting with the application. In the first place, it shares all the persistent objects that are accessible via projects. Then it can send strings to the application, that will be evaluated by a method attached to the report object or a dynamically defined procedure. This can be used as control mechanism, to initialise data, to access data elements in various structures or to launch any other type of procedure.

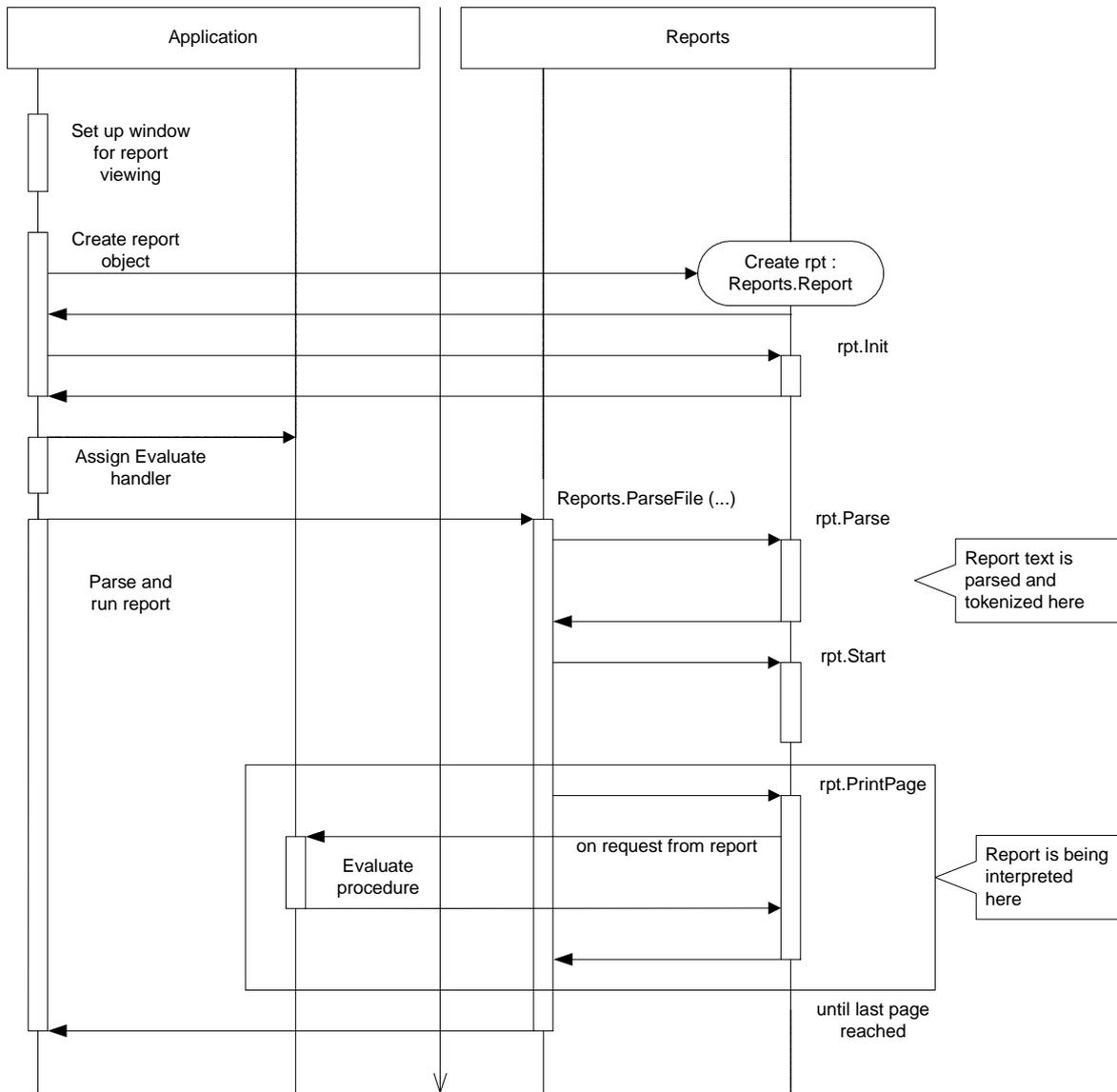


Figure 34 - Interaction of application and report object

Beyond that, it is quite autonomous and can perform many functions independently of the Oberon-2 code. You can create local variables, evaluate arithmetic expressions, perform string operations, use simple conditional structures, access database files and scroll lists etc. You can format text output in

various ways and insert visual objects, if they have been conceived to be flexible and adaptable to the output medium.

Text output can also be re-directed to files, such as text files in flat format or as comma-separated data files – usually with CSV extension – that can be imported into most spreadsheet and database applications.

Scripts can use include files, which can be named in any way you like, but it is advised to use the extension INC, for easy identification. The default extension for script files is FMT (for ForMaT). It is even possible to use dynamic include files, i.e. to generate new include files during the execution of a script and then to include the resulting file into the script as it is being executed, providing for entirely dynamic interpretation.

To understand all the subtleties of report evaluation, you should consult the major modules that deliver the power behind it:

- Reports.ob2 for the general report syntax and evaluation
- RptTbl.ob2 for report tables
- DbReport.ob2 for extended database reports
- StrFmt.ob2 for the translation of embedded strings
- Expressions.ob2 for the handling of mathematical, logical, string and date expressions

Rather than a lengthy explanation, we'll use some sample code to illustrate the process. First, a short demonstration report, that uses some of the available features: DEMO.FMT

```
-- Define fonts to be used in this report, cf. module Fonts
FONT 0 "Times New Roman;h=-11;p=v"
FONT 1 "Arial;h=-13;p=v;a=3"

-- Set margin on left hand side of report output
MARGIN 4

-- Define local variables; use standard class names for value classes
-- and the standard definitions for objects of those classes
VAR Num idx INT END
VAR Real result FIX LEN 15 DIGITS 2 TICKS END

-- Header for each page that will print the title "Demo" and on the
-- top right hand side of the page, the report date, time and page
-- number; NB: for this to work you must include the module RptTable
-- in at least one module that is linked to your application
HEADER
  OBJECT 1 "Table;Demo##50;@>@[RptDate] / @[RptTime] Page @[PageNb]"
  NEWLINE 2; -- Add 2 blank lines
END - Head

-- string in IF statement must be handled by Evaluate handler or method
IF getFirst THEN
  -- set idx variable to zero
  SET idx 0
  REPEAT
    -- see module Translate for details on variable translation
    PRINT "Element @[idx] found"
    -- assign the result of the expression to variable
    SET result=idx * 2.5;
    MOVETO 40 -- Move to column 40 for the next print
    -- Print the result of the above calculation
    PRINT "@[idx]*2.5 = @[result]"
    NEWLINE -- Move on to next line
    -- increment idx variable, then print it inside a string;
    INC idx
    -- the string in the UNTIL statement
    -- must be handled by the evaluation procedure or method
  UNTIL NOT getNext
END -- if getFirst
```

And here is the Oberon-2 code that invokes and controls the above report; all the non-essentials have been omitted:

```

MODULE Demo;

IMPORT Reports,WinEvent,Startup,App:=DemoApp, ...
...
VAR counter: INTEGER;

PROCEDURE Evaluate (rpt: Reports.Report; VAR s: ARRAY OF CHAR): BOOLEAN;
BEGIN
  IF s = 'getFirst' THEN          -- should match the string from report
    counter := 0;RETURN TRUE; -- Initialisation
  ELSIF s = 'getNext' THEN
    INC (counter);                -- Action
    RETURN counter < 10;         -- Termination condition
  END; (* if s *)
  RETURN FALSE;
END Evaluate;

PROCEDURE RunReport* (name: ARRAY OF CHAR);
VAR
  rpt: Reports.Report;
  ok : BOOLEAN;
BEGIN
  WinEvent.CaptureWait;          -- Parsing could be lengthy
  NEW (rpt); rpt.Init;           -- Creation of report object
  rpt.eval := Evaluate;         -- Assign evaluation procedure
                                -- Parse and start preview process
  ok := Reports.ParseFile (name, rpt, FALSE, TRUE);
  WinEvent.ReleaseWait;
END RunReport;

...
PROCEDURE DefaultHandler;
BEGIN
  IF (ev.tp = Events.Command) & (ev.code = App.DemoReportCmd) THEN
    RunReport ('DEMO');
  END;
END DefaultHandler;

PROCEDURE (p: Project) InitInstance* (): BOOLEAN;
BEGIN
  IF p.InitInstance^ () & App.Init (p, TRUE) THEN
    ... -- Assign windows for report preview (defined in App)
    Reports.Assign (App.rptTopWin, App.rptViewWin, NIL, 0, 0);
    ...
    RETURN TRUE
  END; (* if InitInstance *)
  RETURN FALSE;
END InitInstance;

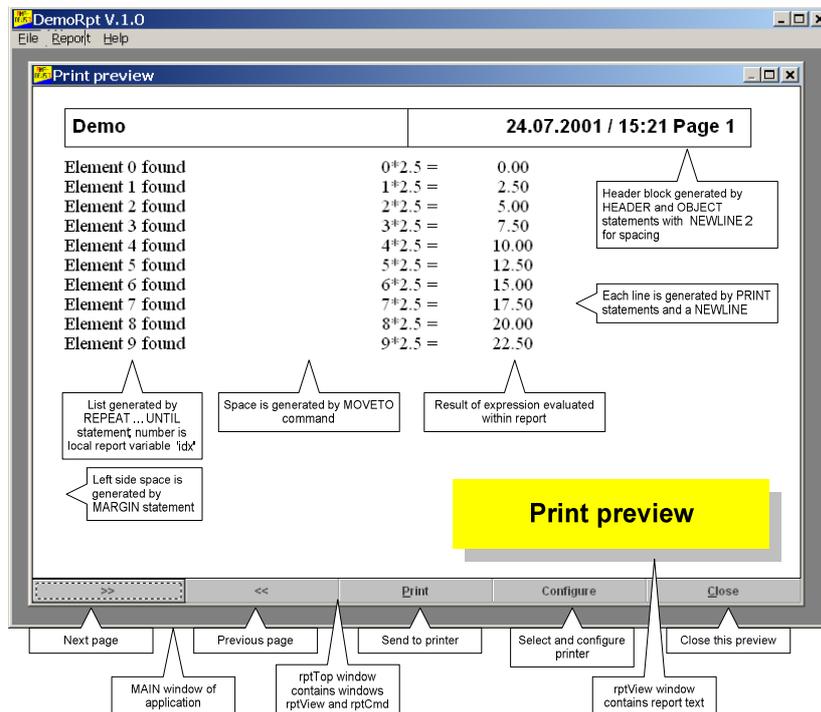
BEGIN ... (* Usual program initialisation code *)
END Demo;

```

You can find the full, working code in the sample program DemoRpt.

As you may notice, 2 windows are sent to Reports.Assign during program initialisation. These windows will be used for the report preview. The top window is structural and may contain other windows. The actual report viewing window is assumed to be an empty standard window. It will be used to display the report contents. Typically, you will want to create appropriate windows along with other standard interface elements and group them in a generic application, which remains less or more unchanged for most of your applications. Please see the chapter on “Basic application objects”.

Here is the output of the demo report in a typical viewing window:



**Figure 35 - Simple report viewer**

Keyboard and mouse commands for the viewer:

- PgUp, Left, Up, Back, mouse wheel back : Previous page
- PgDn, Right, Down and mouse wheel forward : Next page
- Home, End: first and last page already generated and still in buffer (per default 100 pages back)
- 

Note that the result on paper should be very similar, except that the output will be adapted to the format of the paper, instead of matching the dimensions of the preview window.

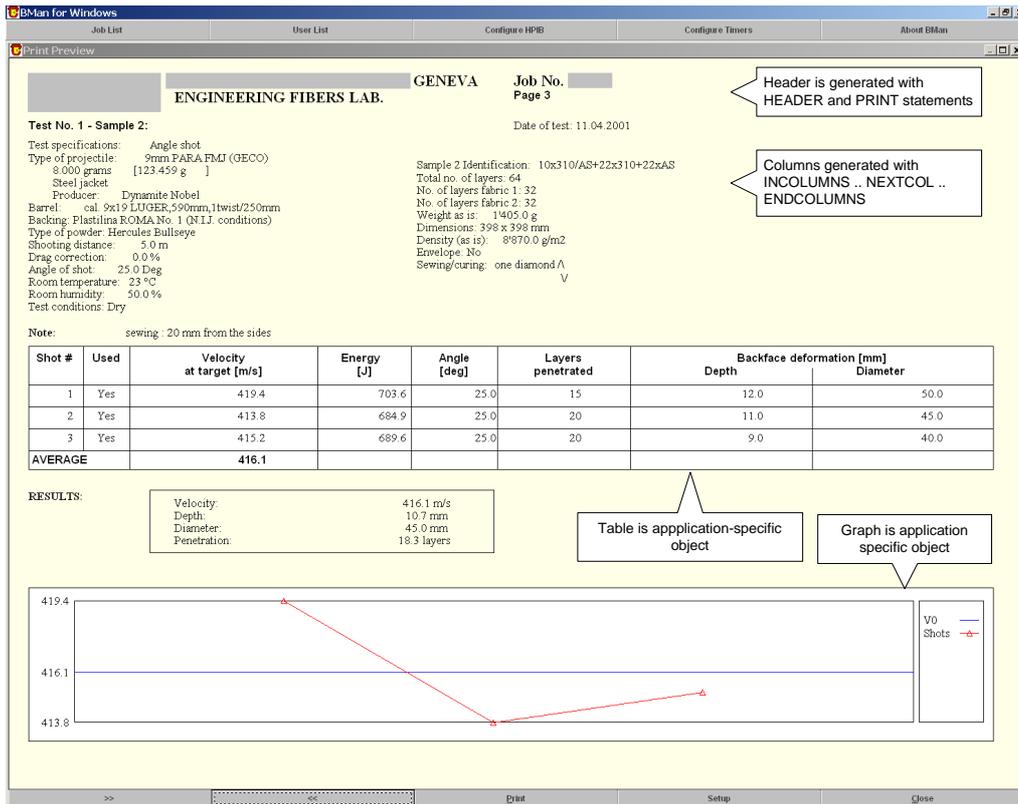
Regarding the Oberon-2 code for the report object or the Evaluation handler, there is not much more to be added, except to say that the complexity of what can be done during the handling of a report request has practically no limits. The interface is really simple. You just have to remember that any data that you wish to access from one individual call to the next (such as the “getFirst” and “getNext”) needs to be saved globally, i.e. local variables of the Evaluate procedure will be lost between calls.

If you use DbReport, this will be done automatically for all open databases and their associated records.

As for the report script, there are a lot of implicit, not so obvious features, through which it is possible to achieve quite interesting results. The best way to learn them is again by example, so the next section will demonstrate some report output and how it was generated.

### 2.18.1 Detailed study of report elements

The first complex sample is taken from a laboratory application ballistic protection equipment testing. The report represents tables and a small graph along with a set of parameters. It makes use of column printing, report tables and graphical objects. We'll use it to go through the various elements of a report script.



**Figure 36 - Sample report with graphical objects**

### 2.18.1.1 Fonts

To produce text output, you have to define the fonts that you wish to work with. Amadeus-3 reports allow you to set up a list of standard fonts for a report, which can then be referenced by index number:

```
FONT 0 "Times New Roman;h=-11;p=v"
FONT 1 "Arial;h=-13;p=v;a=3"
```

You may define fonts in a standard include file for your application, so as to use always the same fonts for every report. You may also re-define fonts that have been previously defined, so you may override a standard font definition in a specific report.

You can also define fonts dynamically, i.e. during execution time. The FONT statement defines a given font statically, during parsing. The SETFONT statement is integrated into the code and is executed like another statement in the execution sequence, altering the font table from that point on. You can also use dynamically generated or program-supplied strings for the SETFONT statement, as in SETFONT "@[fontString]".

### 2.18.1.2 Simple output control commands

You can send text to the report output channel by using the following simple commands:

PRINT, NEWLINE, MOVETO

A PRINT statement, as well as all other statements that require a font to produce output, will use the font 0 (zero) by default, unless you specify otherwise by including the font number right after the statement. The NEWLINE command parameter is a repeat factor: a positive number means insert n times a newline, a negative number means insert a line that is 1/n th of the normal line height.

Example:

```
PRINT 1 "TITLE" NEWLINE 3
```

The MOVETO command sets the column within the current line. The unit is the average character width of font 0. For more precise placement commands see the following topic.

### 2.18.1.3 Conditional statements

The execution flow of a report can be controlled with conditional statements, including:

IF, ELSIF THEN, ENDIF, WHILE, DO, REPEAT, UNTIL

The condition may be a complex statement, cf. module Expressions.ob2 and the following topic on variables.

### 2.18.1.4 Variables

You may define local variables inside a report script, which you can freely use for the entire lifetime of the report. The syntax is VAR followed by the standard definition of a corresponding persistent object.

```
VAR Str name LEN 80 END
SET name = "A long string for " + user.name;
PRINT "@[name]" NEWLINE
```

You can use expressions to set variables and of course you can use variables inside expressions. The type of the expression depends on where it is used: if it is a condition, then the result type should be BOOLEAN, otherwise it should be the same as the variable to which you assign the expression.

### 2.18.1.5 Standard variables

Some variables are internal and may be used in any report. They are updated by the report:

- RptDate = current data when report was printed
- RptTime = current time when report was printed
- RptName = report name, i.e. the file name used to load the report
- NbLines = total number of lines per page using FONT 0
- PageNb = current page number
- LineNb = current line number
- Margin = currently active margin margin setting
- Column = current column
- Footer = Footer margin for page
- PX = horizontal position in report in pixels
- PY = vertical position in report in pixels
- MX = maximum number of horizontal pixels
- MY = maximum number of vertical pixels
- MaxY = max vertical position reached
- MaxPage = highest page number for total report
- Answer = string returned from latest call to CMD

NB: maxPage is calculated only if it is not initialized to NoCount; it requires that the entire report be generated for the current display / print format

### 2.18.1.6 Constants

Normally the variables imported from the application are used unchanged, i.e. if your application handles customer records and you run a report that prints the currently displayed customer, this report will use the currently loaded record. This was fine as long as reports were modal and unique. The support of multiple reports which can be re-used for reformatting, export etc., it became necessary to make sure that data that might be changed can be fixed for a given report.

This is accomplished via constant assignments. A constant assignment assigns any expression to an internal variable; this expression will be evaluated only once, during the first execution. After that point, the resulting value will be stored and assigned on every subsequent run of the report.

Example of constant use:

```
VAR Num contactId LONG END CONST contactId = contact.id;

CMD DbFile:contact "GETID @[contactId]"
```

During the first execution, the value of contact.id is assigned to contactId and on every subsequent run, the same value will be used. So by reloading the corresponding contact record, the report will always print the same data.

### 2.18.1.7 Report TITLE command

Each report window should be easily identified, especially when the user opens multiple report windows. This can be accomplished by assigning a dynamic report title from within the report script:

```
TITLE "Report for @[contact.name] @[contact.first]"
```

This assigns the string "Report for" followed by the name and first name of the current contact.name and contact.first record, re-using the previous example.

### 2.18.1.8 Procedures

You may create procedures for repetitive or logical sub-functions, which may then be called from within the same report.

```
PROC printClient
  PRINT "@[client.name], @[client.adr], @[client.city]" NEWLINE
  PRINT "@[client.status]" NEWLINE
END
```

Special procedures include HEADER and FOOTER, which, when defined, are automatically called at the beginning and end of each page. They may be re-defined during the report execution. The active one is always the most recent one found in the execution flow.

```
HEADER
  MOVETO 80 PRINT "Page: @[PageNb]" NEWLINE
END - header
```

By using a standard include file, you can call specific standard procedures, such as a typical header or footer.

```
PROC setHeader
  HEADER
    PRINT "Demo application" MOVETO 80 PRINT "Page @[PageNb]" NEWLINE
  END
END - set header
```

### 2.18.1.9 Positioning

The positioning of output is controlled automatically by the report object. It tracks the extent of the objects added to the report and moves the X and Y counters accordingly. When a PRINT operation is performed, it uses the font used to calculate the new position. For NEWLINE, it always uses the font 0 line height to increment the Y position, while setting the X position to the current margin value (set with the MARGIN command). It also increments the next position after inserting display objects with an OBJECT command. The object extent on both, X and Y axis is used as increment value.

The report object will adjust the maximum width and height of each virtual page according to the dimension of the output device (e.g. a window or a printer page) and will insert page breaks where necessary.

If you specify a FOOTER, then the value set in the Footer variable will be used to calculate an offset from the bottom of the page at which the footer procedure will be called, followed by a page break. This effectively reduces the available space on each page accordingly.

```
SET Footer=5;
FOOTER
  OBJECT 6 "Table:@LINE=//;@NF:##5;@NF:Disclaimer text##90"
END - footer
```

You can force a page break by using the BREAK command, which can also be conditional. If you specify "BREAK number", then a page break is only inserted if there are fewer than 'number' lines left on the page (based on font 0 again).

You can control output positioning much more precisely by directly using the output position variables. These are called PX and PY. You can use and set them in any way, just like regular variables. The maximum width of the page is stored in MX, the maximum height in MY. This allows for some interesting applications, e.g. printing output at regular intervals. Here is an example which prints labels based on the fact that each page may contain 2 columns of 8 labels each:

```
VAR Num adrInc INT END -- height of each label
VAR Num adrCol INT END -- width of first colum

SET adrCol=MX / 2;
SET adrInc=MY / 8;

PROC Label
  HEADER NEWLINE END - Insert blank line at beginning of each page
  PRINT "" - forces the report to calculate the current position
  MOVETO 0 SET adrTop=PY;
  PRINT "Account number: @[cpt.nb]"
  SET PX=adrCol;
  BLOCK adr.dest NEWLINE SET PX=adrCol;
  IF ~EMPTY adr.zip THEN PRINT "@[adr.zip] - " END
  PRINT "@[adr.city]" NEWLINE
  SET PY=adrTop + adrInc; -- set top position for next label
  IF PY > MY THEN BREAK END;
END - Label
```

### 2.18.1.10 Column printing

If you just want to print a few strings, but lign them up in columns, you can do so by using the INCOLUMN statement. This initialises the column mode, in which text is automatically printed in columns of fixed width and height, both of which you can specify as parameters. In this way, you don't have to worry about the positioning of strings, you can simply print them successively, knowing that they will lign up into columns, even if you don't know how many there will be or what structure to give them. This also works with block of multi-line text via the BLOCK command.

```
PROC testParams
  PRINT 1 "Test No. @[test.idx] - Sample @[pack.idx]:"
  MOVETO 100 PRINT "Date of test: @[test.date]" NEWLINE 2
  PRINT "Test specifications: " MOVETO 25 BLOCK test.znote NEWLINE
  IF NOT EMPTY test.note THEN -- execute only if variable not empty
    PRINT 3 "Comment: " MOVETO 14 BLOCK test.note NEWLINE
  END
  INCOLUMNS 80,2 -- starting 2 column mode, width 80 character
  CALL threat
  PRINT "Barrel: " PRINT "@[test.barrel]" NEWLINE
  PRINT "Backing: @[test.backing]" NEWLINE
  PRINT "Type of powder: @[test.powder]" NEWLINE
  PRINT "Shooting distance: @[test.dist] m" NEWLINE
  PRINT "Drag correction: @[test.drag] %" NEWLINE
  PRINT "Angle of shot: @[test.angle] Deg" NEWLINE
  PRINT "Room temperature: @[test.temp] °C" NEWLINE
  PRINT "Room humidity: @[test.humid] %" NEWLINE
  IF (test.type = 2) & NOT EMPTY test.conditionned THEN
    PRINT "Conditionned" NEWLINE
  END
  PRINT "Test conditions: @[test.cond?Cond] "
  IF NOT EMPTY test.condNote THEN BLOCK test.condNote END
  NEWLINE
  NEXTCOL 80 -- move to next column, also of width 80
  NEWLINE
  PRINT "Sample @[pack.idx] Identification: "
  MOVETO 25 BLOCK pack.znote NEWLINE -- print a multi-line text block
  PRINT "Total no. of layers: @[test.nbLayers]" NEWLINE
  CALL layers
  PRINT "Weight as is: @[pack.dry] g" NEWLINE
  IF NOT EMPTY pack.wet THEN
    PRINT "Weight Wet: @[pack.wet] g" NEWLINE
    PRINT "Water absorbtion: @[pack.absorb] %" NEWLINE
  END
  IF NOT EMPTY test.dim THEN
    PRINT "Dimensions: @[test.dim]" NEWLINE
  END
  IF NOT EMPTY pack.density THEN
    PRINT "Density (as is): @[pack.density] g/m2" NEWLINE
  END
  IF NOT EMPTY pack.thick THEN
    PRINT "Thickness: @[pack.thick]" NEWLINE
  END
  PRINT "Envelope: @[pack.env]" NEWLINE
  PRINT "Sewing/curing: " MOVETO 16 BLOCK test.sewing NEWLINE
  ENDCOLUMNS -- end column mode
  NEWLINE
END - testParams
```

### 2.18.1.11 Including display objects

The special interest here is the possibility of inserting graphical objects in a report with parameterised OBJECT statements: the string following the OBJECT statement up to the first “;” character defines the class name of the object, the rest of the string is passed on to the object in its name field as parameter. This parameter can then be interpreted by the object to generate an appropriate representation.

```
PRINT "" BREAK 15 - force position calculation, check if enough space on page
CALL testCommon
OBJECT "BTable;FRAG" -- This inserts a BTable object with parameter FRAG

PRINT "" BREAK 15
OBJECT "StatGraph;FRAG" -- This inserts a statistics graph with parameter FRAG
NEWLINE 2
```

You can insert all kinds of graphical display objects, including bitmaps

```
OBJECT "Bitmap;\photos\demo\A3.jpg" -- specifies path of bitmap to be inserted
```

or specific report tables

```
OBJECT "Table;@[client.name]##20;@[client.first]##30;@[client.adr]##50" NEWLINE
-- generates a table with multiple columns, which may contain multi-line objects
-- multiple tables that follow each other and that are of the same format will
-- form a single, continuous table
```

Please refer to module RptTbl.ob2 for more details on how to use report tables.

### 2.18.1.12 Include files

You can also include code from other files by using the PARSE command:

```
PARSE "file_name.INC"
```

The code in the include file will be considered just as if it were from the main report file. Include files may include other files as well, recursively.

The file name is going to be translated before the file is included, so it may contain variable names, such as “[filename]@[counter].[extension]”, where “filename”, “counter” and “extension” are all values that were declared either globally or locally for the report.

### 2.18.1.13 Dynamic code inclusion

You may also include dynamic code, i.e. report code that is generated while the report is being executed by a call to an object or the report control procedure. For these, you should always use a procedure, along with the command `DYNPARSE`, as in the following code extract:

```
PROC StatTbl
    DYNPARSE "Stat.INC"
END - StatTbl
CALL StatTbl
```

A typical use is the following statistics table, `Stat.INC`. The application generates this file either before or while running the report by simply writing report commands as strings to the it:

```
OBJECT          5          "Table;CATEGORIE##12;@^CHF##9;@^EUR##9;@^GBP##9;@^Divers
Euro##9;@^USD##9;@^CAD##9;@^Zone Dollar##9;@^JPY##9;@^Autres##9;@^TOTAL##9"
NEWLINE
OBJECT "Table;ESPECES##12;@> 219'095.34//73.43 %##9;@> -161'489.05//54.13 %##9;@>
//##9;@> //##9;@> 175'220.79//58.73 %##9;@> //##9;@> //##9;@> //##9;@> //##9;@>
232'827.08//78.04 %##9"
NEWLINE
OBJECT "Table;MARCHE MONETAIRE##12;@> //##9;@> //##9;@> //##9;@> //##9;@> //##9;@>
//##9;@> //##9;@> //##9;@> //##9;@> //##9;@> //##9;"
NEWLINE
OBJECT "Table;OBLIGATIONS##12;@> //##9;@> 49'761.36//16.68 %##9;@> //##9;@>
//##9;@> //##9;@> //##9;@> //##9;@> //##9;@> //##9;@> 49'761.36//16.68 %##9"
NEWLINE
OBJECT "Table;ACTIONS##12;@> 7'140.00//2.39 %##9;@> 8'277.23//2.78 %##9;@>
//##9;@> //##9;@> 359.87//0.12 %##9;@> //##9;@> //##9;@> //##9;@> //##9;@>
15'777.11//5.29 %##9"
NEWLINE
OBJECT "Table;FONDS##12;@> //##9;@> //##9;@> //##9;@> //##9;@> //##9;@> //##9;@>
//##9;@> //##9;@> //##9;@> //##9;"
NEWLINE
```

The power of this system will easily become apparent to you, when you compare the complexity of nicely formatting a table with text and numbers for printing and display against generating a simple text file as the one above. To wit, here is the corresponding output (in French, as it is copied from an actual portfolio management system for French speakers):

CATEGORIE	CHF	EUR	GBP	Divers Euro	USD	CAD	Zone Dollar	JPY	Autres	TOTAL
ESPECES	12'746.63 6.58 %	12.39 0.00 %			2'304.69 1.19 %					15'063.71 7.78 %
MARCHE MONETAIRE					-116'720.09 -60.25 %					-116'720.09 -60.25 %
OBLIGATIONS	60'553.54 31.26 %	26'311.85 13.58 %			89'150.66 46.02 %					176'016.06 90.86 %
ACTIONS	62'725.00 32.38 %	56'534.58 29.19 %								119'259.57 61.56 %
FONDS										
CHANGE										
TOFF		98.34 0.05 %								98.34 0.05 %
TOTAL	136'025.17 70.22 %	82'957.15 42.82 %			-25'264.74 -13.04 %					193'717.58

Figure 37 - Sample of Report Table

### 2.18.1.14 Use of the Command method within a report script

As you may have noticed, persistent objects have a Command method, which answers to a certain number of command strings; You can use this feature from within a report script through the CMD command, which may be used either as procedure or as condition, eg.:

```
CMD Window.client "RESET DEFAULT"
```

or

```
IF CMD DbFile.client "GET FIRST" THEN
```

If the command returns a string, the value of this string will be saved in the report variable "Answer", so you can access it as:

```
VAR Container "myValue" END
IF CMD Container.myValue "FIND Str client.name" THEN
  CMD Container.myValue "GET" PRINT "@[Answer]"
END
```

will print the contents of the variable contained in myValue.

## 2.18.2 Access to data sources

Very often, you will want to use some data source that is already defined within your application in a report, such as the contents of a scroll window. This source comes complete with access method, filter and initialisation. There is a very simple method for doing so:

First, you need to identify your data source with a unique name. You can do so within A3Edit, by accessing the "Source settings" on a scroll window: the first field is the source name.

You may then access this source from within your report in the following way:

Define a "Stepper" object and assign the desired source to it by name, then, you can access it with CMD commands. Example:

```
DEF Stepper "src" ENDOBJ CMD src "ASSIGN mySource"
CMD src "SETINFO" -- Initialise the filter
IF CMD src "FIRST" THEN -- Get the first element
  REPEAT
    -- do what you wish to do for each element
  UNTIL NOT CMD src "NEXT" -- until there are no elements left
END
```

Here is a generic report, that will print out various lists with different layouts, based on the variable "auxData", which must be set before the report is called. To be fully generic, you can assign the name of the source in an active scroll window to such a variable and then call the report, which will automatically print a list of all the elements in the list that the user is editing.

```
DEF Stepper "dataStp" ENDOBJ CMD dataStp "ASSIGN @[auxData]"
VAR Str "head" LEN 300 END
VAR Str "fmt" LEN 300 END
VAR Str "ttl" LEN 80 END

HEADER
  OBJECT 1 "Data list: @[ttl]##50;@NF:@>@[RptDate]##30;@NF:@>Page @[PageNb]"
  NEWLINE
  IF NOT EMPTY head THEN OBJECT 1 "Table;@[head]" NEWLINE END
END -- header

PROC loop
  CMD dataStp "SETINFO"
  IF CMD dataStp "FIRST" THEN
    REPEAT
      OBJECT "Table;@[fmt]" NEWLINE
      -- The @! Forces a double translation; first, the contents of each
      -- variable is translated, then the resulting string is translated
      -- again, which allows for embedded variables
    UNTIL NOT CMD dataStp "NEXT"
  END -- if first
END

IF auxData = "centryList" THEN
```

```

STORE ttl "Country"
STORE head "Code##15;Nom##60"
STORE fmt "@[cntry.code]##15;@[cntry.name]##60"
-- use STORE, not SET, so the contents of ttl, head and fmt
-- is not translated immediately
CALL loop
ELSIF auxData = "contactList" THEN
STORE ttl "Types of contacts"
STORE head ""
STORE fmt "@[kind.name]##60"
CALL loop
ELSIF auxData = "cityList" THEN
STORE ttl "Cities"
STORE head "Nom##40;CP##15;Pays##40"
STORE fmt "@[city.name]##40;@[city.zip]##15;@[city.cntry]##40"
CALL loop
END -- getInfo

```

Note that the nature of the data source is completely hidden. It could be a database table, a memory list, an object group or any other sequential data structure for which an `Sequence.Source` access method has been defined.

### 2.18.3 Direct access to database tables

Instead of using pre-defined data sources, you may also access database tables directly. In this case, you use `CMD` to operate on the desired object directly.

```

VAR Bool exit END
CMD DbFile.invoice "RESET"
SET invoice.date=RptDate;
-- set invoice date to report date (usually current date)
IF CMD DbFile.invoice "GET GTEQ BY date" THEN
-- Get first invoice record that is greater or equal
-- to the RptDate value when searching by the date key
REPEAT
CLEAR exit
IF invoice.date # RptDate THEN
-- stop when invoice date is not RptDate anymore
SET exit=TRUE
ELSE
OBJ "Table;@[invoice.nb]##10;@>@[invoice.amount]##20"
NEWLINE
END
IF NOT CMD DbFile.invoice "GET NEXT" THEN SET exit=TRUE END
-- stop when no further record are found
UNTIL exit
END - if any found

```

### 2.18.4 DbReport Extensions

The module DbReport adds several extensions useful for working with databases, which is what you very often do in reports. To take advantage of these extensions, simply declare your report class to be based on

`DbReport.Report`.

The following features become active:

- Each time a page is printed, all of your database files are first saved and then restored after the page was entire produced. This avoids disruptions due to database access operations during the generation of the report output and means you don't have to worry about it in your report command handling procedures.
- You gain special commands, that allow you to generically access databases, database files and database fields.

The following special objects are declared and attached to your report:

- `RptDb` current database
- `RptFl` current database file
- `RptFld` current field

Note that these objects are of type "Persist.Container", so that you may simultaneously refer these objects through their name from the report as well as get at the objects they refer to.

The commands that are used to access them are the following:

- `DbFirst / DbNext` Sets `rpt.db` to first / next available database in application
- `DbFirstFl / DbNextFl` Sets `rpt.fl` to first / next available file in `rpt.db` database
- `DbFirstFld / DbNextFld` Sets `rpt.fl` to first / next available field in `pt.fl` file

## 2.18.5 Generic Database Report

Here is a generic database browser report, which allows you to list the complete contents of a database. You will have to admit that it is rather compact code:

```
-- This script will browse all databases of the current application
-- and list their structure and contents

FONT 0 "Arial;h=-10;p=v;a=2"
FONT 1 "Arial;h=-12;p=v;a=2"
FONT 2 "Arial;h=-12;p=v;a=3"

VAR Str line LEN 500 END
VAR Num cnt INT END
VAR Real len FIX DIGITS 0 END

IF DbFirst THEN
  REPEAT
    CMD Container.RptDb "SUBNAME"
    OBJECT 2 "Table;Database @[Answer]##100" NEWLINE
    IF DbFirstFl THEN
      REPEAT
        CMD Container.RptFl "SUBSHORT"
        OBJECT 1 "Table;File @[Answer]##100" NEWLINE
        SET cnt 0 PERFORM DbFirstFld REPEAT INC cnt UNTIL NOT DbNextFld
        SET len "100 / @[cnt]"
        IF EMPTY len THEN SET len "10" END
        IF DbFirstFld THEN
          SET line ""
          REPEAT
            CMD Container.RptFld "SUBSHORT"
            IF NOT EMPTY line THEN SET line "@[line];" END
            SET line "@[line]@[Answer]##@[len,&]"
          UNTIL NOT DbNextFld
          OBJECT "Table;@[line]" NEWLINE
        END -- DbFirstFld
        IF CMD Container.RptFl "GET FIRST" THEN
          REPEAT
            IF DbFirstFld THEN
              SET line ""
              REPEAT
                CMD Container.RptFld "GET"
                IF NOT EMPTY line THEN SET line "@[line];" END
                SET line "@[line]@[Answer]##@[len,&]"
              UNTIL NOT DbNextFld
              OBJECT "Table;@[line]" NEWLINE
            END -- if firstFld
            UNTIL NOT CMD Container.RptFl "GET NEXT"
          END -- if
          NEWLINE
        UNTIL NOT DbNextFl
      END -- DbFirstFl
    BREAK
  UNTIL NOT DbNext
END - DbFirst
```

### 2.18.6 Insertig objects to fill in page

Some page layouts require that you generate the complete report before you can fill in some blank space. A typical layout example would be an invoice, where you have a list of items, followed by a total and maybe some additional text, e.g. item-specific gurantee instructions, a list of payments etc.

You don't know how much space the variable area at the end of the last page will be, because the total area may change based on additional variable lists, text – which can change in area based on font and output device properties – etc. But you would like to distribute whatever blank space there is so that it fills in the largest item list.

You can achieve this with the following piece of code:

```
VAR Num insPX INT END CLEAR insPX
VAR Num insPY INT END CLEAR insPY
VAR Str fillFmt LEN 100 END

PROC FillTbl
  -- procedure fills item table with blank line items until page is full
  IF ~EMPTY insPY THEN
    IF MaxY + AvgHg * (2 + Footer) < MY THEN
      SET PX = insPX; SET PY = insPY;
      REPEAT
        INSERT 0 "Table;@[fillFmt]" NEWLINE
        SET PX = insPX;
      UNTIL MaxY + AvgHg * (2 + Footer) >= MY;
      IF MaxY > MY THEN DROP END
    END -- if space left
  END -- if insPY
END - FillTbl
```

Before calling this procedure from your own code, you must:

1. Set the variable insPX and insPY to the current position at the location where you want to insert the blank space: SET insPX = PX; SET inxPY = PY;
2. Set the variable fillFmt to the format string you wish to use with the table, e.g.  
SET fillFmt = "@LINE=|4//;##14;##18;##34;##10;##12;##12";
3. Call FillTbl at the end of the report page you wish to fill

# India House

Rue Louis de Savoie 26/28, 1110 MORGES, Tel: +41/21 801 22 44

*Rue de Lausanne 37/25*  
1400 Fribourg  
Tel: 026 341 8877

*Benjamin Constant 1*  
1003 Lausanne  
Tel: 021 323.09.10

*Avenue du Casino 47/49*  
1820 MONTREUX  
Tel: 41 21 961 13 66  
Fax: 41 21 961 13 30

*Rue Louis de Savoie 26/28*  
1110 MORGES  
Tel: +41 21 801 22 44

*Grand-Rue 10*  
1260-NYON  
Tel: +41 22 361 20 23  
Fax: +41 22 361 23 03

Date : 03.02.2006  
Bill No. : 2006-MG  
          -000623

Customer: Oberson Franc  
          0218004082

TVA No.  
457789

Status : Paid

Code	Brand	Identification	Quantity	Selling Price	Total Price
0106000110	Carpets(Tapis)		1	299.00	299.00
0106000177	Carpets(Tapis)		1	990.00	990.00
Point of insertion insPY					
Discount					799.00
<b>Total</b>					<b>490.00</b>

**Payment Details :**

Date	Mode	Amount	Days (if BVR)
03.02.2006	Cash	490.00	

Amount           **490.00**  
Paid:             **0.00**  
Balance:

*INDIA HOUSE vous remercie pour votre achat et espere vous revoir tres prochainement dans l'un de nos magasins.*  
*Pas de remboursement en numeraire           Echange ou a valoir dans les 8 jours*

Background picture generated by page header or footer

Dynamically generated address blocks

Header and line items

Variable area filled with blank lines

Area of unknown length

Area for comments, guarantee texts etc.

Redistributed white space - blank form lines inserted after last line of page was generated

INSERT lines

Dynamic list

Figure 38 - Complex form using object insertion to fill blank area

## 2.20 Automatic report generation from scroll window

RptTbl.ob2 provides a tool that allows you to generically produce a report based on a scroll window – the RptTbl.Value. Proceede as follows:

1. Create a local report variable of type TblFmt
2. Assign the name of the scroll window to this variable
3. Use the variable to generate the report title, column header and data lines

Sample code:

```
VAR TblFmt fmt END
SET fmt = windowName

DEF Stepper "dataStp" ENDOBJ CMD dataStp "ASSIGN @[fmt,#3]"

HEADER
  OBJECT "Table;@[fmt,#2]" NEWLINE - title
  OBJECT "Table;@[fmt,#1]" NEWLINE - column header
END

CMD dataStp "SETINFO"
  IF CMD dataStp "FIRST" THEN
    REPEAT
      OBJECT "Table;@!@LINE=|_@[fmt,#0]" NEWLINE
    UNTIL ~CMD dataStp "NEXT"
  END -- if first
END -- loop
```

## 2.21 Report Debugging

Report parsing works fine, but error reporting is rather sketchy, at this point. Until we get around to doing it properly, when a report doesn't work as expected, start by trying a few things before assuming that it won't work at all:

- Check that your report file (\*.FMT) and all of the include files (\*.INC) are actually present where you expect them, typically in the execution directory of the application. If the current directory is not the application directory, you may want to use the WinTools. DefaultFileName function to locate the files in the directory of the executable.
- Check for parsing errors. You can include RptDebug.ob2 into your application and then add the command
 

```
PERFORM ListRpt
```

 at the beginning of your report. This will list an abbreviated form of the report, as it was parsed, allowing you to find inconsistencies or errors.
- Make sure your application is running the code properly. That's very easy: just add a few PRINT / NEWLINE statements in strategic locations throughout your report, in places that are NOT subject to conditional execution, such as between procedures. That way, you can find out if your code was properly parsed or where to look for syntax errors.
- Find logical errors in the same way, by printing variables in the flow of report execution, as a trace.

## 2.22 Export to stream

The report output can not just be used for the display and printing, it can also be used to generate various text formats. All the standard report objects, including report tables, are able to convert textual elements so that they can be used to produce either fairly well structured flat text or field-based output for use with CSV-format files, which can then be imported by other software such as spreadsheets (Excel etc).

The programming commands are `Reports.Report.ExportToStream` and `Reports.Report.Export`. With `ExportToStream`, you can generate output to any stream, i.e. also to a `MemStream.Stream`, which includes simple strings. Using such features, you can also generate text to send as email and any other text-based use.

There are a few caveats when generating text output: you usually will not want any intermediate headers, so you should disable them for non-paged output. To do so, simply insert a test to see if the output goes to a standard device or a stream:

```
IF ~ ToStream THEN
  - print header
END
```

You can check for the output type through the following standard eval strings.

Condition	TRUE if output goes to
<code>ToDisplay</code>	a display device
<code>ToPrinter</code>	a printer device
<code>Text</code>	strictly flat text file
<code>ToStream</code>	a stream of whatever type
<code>ToFile</code>	a stream and the stream is a file
<code>FMT=XXX</code>	stream and name extension (usually file name) matches the string after the equal sign, which represents the file format extension, e.g. <code>XXX = CSV, TXT</code> etc.

When exporting to a plain text file, the output will be formatted as closely as possible as the displayed report. Tables will be reproduced and multi-line fields will be split even inside columns. Of course the resulting file may not be perfect, but as close as possible to the original.

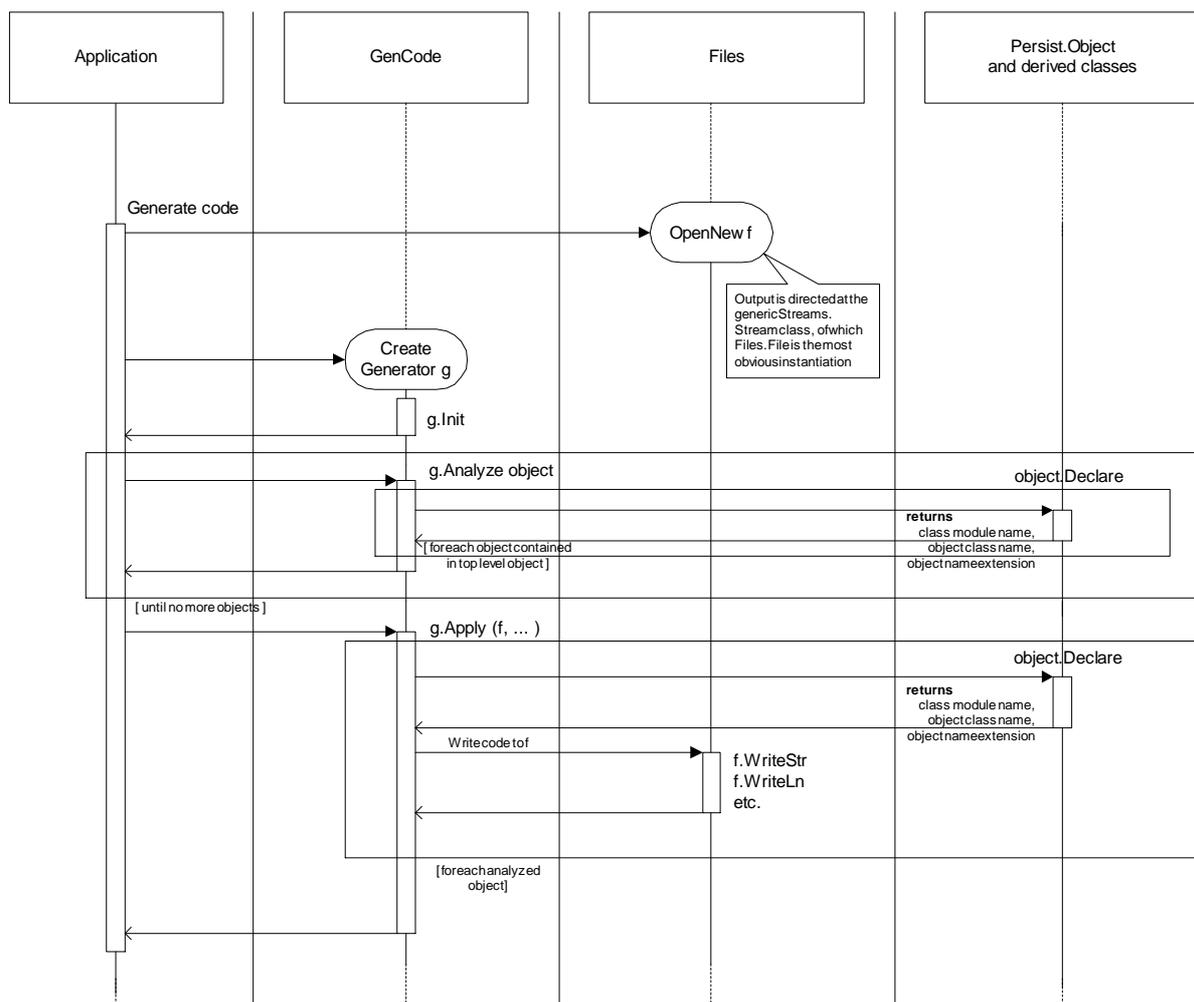
When you export to a formatted stream such as CSV, multi-line fields are automatically added as a single entry in the resulting text file, formatting will be ignored, but line breaks will be inserted. That means that a multi-line address field for example will be exported as a single field with line breaks, even if used within a table.

## 2.23 Code generation

Amadeus-3 does support a code generator module, but with a very limited scope. Code is only generated to establish a link with externally declare variables and other objects, so that they become available for direct use within an application program. The mechanism is the following:

- Create a `GenCode.Generator` object
- Create one or more objects (or a group of objects) for which you wish to generate data declarations and initialisation code
- Use the object `Analyzer` (method of `GenCode.Generator`) on the desired object(s)
- Invoke the code generator method of the generator object

Graphically speaking:



**Figure 39 - The code generator**

The code generator makes use of each persistent object's `Declare` method to find the module which exports the object, the object's type declaration and its suffix (if any). It will include all modules found by the `Analyze` method in the generated module's import list.

Declarations for commands and other constants are always generated.

Declarations for database file index numbers are generated for all files that are themselves exported.

Remember that all objects are available to the application regardless of whether they have been declared in a definition module or not, they just won't have a direct handle; you have to search for them by name or id number plus object class, or by examining all objects of a window.

For an example of the code generated by module `GenCode`, please see chapter « 2.15.4 Code for database access ».

### **Why generate code for the objects in an object script?**

Usually, you will want to generate code for:

- Global variables that are used for data entry, database access, reports etc, as well as for computations and other manipulations within the program code.
- Databases and database files, so that your access to them is simplified
- Popup menus, as they usually have to be manipulated directly
- Windows that are referenced explicitly in your code, in particular scrolling windows or top-level windows
- NB: a lot of windows will never be referenced explicitly, because they are only used for structural purpose, such as button windows in the same group as a scroll window.

In addition, by using a generated module you have more certainty that an object that you use inside your program still really exists within the object script, as it's been declared at the code level. If the initialisation procedure detects an inconsistency between requested objects and objects found in the object script, it will return an error condition.

### 3. THE APPLICATION EDITOR A3EDIT

The application editor is a tool for generating user interface objects. It's not limited to single-screen or resource item editing, as most common Windows-based tools currently available. It looks at an application as a unit with many related and linked elements, instead.

Looking at the following figure, you will quickly see the most important interface features:

1. The top level is very simple. Only a few buttons and very few menu entries are shown. Most interactions take place at the object level, either through direct manipulation (drag-and-drop, mouse-based resize and move operations) or through context-sensitive menus and dialog screens, which pop up when you click or double-click the left OR right mouse button.
2. Multiple windows may be edited simultaneously. As you will see, objects may even be moved and copied from one window to another. And as unique feature so far only found in Amadeus-3, you may drag and drop *windows* (!), adding them to other windows as child windows.
3. Composite windows are supported (e.g. combinations of frames, scrolling list, button windows). Such composite windows may even have automatic attachment constraints, which allow the group to be resized as if it was a single window.

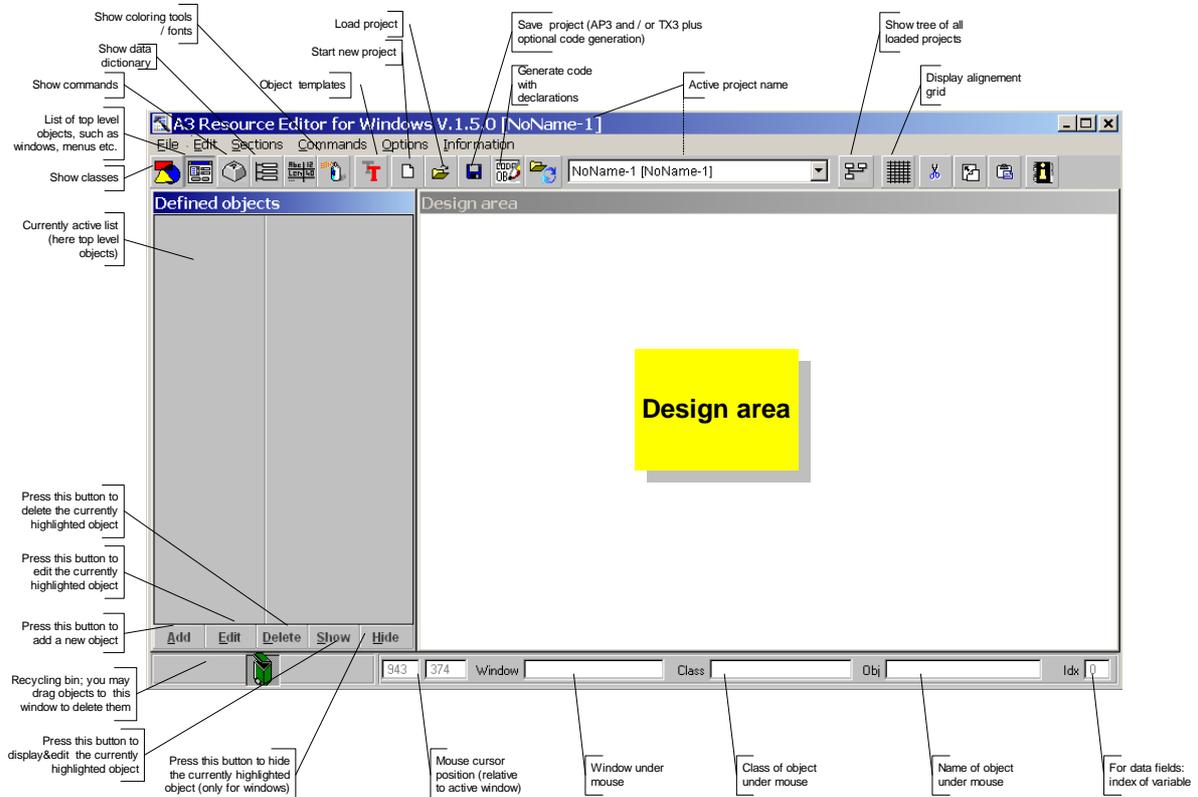


Figure 40 - The application editor

### 3.1 User interface

The following are the general procedures valid throughout the A3Edit application.

#### 3.1.1 Command keys

For object list editing, the following keys and procedures apply:

- The « Add » command may be invoked by pressing the « INS » (Insert) key
- The « Delete » command may be invoked by pressing the « DEL » (Delete) key
- The « Edit » or « Show » command may be invoked by pressing the « ENTER » key
- The « Hide » command may be invoked by pressing the « Space » key

#### 3.1.2 Mouse actions

You may invoke many actions by using the mouse instead of the keyboard. Here is a list with the simple mouse actions:

- In the object list:
  - double-click the left mouse button to show or edit the clicked object
  - double-click the right mouse button to hide a displayed window
  - Control + click the right mouse button to change an object's type
- Inside an application window, double-click the left mouse button in an empty area to add a new display object
- Inside an application window, double-click the left mouse button on top of a window object to edit the settings of that object
- Inside an application window, double-click the right button to display a context-sensitive menu, either for the window itself or for the clicked object underneath the mouse. These objects will be examined more in detail later on.
- Inside the stage window (within which all application windows are displayed for design) or the “trash” area, double-click the left or right mouse button; a popup menu will appear, containing a list of all the application windows currently displayed. By selecting a window name from this menu, the corresponding window will either be brought to the top (left button click) or hidden (right button click).

#### 3.1.3 Searching for list items by name

To find an item in a list, make that list the active window (e.g. by clicking on it). Then simply start typing the name you are looking for. A search window will pop up, showing you the text you typed. When you press enter, the highlight will move to the first matching line (if any). From there, you may reach other matching lines by pressing « Ctrl-N » (for « Next »).

#### 3.1.4 Shortcuts

The main lists may be reached with the following shortcuts:

- Alt-I: Class list
- Alt-O: Object list
- Alt-M: Command code list
- Alt-V: Data dictionary (variables)
- Alt-N: Constant list
- Alt-B: Colouring list
- Alt-T: Templates
- Alt-P: Project tree view
- Alt-L: Layer and object view
- Alt-Q: Stage objects

### 3.1.5 Shortcut handling logic

When you press Alt-P or Alt-L for access to the project tree and layer view, the corresponding window is popped up and made active unless it is already the active window, in which case it will instead be hidden.

When you press Alt-O, Alt-V etc. i.e. one of the buttons for access to the main lists, the corresponding list is made active unless it already is the active window, in which case the focus is passed to the button window.

When you press Alt-Q, the focus is set to the stage window, i.e. any user window that is currently displayed. Tab and Shift-Tab allows you to navigate normally through all user objects.

### 3.1.6 Ordering objects in a list

Where applicable, elements may be ordered by performing the following operation:

1. Select the element you wish to move to a different place within the list
2. Press and hold the left mouse button
3. Drag the mouse in the direction in which you wish to move the element. During this operation, you will notice that the moved element is being exchanged with elements in the direction of the move.
4. When the desired location is reached, release the mouse button.

Where this operation does not work, objects may not be sorted manually. It does work in the general object list, the data dictionary and the colour table. It does not work in the command table (commands are always sorted by ascending command code value).

NB: When reaching the upper or lower border of a window, the list will automatically be scrolled during this operation.

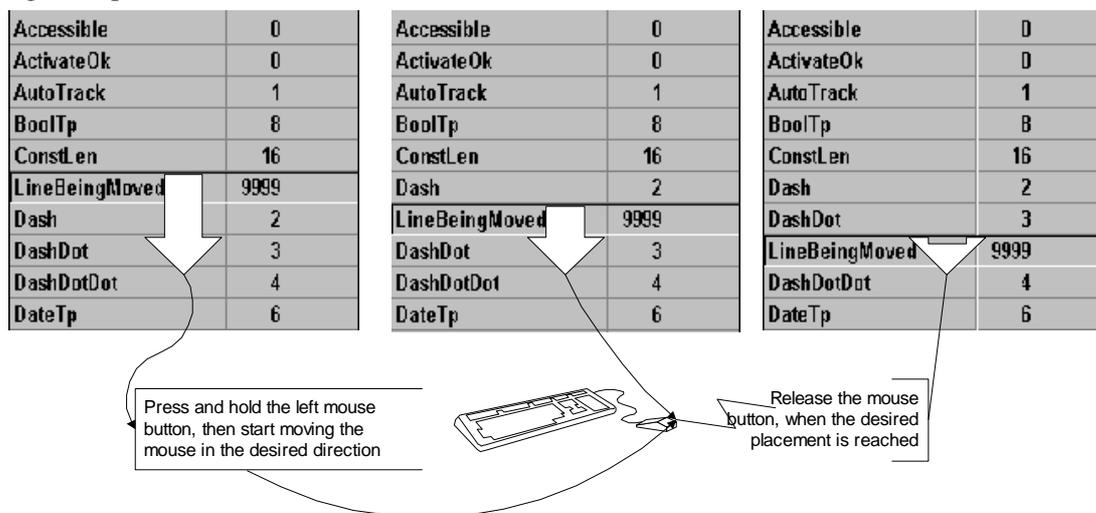


Figure 41 - Ordering objects

### 3.1.7 Object placement and attribute editing

All visual objects be clicked with either the left or the right button.

- You may move a window or an object by first selecting it by clicking the left mouse button with the object underneath the mouse pointer and then moving the mouse while keeping the left mouse button pressed.
- You may resize a window or an object by first selecting it by pressing the left mouse button on top of this object. For windows, you must also hold the “Ctrl-“ key, while left-clicking to select. This will turn on the object’s handles. Then you may move the mouse cursor over one of the corners or sides of the object and click the left mouse button. Now, while keeping the left mouse button pressed, start resizing the object as desired

NB: Both, the move and the resize operation are constrained by the window’s alignment grid. The grid parameters may be set in the window’s setting mask. To achieve unconstrained move and resize operations, press and hold a « Shift » key while performing the operation.

The resize operation is not constrained when mouse operations are used on the upper and right window border or a corner other than the bottom right one. This is due to the fact that the object will be aligned based on it’s top left corner. It could become impossible to resize the object on the top left side due to alignment constraints. Therefore, after resizing the object to the upper/left side, move it slightly to force a re-alignment to the underlying grid, if this is what you want.

You may press the “Grid” key or use the “Options / Draw grid” menu function to turn on the display of the alignment grid.

Also note that for windows that have the scroll bar attribute set, the window will automatically be scrolled when a border is reached during an object movement or resize operation.

Within a window, the following mouse actions have special meaning:

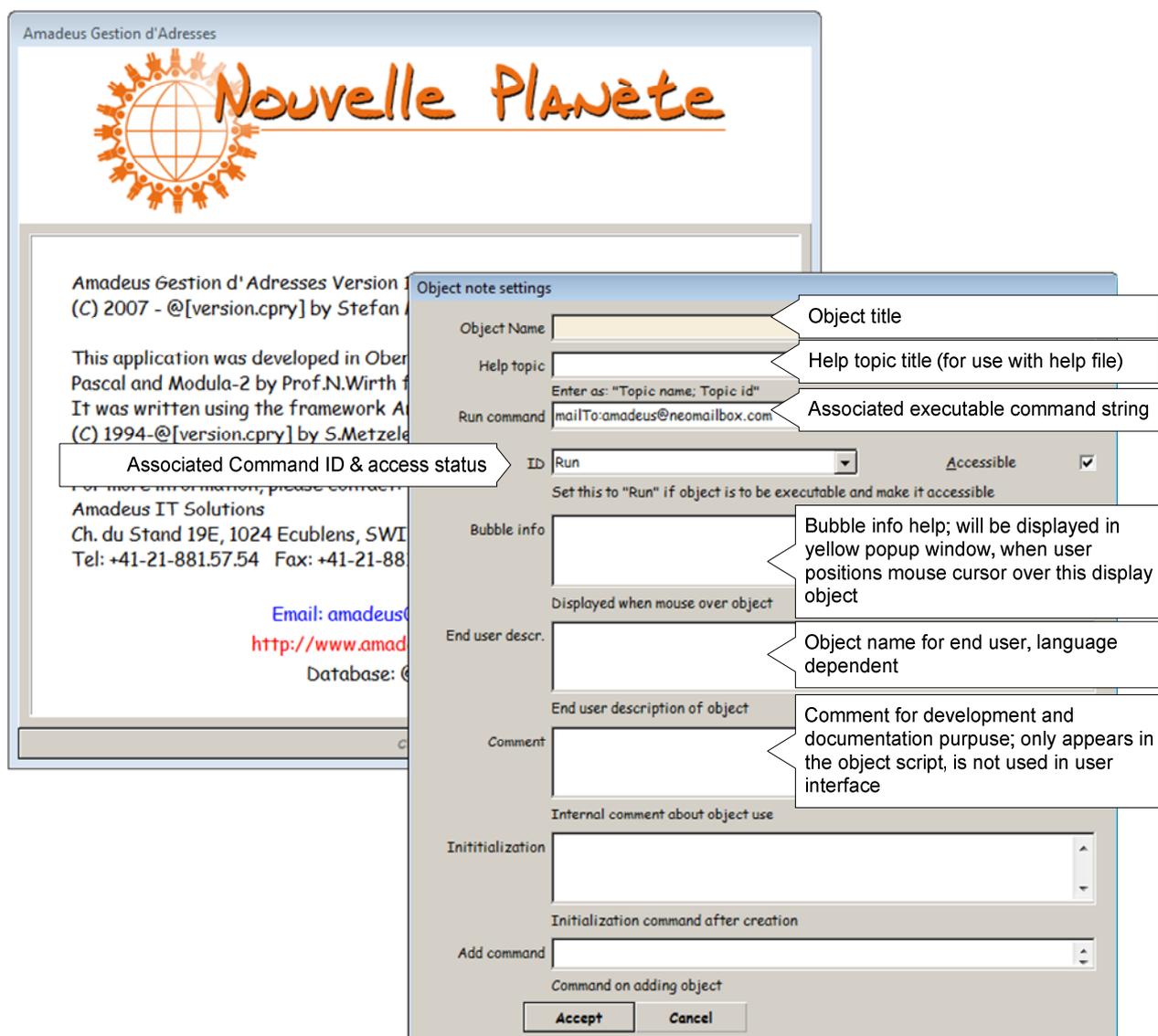
- Double-clicking the left button on top of a window object (buttons, fields, bitmaps etc.) pops up the basic object settings window, based on the object’s type.
- Double-clicking the left button on an empty location of a window (with no window objects underneath) pops up a menu prompting for an display object to be added.
- Double-clicking the right button on windows or window objects pops up a context-sensitive (object-dependent) menu. The first item of all these menus is the option to edit the object settings. Further options vary with the selected object class.
- To move or resize a window without title bar or resizable frame, move the mouse cursor over an empty area of the window, press and hold the « Ctrl » key, then press and hold the left mouse button. This will select the window (handles will show up around the window). You may now either move it by moving the mouse while keeping the left mouse button and « Ctrl » key pressed, or you may resize it by clicking and holding the left mouse button just on the edge of the window, then moving the mouse as desired. This works even for windows without resizable (thick) frames.

NB: Always press the « Ctrl » key to initiate window movement, even when the window is already selected, but don’t hold the “Ctrl” key when performing window Drag & Drop if you do not wish to copy the window.

Also note that when there are other windows just underneath the border area of a selected window, all mouse messages in that area will go to that window, disabling the resize operation. To circumvent this problem, either close the other windows or first move the desired target window to where it’s border is accessible. This problem does not exist where child windows are concerned. Their border over the owning parent window is always recognised as a boundary belonging to an object attached to the parent window.

### 3.1.7.1 Adding object notes

All context-sensitive menus for display objects contain an option entitled “Set Help / Note”. Selecting this option pops up a window with the following fields:



**Figure 42 - Object note settings**

You may also access this screen by pressing “Shift+Left-Click” on any list (objects, variables) in A3Edit.

Not the “Run” field, which, in association with the command ID set to “Run” allows you to create objects that trigger external commands, such as running a program, opening a web page or sending an email.

The above is a good demonstration of this concept: an “About” screen with an email link (command “mailto:address”) and a web link (command: “http://webaddress”).

## 3.2 Drag & Drop operations

The right mouse button usually initialises drag&drop operations, but to allow for double click actions, the drag&drop sequence only starts when the mouse has been moved more than a certain number of pixels. To perform a full drag&drop operation, you proceed as follows:

- first, select an object with the mouse, press and hold the right mouse button
- drag the object by moving the mouse while keeping the mouse button pressed,
- release the mouse button when the object is on top of it's destination

The exact location where the object will be dropped is based on the upper left corner of the dragged object, which is indicated by a cross-hair cursor. If the destination window has an alignment grid set, this alignment constraint will be applied to the dropped object.

Modifiers may be applied for special effects, in particular:

- pressing the « Ctrl » key while performing the grab operation will make a copy of the object to be dragged (where allowed). Note that this even works for windows. You will have to remember changing the object's name, though, mostly where exported objects are concerned, which may otherwise generate name clashes.
- pressing the « Shift » key while performing the drag operation will void the alignment to the destination window's grid, allowing pixel-precise insertion of the dropped object.

In some occasions, you may grab an item from a list using the standard drag&drop method, i.e. you move the mouse to a scroll list, press and hold the right mouse button on top of the desired line and - when a draggable object is displayed - you move the mouse to the destination, releasing the mouse button when the object is on top of it's drag destination.

### 3.2.1 Special Drag & Drop functions

The following operations may be performed using drag&drop operations:

#### **Deleting objects (Recycling bin)**

Most objects may be deleted by dropping them on top of the recycling bin. This works for:

- Basic window objects (buttons, fields etc.)
- Windows (this will delete the window and all associated objects)
- Variables picked up from the data dictionary list
- Colour objects (brushes and pens) etc.

#### **Changing window background colour**

To change a window's background colour directly, pick a brush from the « Colour objects » list and drop it on top of the desired window.

#### **Adding buttons to a window**

You may add a button to a window by grabbing a code from the command list. This generates a button, which you may then drop on any application window.

#### **Adding options to a menu**

You may also grab a command code - which generates a button - and drop this button on the menu editing window instead of some application window. This will cause a dialog window to pop up, prompting you for a new menu option, where the option's result code is equal to the command code that you dropped on the menu window.

#### **Making child windows**

By dropping a window on top of another (application-)window, the dropped window becomes a full child window of the destination window. Please note an important difference here:

A window is not a child window just because it has a parent. Having an assigned parent window (usually the application's main window) simply means that whenever the window is displayed, it will appear within the parent window's boundaries. Being a full child window implies being popped up whenever the parent window is displayed. In addition, certain placement constraints may be

applied to child windows of a same parent window, called attachments. You can return a child window to full autonomous status by dropping it onto the window design area.

### **Copying objects**

Of course, any object or group of objects may be copied using the procedure described in the general drag&drop operations, i.e. by pressing the « Ctrl » key while initiating the d&d operation. You may copy objects between windows or you may even copy windows. The copied window will have an index appended, which avoids having an identical name for two objects of the same class.

### **Special copy effects**

When you copy some objects, you may achieve special effects. Here are a few examples:

**Toggles** which are members of a toggle group: If the toggle group has the « Number » attribute set, (i.e. it's members' state will be reflected by a single numeric value), then the index value (result value) of each such toggle field will automatically be incremented every time you make a copy of such a field.

**Windows:** When you copy a window, the name of the new window will automatically have a number added to it's name (or the ending number incremented) to differentiate it from the original.

The contents of the original window will not be copied along with the window itself. To add a complete copy of it's previous contents, mark the contents with the rubber-band marking tool (cf. "2.13.2 Other standard mouse operations") and perform a standard copy of all the marked items, then drop them onto the new window.

## **3.2.2 Drag & Drop to scroll windows**

When dragging objects to scroll windows, the meaning of the operation changes.

- Classes derived from Sequence.Source (such as "Db source" and "List source") will be accepted as scroll source.
- Decor objects (grid & ledger or others) change the current scroll décor
- Brush objects change the window background colour
- Variables will be accepted as scroll columns
- Display objects will prompt you to decide if they should be used as column header; if yes, they replace any existing header for the selected column; otherwise they are accepted as normal display object, appearing as background of the scroll window.

### 3.2.3 Adding objects

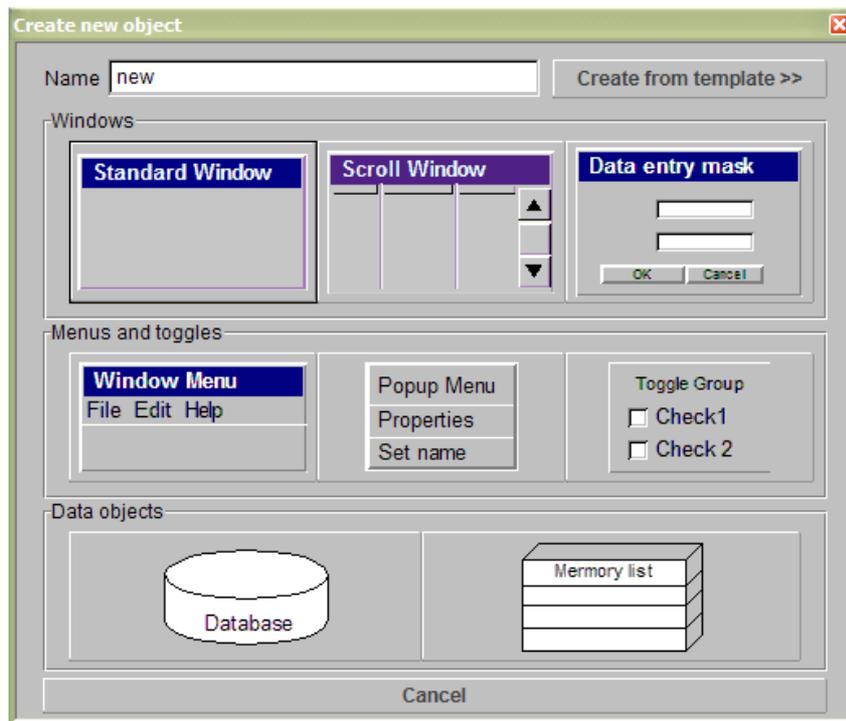
A3Edit supports many object classes and support structures, which are grouped logically into lists. To add or edit objects, first go to the appropriate list, then perform the list-specific add operation.

- To add or edit object classes, to create modules to handle the classes etc. go to the class list:



and press the “Add” button or the “Insert” key.

- To add general objects (windows, databases etc.), go to the object list  and press the “Add” button or “Insert” key. A popup window will ask you to enter the object name and to select an object class. You may choose to create the object from a template.

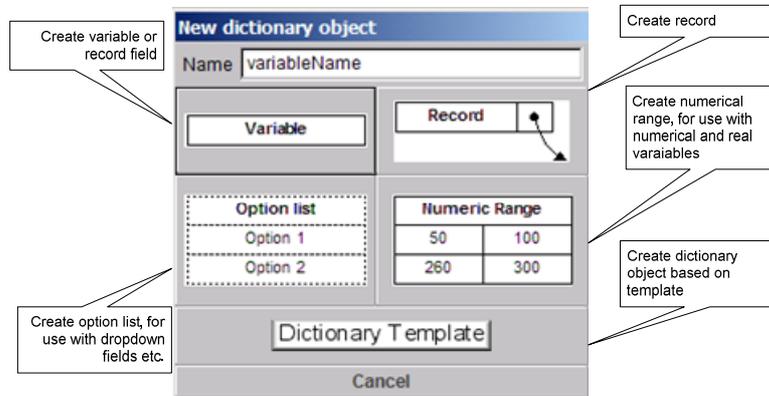


**Figure 43 - Creating new objects**

- To add command codes, go to the command code list  and press the “Add” button or the “Insert” key. Enter the command name and ID. Advice: Choose ID number to that they are logically grouped. Leave room for the addition of new codes. You have up to 1000 numbers per table.
- To add dictionary objects, go to the dictionary list . For more information, read the chapter on editing the data dictionary below.
- To add constants to be used in the definition of data dictionary objects, go to the constants list  and press the “Add” button or the “Insert” key. Enter the constant name and value.
- To decorative objects such as fonts, brush and pen object, go to the spraypaint list  and press the “Brush”, “Pen” or “Font” key, then enter parameters according to the selected object class.

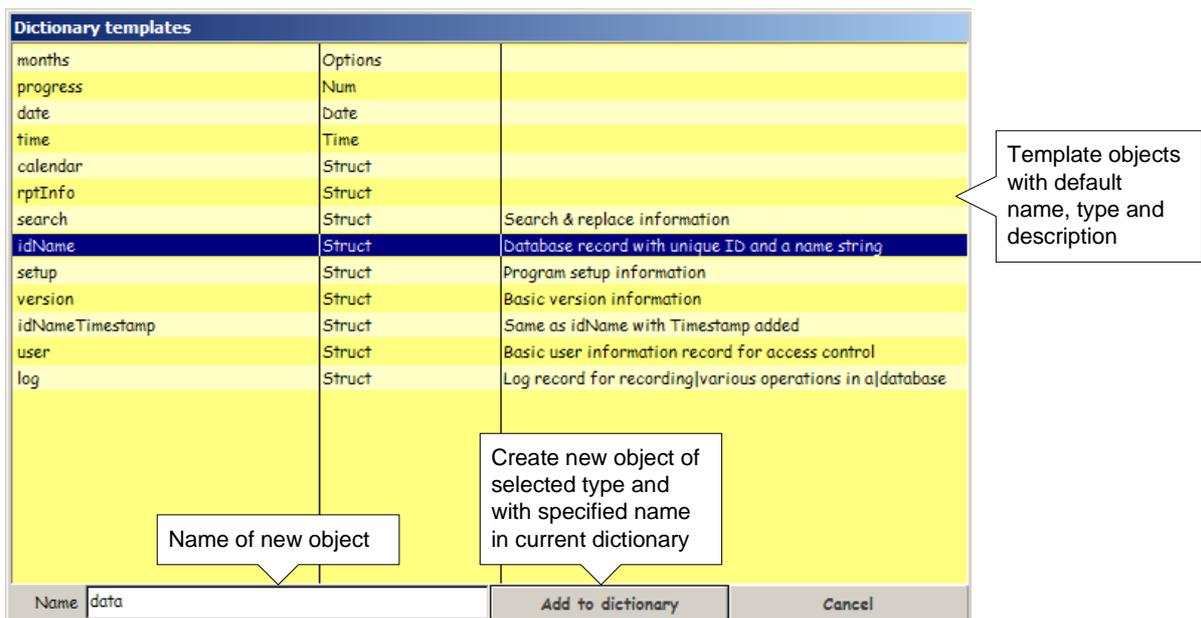
### 3.2.4 Editing the Data Dictionary

The data dictionary supports variables, records, option lists and numeric and real number ranges. To create an object of a specific type, press the “Add” button or the “Insert” key. Then fill in the object name and select the type of object you wishf to create. A new window may appear with class-specific information. Fill in required and optional parameters accordingly.



**Figure 44 - Creating a new dictionary entry**

If you choose the “Dictionary template” option, you will be prompted to select one of the template data dictionary items. Chose one, it will be inserted just above the currently selected item in the list.

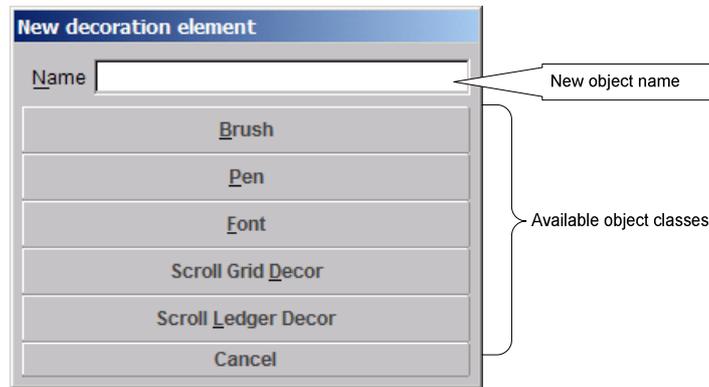


**Figure 45 - Creating dictionary object from tempalte**

### 3.3 Editing decorative objects

In the list with the spray paint logo, all decorative objects are grouped together:

- Brushes
- Pens
- Fonts
- Scroll decors



**Figure 46 - New decorative object**

To create a new object in this section, press the Add button, then specify a name and choose the appropriate object class. Then enter class-specific parameters.

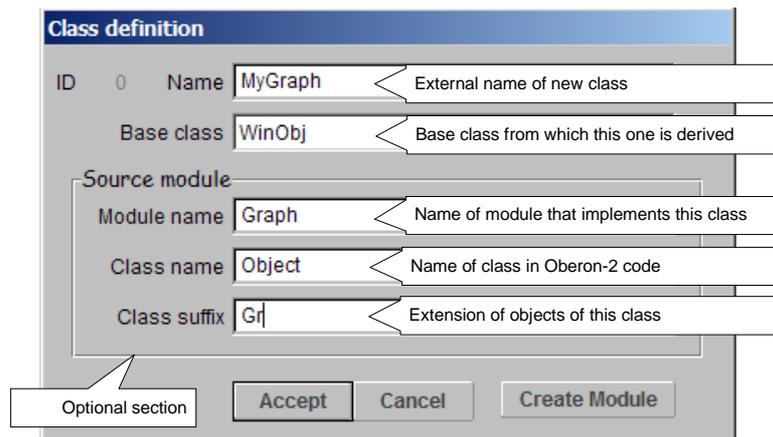
All of the decorative objects can be dragged & dropped to various objects in the design area :

- Brushes are used generally to define background colours for windows, fields etc. Drop them on windows, fields, coloured drawing objects to change their colour.
- Pens are used to draw lines, outlines of rectangles etc. You can drop them on drawing objects.
- Font objects can be used to change the font used in a specific field or object. You can drop them on fields, drawing objects, labels etc.
- Scroll grid decors specifies just if vertical and / or horizontal lines should be drawn on a scroll window. Drop them on scroll windows to change their appearance.
- Scroll ledger décors are used for the bicoloured scroll lists. They define 2 colours and the option to draw vertical and / or horizontal lines. Drop them on scroll windows to change their appearance.

### 3.3.1 Adding application-specific classes

Amadeus-3 is fully object-oriented and hence supports application-defined object classes. The obvious problem with supporting unknown classes in the user interface design tool is that precisely these classes are not known to the design tool. Hence we need a tool to identify unknown object classes and to integrate objects belonging to such external classes into various sections of the application.

To generate new object classes, first go to the Class list (Alt-L or ) and press “Add”. Fill in the following mask and if you want to create an Oberon-2 module from a standard template, press the “Create Module” button.



Class definition	
ID	0
Name	MyGraph
Base class	WinObj
Source module	
Module name	Graph
Class name	Object
Class suffix	Gr
<input type="button" value="Accept"/> <input type="button" value="Cancel"/> <input type="button" value="Create Module"/>	

Figure 47 - Create Module from Application Class

### 3.3.2 Using application-specific classes

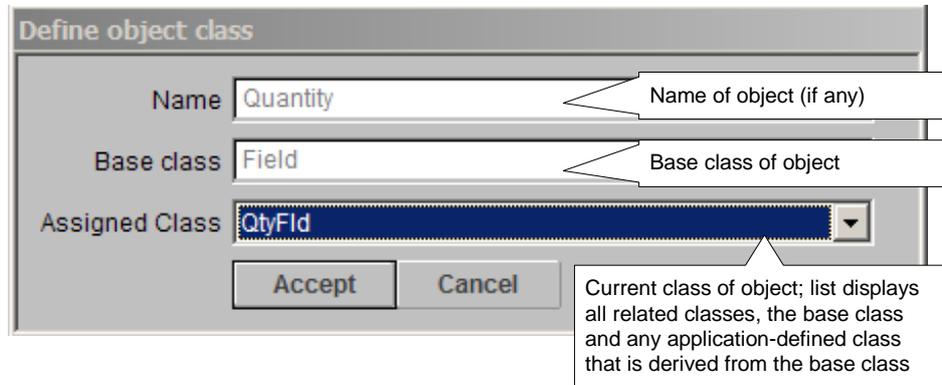
There are various ways to use application-specific classes, depending on the base class:

- Display and other object classes may be used to re-define basic objects, cf. below
- Scroll source objects may be used to define the source for scroll windows; just drag&drop the source object onto the scroll window; you may then set any additional parameters through the function “Source settings” in the scroll window context menu (Right-Click on the window).
- You may remove a scroll source through the function “Remove source” of the same menu.

### 3.3.3 Changing the class of an object

Windows, database files, variables etc. may be selected in their main list with Control-Left-Click, which will bring up a window for the selection of the desired object class to be assigned to the selected object.

Display objects may be selected when shown on screen, also with Control-Left-Click.



**Figure 48 - Define Object Class**

The extended class should be fully compatible with the base class, so it should replace the previous element perfectly.

When the application file (AP3 or TX3) is loaded by the application program, it should be linked with a module that supplies the support for the extended object class or you will only see the default behavior of the base class. The module that supplies a class should register the name of this class through the Persist.Register procedure.

If you generate the object class module through a template, make sure you actually **IMPORT** the module into your application program, but you do not necessarily have to call any procedure. If your module requires some initialisation after the application data was loaded, you can install the installation procedure with a call to Starup.InstallInitProc.

### 3.3.4 Viewing and editing window layers and objects

Amadeus-3 supports window layers, i.e. objects can be arranged in separate lists that are displayed successively, allowing each object to be displayed on top of other objects. Each layer can be selectively made visible or invisible and active or inactive. This can be great for underlaying bitmaps or overlaying frames that you want to display “out-of-sequence” or that you don’t want to react to user input.

- The layer and object list is combined into a single window that you can pop up by either pressing [Alt-L] or the “Layer” button in the button bar of A3Edit.
- To display the layers and objects of a window, display that window and activate it by pressing the left mouse button, which brings it to the top and changes the layer/object list to reflect this window; the window name (if any) will be displayed at the top of these lists.
- You can add and delete layers and set their status
- By selecting a layer in the layer list (other than the default layer) and pressing [Space], you can toggle the “Visible” attribute, hiding or displaying the contents of a given layer.

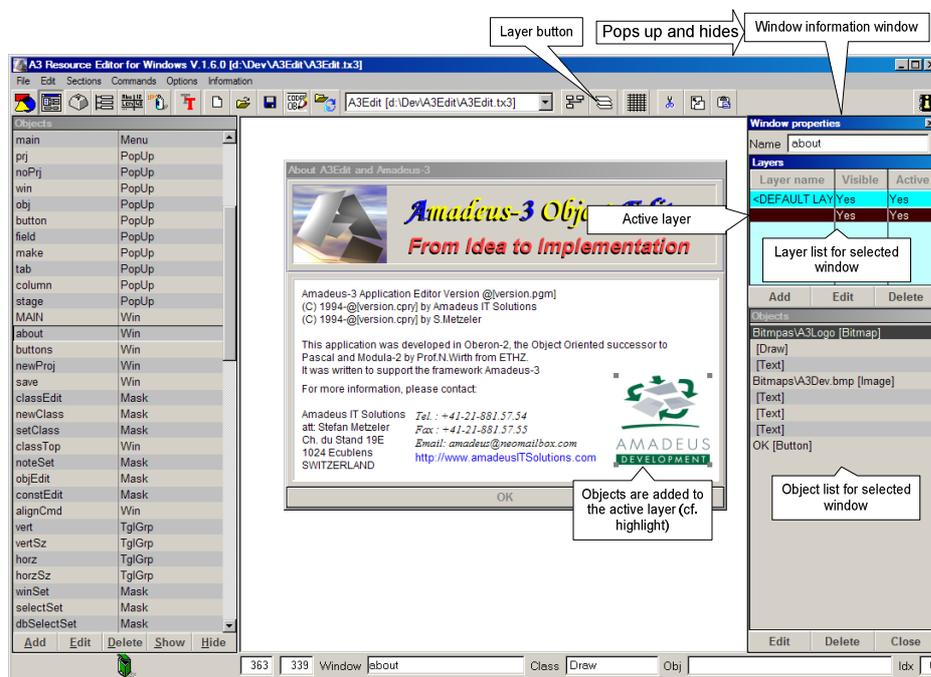


Figure 49 - Layer and object lists

### 3.3.5 Arranging window objects

For complex window layouts, you may use the following methods to organise the contents:

- Fixed coordinates for objects and child windows  
These work well for static windows, which do not need to be resized. It is also the default mode for the placement of any display object. You put it in a specific location of a window and that is where it will appear, when that window is called to the screen.
- Guidelines for flexible positioning  
the most powerful method, since it is very flexible and adaptable and is intended for dynamic, resizable windows. You start out by defining a set of guidelines within a window, then you attach display object – including child windows – to these guidelines.
- Child window attachments  
is the original method for managing resizable windows. It is still a powerful method for defining strictly child-window relations. Combined with Auto-Fill windows it can provide a simple and efficient mechanism for window arrangement.
- Application-calculated, dynamic window arrangements  
This consists in creating procedures or methods that will be called when a window has been resized by the user. Such a procedure may of course do anything you want with the window and it's contents. You can

You may – and probably will – use all of them together. The first 3 methods can be applied directly from within A3Edit. Method one is obvious – Drag & Drop objects to their destination, place and size them with the mouse etc.

To understand the second method, please read the corresponding chapter “ 2.7 Guidelines: Advanced Display Object Placement ”. A3Edit supports Guideline editing directly and of course, you can resort to editing the object script as well, though guidelines are not really very “human readable”, since it is very hard to translate abstract coordinates into a concrete representation mentally.

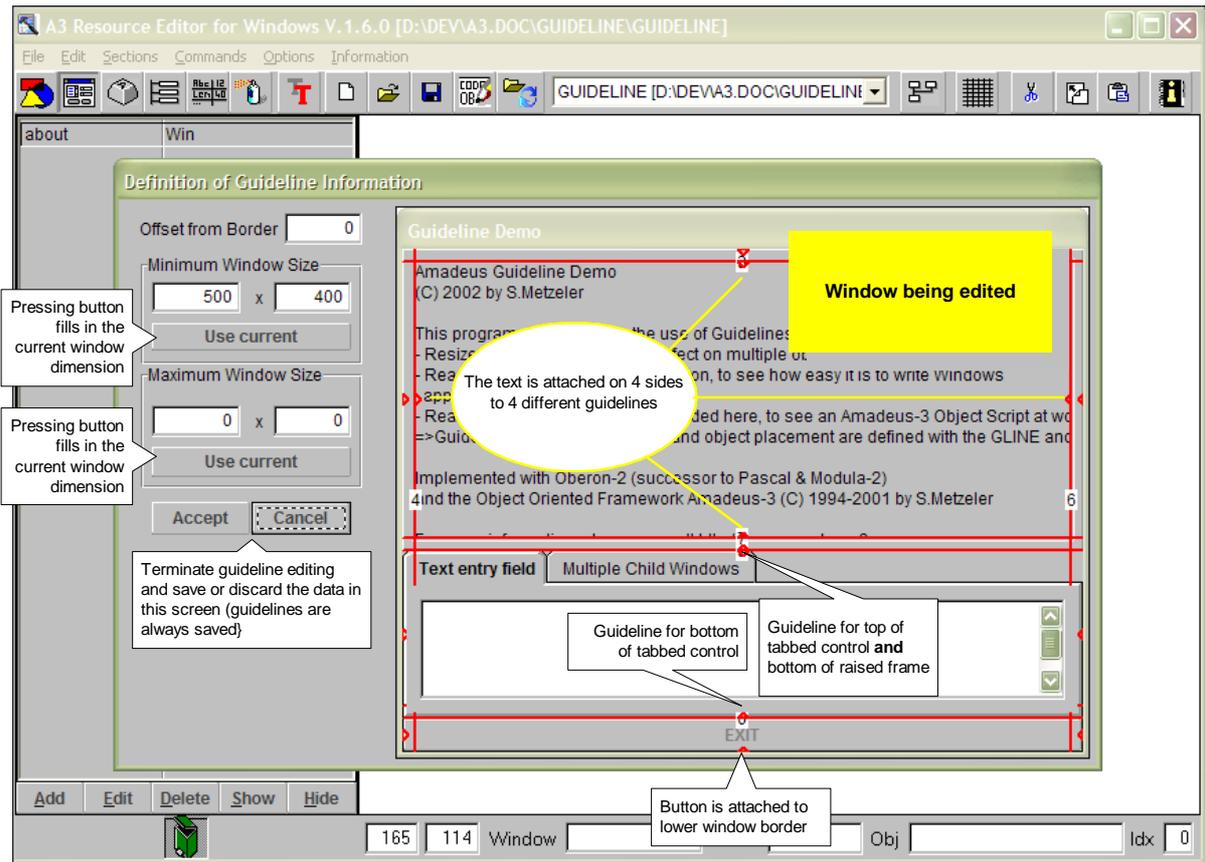
To enter Guideline Editing Mode, bring up the context menu of any window and select “Edit Guidelines”. From there on, the editing mode is changed. A frame window will appear, with the basic guide info fields on the left.

- Guidelines will be displayed as red lines with number on them.
- Links of objects with guidelines will be displayed as little triangles, that point from the guideline to the side of the object that is linked

The editing procedures are the following:

- To edit an existing guideline, double-click it and a mask with it's parameters will appear
- To create new guidelines, double-click the left mouse button, while holding down the Shift key (for horizontal lines) or the Control key (for vertical lines)
- To link an object, you can right-click the object near the side you want to link and then right-click the guideline you want to link to; also works the other way around.
- You may undo the previous LINK operation by pressing Control-U or Control-Z.
- You may also double-click a window object and then edit it's links manually, as well as define the window default border width. This is also the only way to define border links. The “Set all” button automatically sets all links to be border links.
- You may set the minimum and maximum size of the window at the same time, along with the border margin for attachments to the window border.

You can delete guidelines and links by editing them (double-left-click) and then pressing the “Delete” button.



**Figure 50 - Guideline editing**

Scroll windows may use guidelines to attach scroll columns. You can specify these attachments either by clicking the guideline, then the scroll column area or when editing the column in normal mode (not in Guideline Editing mode): right-click on the scroll column, choose “Column settings” from the menu and enter the desired guideline ID in the “Attach to” field.

The lists to the right of the client window contain all available guidelines (top window) and all objects in the window. You may edit both through the standard methods (selecting a line and pressing ENTER or double-clicking it).

You will be particularly happy about this access method, if some elements disappeared from the visible area after entering wrong values for a guideline or attachment.

### 3.3.6 Editing menus

The menu editor is invoked whenever you either do either of the following:

- you add a new menu to the object list
- you double-click a menu object in the object list
- you press the [Enter] key in the object list with a menu item selected
- Menu items are designated by the term « Menu » in the second column of the object list.

Here is a sample screenshot of the A3Edit application while a menu is being edited:

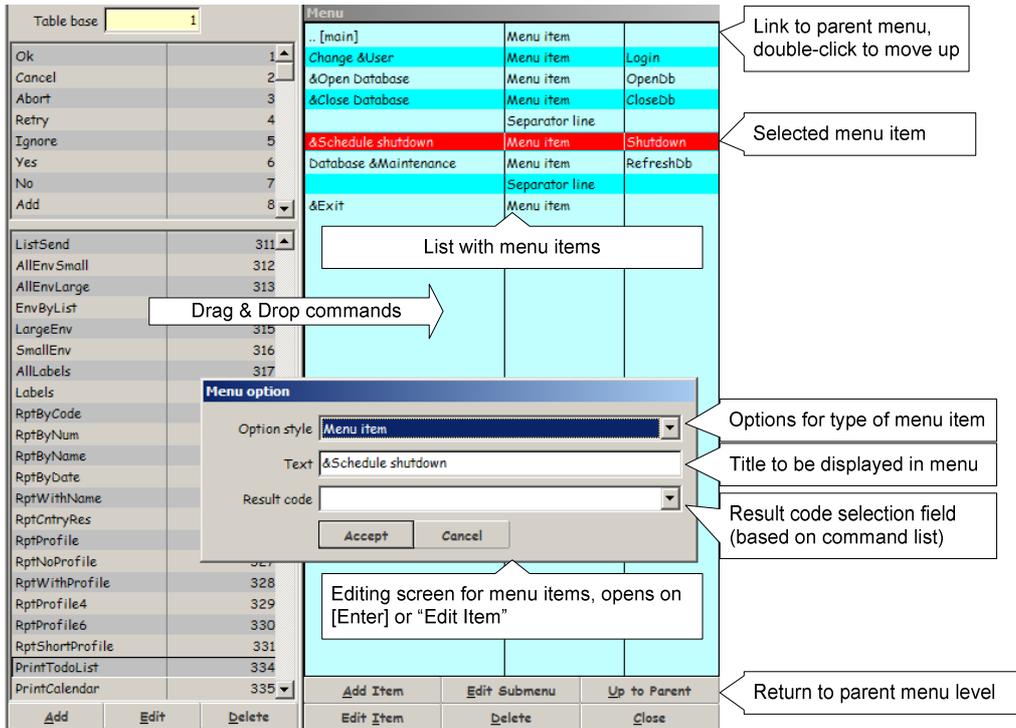


Figure 51 - Menu editing windows

You may add new menu items by dragging command codes from the command code list (with the « button » symbol) to the menu window. When dropped, the name of the command code will appear in both, the « Text » and the « Result » field of the option mask. The « Style » code is set to « Menu Item » by default.

### 3.4 Editing database structures

Based on the data dictionary definitions, you may create database tables. This is supported interactively under A3Edit as for other objects. The database editor is invoked whenever you do one of the following:

- you add a new database to the object list
- you double-click a database object in the object list
- you press the [Enter] key in the object list with a database item selected
- Database objects are designated by the term « Db » in the second column of the object list

Here is a sample screenshot of A3Edit while a database is being edited, with all the database definition windows opened:

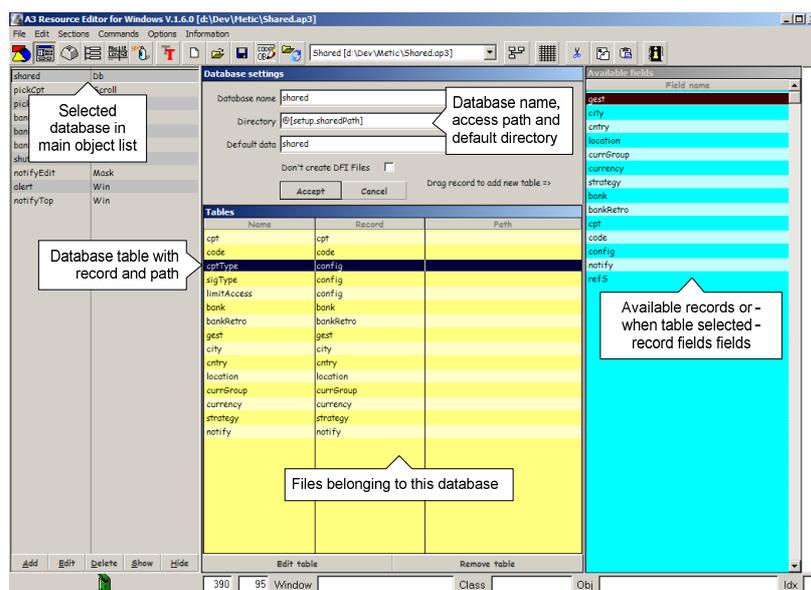


Figure 52 - Database editing

In this view, you see only the top level of the database editor. When edit one of the tables, you will see its definition and fields.

The database path is the place where the tables will be created. This field will be translated and hence may contain references to application variables. These must be properly initialized before you attempt to open or create the database. You can also use relative paths, such as “@[setup.dbPath]..\shared”. The arrangement might then look something like this:

“\Data\main” => This name will be stored in setup.dbPath

“\Data\shared” => The actual name would be “\Data\main\..\shared”

You must ensure that the variable you use – here setup.dbPath – is terminated with a ‘\’, if you use this syntax. As a matter of convention, all directory names should always be stored with the trailing ‘\’, so you never have to worry about this.

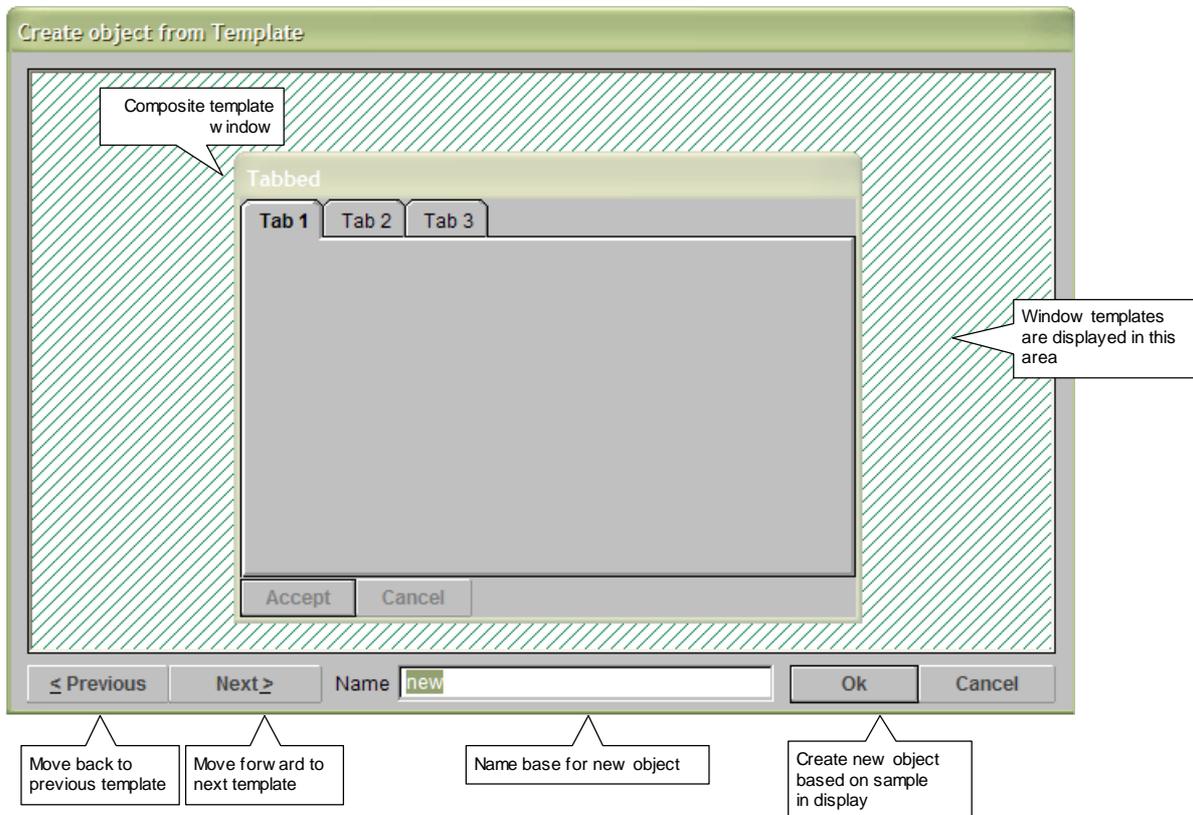
To edit the table, use either the buttons or:

- To add a new table, drag & drop a record from the list on the right to the file list. This will automatically add a new table, associated with that record.
- Editing a table by pressing the [Enter] key.
- Deleting an item from a list: press the [Del] key

For a detailed description of all the elements appearing in these windows, please consult the chapter about database support in Amadeus-3, i.e. « 2.14 Databases ».

### 3.5 Creating windows based on Templates

A3Edit contains a function for creating objects based on templates. You can either access it via the corresponding button  or the menu function “Commands / Create from Template”. When you invoke this function, the following window will appear:



**Figure 53 – Window template selection**

You may step through the list of pre-defined windows and window groups, enter a name and press “OK” to create a window based on the current template. Please note that more than one window may be created depending on the selected template. In some instances, a group of connected windows will be created, such as when you select a scroll window with attached editing mask. The editing mask may not be visible while selecting the template, but it will still be created.

The name will also be changed. The name string you enter is used as base for naming the template windows. For example when you specify the name “customer” and create a data entry mask, the resulting window in your application will be named “customerEdit” and the program object will be generated as “customerEditWin”.

You can change and extend the templates by editing the application “Template.AP3/TX3” that you can find in the A3Edit directory.

### 3.6 Starting a new project

By selecting the option “File / New Project” from the main menu, you will get the following screen:

The screenshot shows a 'New project' dialog box with the following fields and callouts:

- Directory:** A text input field. Callout: "Main directory for your project"
- Project name:** A text input field. Callout: "Name of your project"
- Application name:** A text input field. Callout: "Name of the main application file"
- Main title:** A text input field. Callout: "Main window title"
- Project author (for copyright):** A text input field. Callout: "Author or copyright holder name, will appear in About window"
- Dates:** A section containing two text input fields:
  - Copyright year:** Contains the value '2002'. Callout: "Copyright year or year range (eg. 2000 - 2002) for About window"
  - Version date:** Contains the value '31.03.2002'. Callout: "Date of first program version, also for About window"

At the bottom of the dialog are 'Accept' and 'Cancel' buttons.

**Figure 54 - New project**

When you confirm this screen, a new project directory will be created in the specified location and various files will be copied to it, including a set of application files. These default project files are found in A3Edit\Template\Project. You can modify any files therein or add additional files of your own creation. All standard text files will first be translated, i.e. any references to the above variables will be substituted in Oberon-2 source code files, TX3 application files, INI files, RC files, FMT files, INC files etc. File names themselves will be translated.

Here are the available – and useful – variables for translation:

- newProject.name = name you specified for project
- newProject.app = application name
- newProject..path = destination directory
- newProject.title = main window title
- newProject.author = author name
- newProject.year = copyright year
- newProject.date = version date

### 3.7 Starting a new application

By pressing the « New application » icon or selecting “File / New Application” from the main menu, you create an empty project. From there, you may display lists (General, Colours, Commands, Data dictionary, Constants) and add new objects.

If you want to start an entirely new project, then you should use the function “File / New project”. This will create the application’s own directory with a full set of required files for further development. You may then start by loading the new program’s application data (from the «Application».TX3 file). This will mostly have the advantage that you don’t have to hunt for a place where to store your design and you will be all set to continue.

As soon as you created a new (empty) application, all icons become accessible. You may then display any of the item lists (objects, colours, commands, variables or constants) by clicking the corresponding symbol or by selecting the corresponding option from the main menu.

### 3.8 Loading existing applications for editing

You may of course load an existing application in either of the 2 standard formats, i.e. as

- Binary file (faster) with the extension « \*.AP3 »
- Text (ASCII) file with the extension « \*.TX3 »

When you invoke one of the load/save functions, you will be prompted for a file name with the standard windows file dialog. As soon as you entered or selected a file name and pressed the « OK » button, the load process will be started.

If the loading process is successful, you will be able to access all the loaded objects for viewing or editing.

If errors occur while parsing the input script, a list of the first few errors will be displayed along with an approximate indication (line number, token number) of where the error occurred. In the case of Text files, this probably means that a manual modification was incorrect. There are several frequent causes for syntax errors in object scripts:

- Misspelled names or identifiers
- Incorrect identifier ordering (cf. module « 2.3.5 Ordering of keywords » )
- Missing object classes; one of the required object classes is not available
- Objects are not correctly sorted, i.e. referenced objects don’t precede the object referencing them.

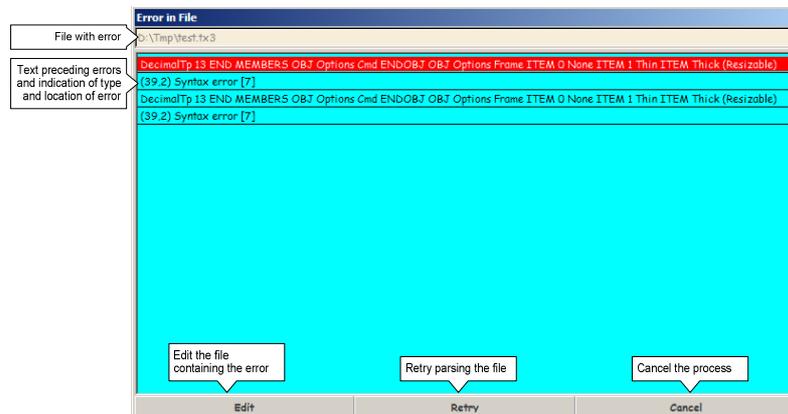
All of the above are usually enforced by A3Edit when it generates the object script. So it’s likely that any such problems arise due to manual modifications to the object script.

Should you run into a situation where you have a complex chain of dependence between different project files, you might want to place some objects into their own, lower level, project file, so that they become easily accessible as shared objects.

If some objects contain mutual references and can’t be broken up, you may have to resolve the issue in your application code after all the project files have been loaded.

### 3.9 Handling Parsing Errors

When syntax errors are detected while parsing an object script, the following window will appear:



**Figure 55 - Parsing error display**

Several commands are available for this list:

1. **Edit:** This will invoke the editor that is associated with the « TX3 » file extension in your system. Under Windows, this association is created using the file manager or some other tool. It is recommended to associate TX3 with a text editor and AP3 with A3Edit.
2. **Retry:** After you made modifications to the object script file, you may press this button to parse the script again.
3. **Cancel:** If the file cannot be parsed correctly even after modifications were made or you do not wish to continue this process, press the « Cancel » button.

The notified errors specify the general type of error and in parenthesis you find the line and token number at which the error occurred.

The list of errors will never be exhaustive, as some errors may make correct parsing of the following definitions impossible. This should be acceptable, as the object script is not meant for intensive manual editing and a great level of syntactic complexity..

*You are strongly advised to backup your object script when storing a new version (flag in the save dialog box), at best once per session.* It is very easy within the editor to make major changes, such as deleting records and all associated fields, entire windows or window families etc. All such manipulations could wrack havoc on a complex project definition, but are easy to recover from with a recent backup of the TX3 (or BX3 for backup) files. These files are really helpful, as you can run a direct text compare operation, then cut and paste definitions between an old and a new version of a given object script.

### 3.10 Loading an application by specifying a start-up parameter

You may pass the name of the application to be loaded as a parameter when starting the A3Edit application. How this is done is system-specific. On GUI systems that support a command-line interface, you may pass the parameter by typing it after the name of the command. Otherwise, you will have to specify it by entering it as special setting on the icon or menu option that you will use to start A3Edit.

NB: The extension of the file you load determines whether A3Edit expects it to be text or binary data. If the extension specified is « .AP3 », A3Edit assumes that it's a binary file. If « .TX3 » is specified, A3Edit tries to interpret the file contents as text file.

### 3.11 Saving application data

For saving the currently loaded application data, you have 3 commands available, represented by the three « Floppy disk » symbols:

- Save as Binary

Writes data to a file as sequence of binary tokens with the extension « AP3 »

- Save as ASCII

Writes data to a text file as a fully editable object script

- Save Both

Saves both, the text and binary version of the application data successively

All the save options first check if the specified file already exists and ask you to confirm if an overwrite is required.

Please note that whenever you request a load or save operation, the file type will be determined by the specified file extension, i.e. no matter what the contents, if you enter a file name ending in « \*.AP3 » it will be saved as a binary file, even if you selected the « Save ASCII » option. In the case of the « Save Both » option, it doesn't matter what extension you specify, except that the file name specified is the one that is tested for overwrite.

### 3.12 Generating code for object declarations

Whenever you created or deleted exported objects, you should re-generate the object declaration code, i.e. the module that declares and initialises all the constants, types and variables defined and exported by your application.

You can perform this action by clicking the « Generate code » symbol or selecting the corresponding main menu option. You will be prompted for the file name, which will also be used as module name. By default, the file name is set to the first 5 characters of the application name (if already defined), plus the suffix « app ».

Example: If your application is called « Demo », then the default name for the generated module will be « DemoApp.OB2 ». If the application is called « Calendar », the resulting name will be « Calenapp ». For portability, this convention is applied even on platforms that do not have the "8.3" restriction on filenames as DOS.

Please read more about code generation in the chapter "2.23 Code generation".

Note: As most modules in your application will depend on some object exported by this automatically generated code, any change to the exported elements of your application will require a fairly complete recompilation, which can be time consuming.

You *have* to re-generate code and recompile whenever you remove a previously declared object from the application, as otherwise an error will be signalled or your application may exhibit strange behaviour. On the other hand, if you generate code and no change is found as compared to the previous symbol file, this is detected by the automatic Oberon-2 make and no further modules are recompiled. So it may be better to generate code when not really required rather than the other way round.

You do not necessarily have to generate code after adding new objects, as the application will work just fine as long as all previously declared objects are present. The new objects may be found and used by name or ID value.

NB: It is quite possible to use any object loaded from an application file without ever generating any code at all. The generated declarations of objects are just a convenience.

### 3.13 Command line arguments

A3Edit may be started with any of the following options:

- -logo: do not show the logo
- +tx3: load only TX3 files, ignore AP3 files when reading a project tree
- +save: store all projects; if +tx3 was specified, store only to binary AP3
- +genCode: generate code for each project
- +backup: create backup files of TX3 files

These options allow the use of A3Edit as a “compiler”; when either +save or +genCode are specified, the program will quit immediately after saving the requested files.

### 3.14 Importing interface elements

There are several functions to import data from files not in an Amadeus-3 format; These imports are obviously very limited as they only contain a fraction of the information required for a full Amadeus-3 application file, but they usually represent a good starting point to get data formats etc. right.

#### 3.14.1 Import Amadeus-2 TX2 files

It is possible to import an Amadeus-2 file and import data structures and windows, although the windows layout will be quite unusable for Windows applications. At least you’ll get the original windows and the fields they contain.

This function is located in the “Commands” section of the main menu of A3Edit under “Import TX2 Scripts”.

For backward compatibility only for long-time users of the Amadeus frameworks.

#### 3.14.2 Import scroll columns from INI files

The early versions of Amadeus-3 defined scroll columns through INI file entries, under the sections [Header] and [Fmt]. The function “Import INI Columns” interprets those INI file entries in order to generate the corresponding columns in the appropriate windows for the currently loaded application. For backward compatibility only for early adopters of the Amadeus-3 framework.

#### 3.14.3 Import CSV Files

If you have one or more CSV files with column definitions, you may want to create field and database table definitions directly based on those files. There will not be any field type definition, except that fields ending in “Id”(case insensitive) will be considered LONGINT.

The fields are converted to Oberon-2 / Amadeus-3 naming conventions:

- the file name is considered to be record and database table name
- all fields start with lowercase characters
- “-“ and “\_” characters are removed
- tableID names where “table” matches the current record name are converted to “id” and are considered to be the unique id key for the table
- if no project exists, you will be prompted to create a new one
- If there is no database table in the current (or new) project, a new one is created with the name “main”

## 4. Tutorial

So you finally installed the Oberon-2 compiler, maybe the C backend compiler and the complete Amadeus-3 environment. Now you want to get some hands-on experience on how to write an application. You will never even start feeling comfortable with this stuff before you haven't written your little « Hello World » application. Now let's do it and write just such a basic application, so you can see how to work with Amadeus-3 and it's basic environment.

Given the flexibility of an open Framework such as Amadeus-3, as opposed to a closed environment environment such as Visual Basic™, the source code overhead is much larger<sup>11</sup>, i.e. you need a minimum amount of initialisation code for each application. You may use a program template to get started on a new application (which is what we'll do in this tutorial). This process is described in detail in the chapter « 0

---

<sup>11</sup> The object code overhead of Visual Basic, on the other hand, is much larger. In fact, a Visual Basic application requires the presence of a VBX file of over 900KB to run, which is a lot more than the size of most Amadeus-3 executables will ever be - and that's just the overhead.

Starting a new project ». It can significantly speed up development and may be customised to suit your own programming style.

#### **4.1 Making sure the environment is properly set**

First, make sure the complete Amadeus-3 environment is installed and correctly initialized. Particularly important are some environment variables and configuration files. On each platform, these settings vary and may have to be adjusted to match your individual installation. If you followed the instructions in the installation guide of chapter « 9 INSTALLATION », this should work straight away, otherwise read on: remember that the environment configuration is set by a command file which may or may not have been called when you restarted your machine or the development task. If you have any problems running the following commands, please read the installation section again.

In the following chapters, we'll assume that the set-up is standard and that all directories have their default names.

## 4.2 Hello World !

Here is about the shortest Amadeus-3 program that actually does anything “visible”, displaying the famous words « Hello World ! » in a window, the one application you can never escape:

```
<+main*
MODULE HelloWorld;

IMPORT
  Startup,WinMgr,Fields;

TYPE
  Project* = POINTER TO ProjectDesc;
  ProjectDesc* = RECORD (Startup.ProjectDesc) END;

VAR
  project*: Project;

PROCEDURE (p: Project) InitInstance* (): BOOLEAN;
VAR lbl: Fields.Label;
BEGIN
  IF ~p.InitInstance^ () THEN RETURN FALSE END;
  WinMgr.mainW.Show (WinMgr.TopPos);
  NEW (lbl); lbl.Init; lbl.SetName ("Hello World!");
  RETURN WinMgr.mainW.Add (lbl);
END InitInstance;

BEGIN NEW (project); project.Init; Startup.WinStart (project, 0);
END HelloWorld.
```

Admittedly, there is a little more code here than what is needed for some scripting languages or some other environments with a lot of implicit assumptions, but the overhead is not so awful large, is it?

At the very top, there is a pragma command, <+main\*>, which has nothing to do with Oberon-2 code. It’s a statement that is meant for the compiler only and you may not need it for every Oberon-2 compiler out there. In the case of the Excelsior/XDS compiler, this pragma indicates that this is the main module of the application and that some additional code is needed, to make it runnable under Windows. On the original Oberon System by Prof. Wirth, nothing like it was required, as there was no “main” module. Any parameterless procedure could basically be called from outside the module.

The next thing you will notice is the import list.

- We need module Startup to import the application control project class and the WinStart procedure.
- We need the WinMgr module to access the main window of our application.
- We need the Fields module to create a text label object.

Next, you will notice the declaration of an object class, Project, sub-class of Startup.Project. Usually only one object of this class is created, in this case it’s the one we generate with the call to NEW (project). This is the object that will control the execution of your application. You will later learn that this method of execution opens the door to full control over every detail of the process.

Startup.Project objects have a lot of standard behavior programmed in, so you don’t need to do much more than change those bits that are particular to your application. All we need to do is create an appropriate object and pass it on to the Startup.WinStart procedure, which will take care of the rest. The only method we need to instantiate – most of the time, in fact – is the one that is called to initialise the current instance of the application. NB: more than one instance could be running at the same time. Windows allows for a general application-specific initialisation and another one, which is tied to the specific instance of an application.

InitInstance, in our case, starts by calling the super-method it inherits from Startup.Project. If there was an application file with user interface objects, this file would have been loaded already by the super-method. If anything goes wrong during this initialisation, the method call returns FALSE and our program should quit.

After standard initialisation, the main window – which in this case was created automatically – is displayed (WinMgr.mainW.Show) and a new object of type Fields.Label is created. This object class simply displays a one line string in a specific location of a window.

We add the label object to the main window (WinMgr.mainW.Add) and leave the initialisation method. The entire rest of the execution – displaying and refreshing the object on-screen, handling all kinds of window events and terminating the application when Alt-F4 is pressed (or the system menu function Close is selected) – is handled completely by Amadeus-3 standard library modules !

But isn't this a bit complicated, just to display one little string? Isn't there some "PRINT" , "WRITE" or "DISPLAY" procedure, that will do the job?

Well, under Windows or any other GUI, you don't just « write » text or « draw » pictures to the screen. There are indeed such functions, but as soon as the area that you changed with such primitive graphics functions needs to be updated, the information that you placed there disappears. After all, it's just been painted to the screen, not saved. Therefore, we need a way to remember what should be displayed in a given area of a window. Then the display object can redraw the corresponding information (itself) any time a change occurs in it's own area of the screen, when it becomes visible.

Here is some more information about Fields.Label and the use of display objects:

- They are all derived from the basic (abstract) object class `WinMgr.Object` .

We create an instance of a particular display object class, in this case `Fields.Label`. By declaring the variable `lbl` as being of class `Fields.Label` and applying `NEW` to it, we have created an *instance of a text label*.

The method call `lbl.Init` makes sure that all fields of the object are *properly initialised*. This is the standard « constructor » of all objects based on the class `Persist.Object`.

The method call `lbl.SetName` defines the string that is to be used for our display. `SetName` is also a basic method of all objects of class `Persist.Object`, but in the case of text labels, it does not define the symbolic object name but the actual display string, which you will find when reading the definition of module `Fields`.

Now all that remains to be done is to add this label to the main window of our application.

- All applications have at least a main window, which is always pointed to by the variable `WinMgr.mainW` and which is identified by the name "MAIN" (reserved name).
- If using an object resource script, you may override the default definition of this window. In this way, you can set the window title, main menu and the background colour. We will get to this later.

As we mentioned before, the main window has already been created with default settings before we defined our label. We explicitly move it to the display. So we don't need to do anything more than call the method `WinMgr.mainW.Add (lbl)` to add the text label to the window. We are using an implicit assumption here, i.e. we expect the default co-ordinates of our label to be (0,0). The act of adding a new object to a window causes the affected area of that window to be refreshed, which in turn causes the display object `lbl` to be displayed. The actual dimensions of the object are calculated automatically when it is displayed for the first time.

The result is what you see when you run this application. For this, we must first compile and link it, to create an executable program.

### 4.3 Compiling the “Hello World” application

The compilation process obviously depends on your operating system, your environment and the compiler you chose. For now, we’ll assume that you are working with a command line interface. As you will find in your compiler document, you may need additional files to compile an application, especially the information about where to find source code, symbol and object files etc.

In the Tutorial directory HelloWorld you will find a project file for use with the XDS compiler, HelloWorld.PRJ.

Amadeus-3 supplies a few basic command files for each supported platform that simplify the common tasks. If your environment was properly configured according to the instructions in chapter « 8 INSTALLATION », then you only have to type the following command to compile and link the demo application:

```
Xc =pr HelloWorld.prj [Enter]
```

For more information on the compilation process, please refer to chapter « 10.1 Compiling and linking ». Of course you may use whatever standard GUI development environment (IDE – Integrated Development Environment) is supplied with your compiler or you may integrate the corresponding calls into your code editor, if it does support external calls. A good example is Multi-Edit.

### 4.4 Running the application

When the application has been properly compiled and linked, you can run it in your GUI environment. The executable name should be the same as the main module’s, according to the platform with or without a special extension such as ‘.EXE’.

You have to ensure is that the application can actually find any required auxiliary files. This can be any of the following: .INI files, .AP3 (or .TX3) files, .DLL files for external libraries etc. A complete discussion of all standard files required for running Amadeus-3 applications can be found in « 5.3 Files to be distributed with end-user applications ».

For the application to find these additional files usually requires that either the application is started from it’s own directory where all the other files are found (cf. system settings on your platform) or that the application directory is specified in the path variable (for DLLs).

- **For now, make sure that you start the demo application from it’s own directory**

At this point, running the application will give you the following display (under Window):

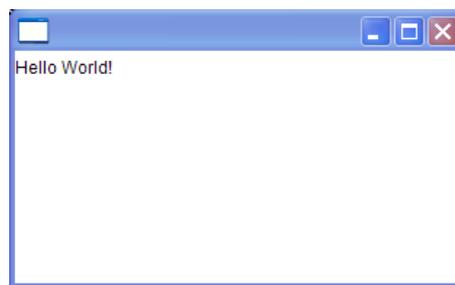


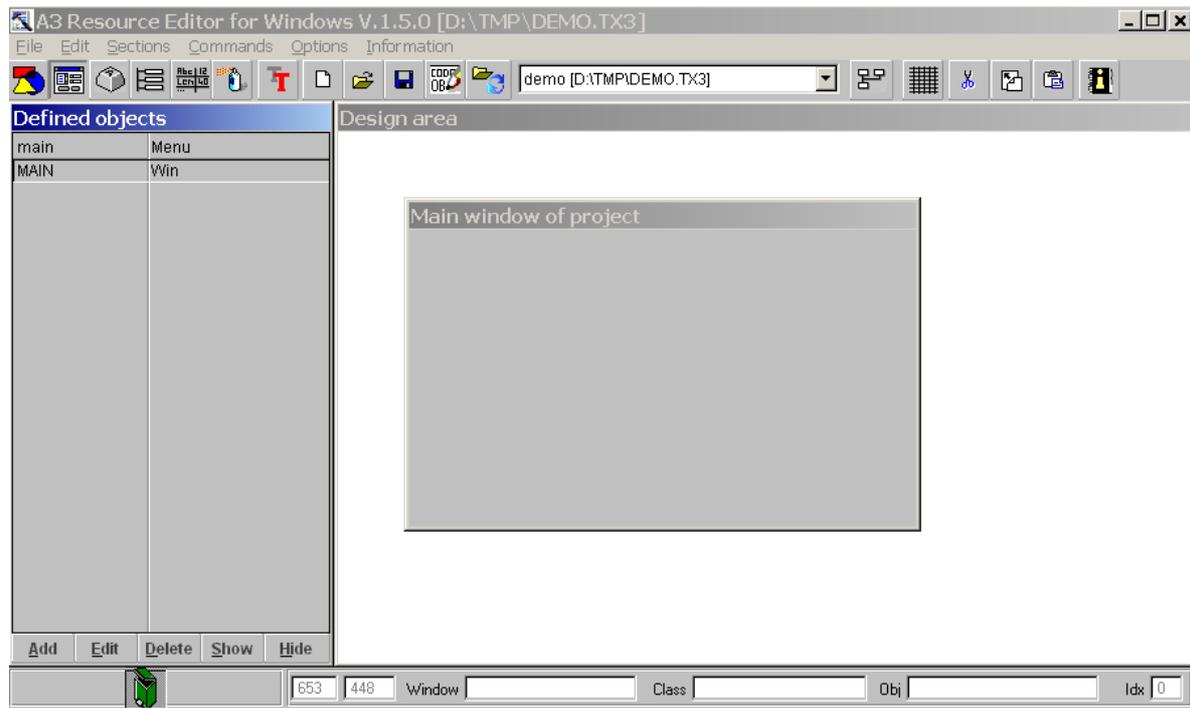
Figure 56 - "Hello World!" Demo program

### Using the application editor A3Edit

Now let's start enhancing our application visually. This is best done with the application editor A3Edit. So start it up and load the Tutorial.TX3 file that you will find in the A3.DOC\Tutorial directory.

**HINT:** As this is an operation that you will repeat often while working on a project, you may want to set up an icon or other command shortcut to load directly the A3Edit program with the application file you wish to use. To allow this, A3Edit accepts the name of a .TX3 or .AP3 file as command line argument.

After loading the Tutorial application, double-click the line "MAIN / Win", to get the following display:



**Figure 57 - A3Edit with initial Tutorial data**

Then double-clicking the first line in the « Defined Objects » list. This will cause the menu editor to pop up over the design area. Through this editor, you can view and edit all the options that you wish to make available through a menu. Menus can be either window menus or popup menus. Close the menu editor with the "Close" button.

Let's turn our attention to the "MAIN" window again, which you can manipulate in various ways. You can resize it by dragging on a corner, move it, add display objects, delete them, change the basic window parameters or perform any of whole range of other operations, as you will see.

## Creating a window

- Click on the icon with the « Object folder » symbol 
- Click the [Add] button at the bottom of the object list.

*You will be prompted for the type of object you wish to create and the name of the object:*

- Set the object name to « buttonTest » and the object class to « Window ».
- Press the [Accept] button

*The full window definition mask will pop up.*

- Set the « Title » field to « Button test ».
- Leave all other options as set by default.
- Press the [Accept] button

*Now you should see the new object « buttonTest » appear in the object list on the left*

- Double-click this object in the list
- The newly created window appears in the « Design » area.*

## Setting windows parameters

To access a window's parameters, either:

- Select it in the object list and press the “Edit” button
- or

- Display the window and double-click the right mouse button within the window area, then select “Settings” (or if you clicked on top of a window object, select “Window settings”)

The option “Set Help / Note” opens a window that allows you to specify different strings:

- Help topic; this topic will be used to search the help file when the user asks for context-specific help, for example by pressing the F1 key.
- Bubble info is the text that will be shown in a help bubble window when the mouse hovers over the object or window for a certain amount of time
- Comment for documentation purpose only; you may use this section to specify any information that is important during the development process; it will appear in the TX3 file, so you may refer to it while writing the application source code

## Adding simple objects to window

**Next we'll add 2 buttons to this window**

- Click on the icon with the « Command button » symbol 

*A list of default command will be displayed.*

- Drag&drop the « Ok » command to the « buttonTest » window

*As soon as you start dragging, a button with the label « Ok » will appear as dragged object.*

- Drop this button some place within the « buttonTest » window.

*The button should appear where you dropped it, now belonging to the « buttonTest » window.*

You may now move and resize the button with the usual mouse actions. You may also change the button settings:

- Double-click the « Ok » button with the left mouse button

*The button settings dialog mask appears.*

- Check the « Default » box.

*The « Accessible » box should be selected by default.*

- Press the [Accept] button in the settings dialog.

Repeat this procedure with the « Cancel » command to get used to the operation sequence.

**Then add a text label**

- Double-click the left mouse button in an empty area of the target window.

*A popup menu appears.*

- Select « Add Label »

*The label settings dialog appears.*

- Enter the text « This is a string » into the « Object title » field.
- Press the [Accept] button

*The specified string appears in the « buttonTest » window at the place where you clicked the mouse initially.*

Your window should have an appearance similar to this, by now:



**Figure 58 - Sample window "Button Test"**

You may also align the buttons, so they look a bit more professional. When you move and resize objects, you will notice that there is a constraint grid. Movement and resize increments are not pixels. This is determined by the « Alignment grid increments » parameters in the windows settings dialog (accessible by double-clicking the right mouse button over an empty area of the target window). When you wish to align an object more precisely, press the "Shift" key while performing the mouse operation. This allows you to move the object without reference to the alignment grid.

### **Object Parameter Setting**

As you've seen, you can access the object's parameters and settings by double-clicking with the left mouse button. This works almost everywhere. In a few instances, this may not be possible, because of possible double meaning of the mouse action. In those circumstances, you may access object settings by right-clicking the object and then selecting "Settings" from the popup menu.

### **Object Context Menu**

By right-clicking an object, you will get an object-specific menu, which contains various functions for manipulating the object. You may also access the owner window's parameters.

The object context menus also offer the option "Set Help / Note", with the same meaning as for windows.

### **Creating a data entry mask and adding fields to it**

A data entry mask may be a simple WinMgr.Window or a often a Scroll.Mask. The latter have the particularity that they may be associated with a scrolling window. In this case, they have a strong relationship and in many operations, such as adding and modifying elements in the scroll list, the mask will be used as default input device.

NB: Any window class that does not explicitly prohibit it should accept data entry fields, buttons and other elements used in data entry and dialogs.

Now create a data entry mask by the same process as for « buttonTest », but make it of class « Scroll Mask » and give it the name « dataEntry » and the title « Data Entry Mask » (or anything else that suits you).

Before we can add data entry fields, we need to create the variables that will be linked to these fields. **Amadeus-3** is data-centric, so the variable comes before the field. A field without some object linked to it would have no meaning.

- Activate the data dictionary list by clicking the « Data Dictionary » icon 
- Press the « Add Variable » button

*The variable definition dialog pops up.*

- Enter the variable name as « string1 ».
- Enter « 20 » in the length field.
- Press the [Accept] button

*The newly created variable is displayed in the dictionary list.*

- Drag&drop this variable to the « dataEntry » mask (*reminder: using the right mouse button*)

As soon as the variable is dropped inside the mask, it is transformed into a data entry field with the default label set to the name of the associated variable.

To change the field label:

- Double-click the data entry field.

*The field settings dialog appears.*

- Set the « Field title » to « First string ».

Repeat this process by defining other variables of various data types and dragging them to the « dataEntry » mask.

**HINT:** It is quite possible to add the same variable more than once to the same window. They will be synchronised automatically. This may not seem very useful, but in a minute you will learn that in fact the same variable may have different representations.

Finally, add the « Ok » and « Cancel » button to this mask, too. To do this the **Amadeus-3** way, copy both buttons from the « buttonTest » window, which should still be open (cf. below, multiple windows operations):

- Mark both buttons by « rubber-banding » them or by clicking them while pressing the « Ctrl » key.
- Drag&drop them to the other window, while pressing the « Ctrl » key, to perform a copy, not a move operation.

Note that while performing copy operation, you actually see the full object group moving with the mouse, not just an outline. This is the special « **Amadeus-3 drag & drop** » !

The copied objects retain most settings from the original object, except where such a setting would not make sense. The « Ok » button, for example, will retain the « Default » attribute.

### Adding object from the class list

Certain objects may be created directly from the class list. All display object classes may basically just be dragged onto a window, to create an object of that class. If the class is not known to A3Edit, then it simply displays it as a blank square, which you may size and place within the window. At load time, undefined objects will be replaced with objects of the proper class, IF the class is available.

Non-display objects will not be accepted in the same way. Some may still be used in Drag & Drop operations. This is especially true of data sources (derived from Sequence.Source). Any data source may be dragged to a scroll window, which then automatically is assigned a data source of the same type. From there on, you may set data source specific parameters by right-clicking the scroll window contents.

Various scroll sources are known to A3Edit and can therefore be parametrize. This is especially true for:

- ListView: you may associate it directly with a memory list you created in A3Edit

- DbView: you may set the Db file it is based on, the key to use, filter conditions and related files to be saved during access operations etc.

More on this a bit later, in the chapter “Creating a scroll window based on a database table”.

### Special variable types

Boolean variables are not displayed as normal data entry fields. When you drop a boolean variable on a window, it appears as a checkbox.

Several checkboxes may also be grouped together, to create multiple-choice sets with mutually or setwise exclusive options. This is achieved by doing this:

Create a boolean variable of name « multiChoice » with array size 5.

Drag this variable to the « dataEntry » mask to create a checkbox.

Select the « Objects » list 

Press the « Add » button.

Create an object of class « Button group » and name it « multiChoice ».

In the button group settings dialog, set the « Group style » to « Exclusive set ».

- Double-click the checkbox.
- In the checkbox settings dialog, pull down the « Owner » droplist and select « multiChoice ».
- Press the [Accept] button.

*The checkbox has changed to a « radio button ».*

Now make further instances of this toggle, to give access to all possible 5 settings:

- Press the [Ctrl] while you drag&drop this toggle.
- Drop the copy onto the same screen.
- Double-click the copy to see the settings.

*All items are unchanged, except for the array index, which has been incremented by 1 (the lowest index being 0).*

- Repeat the copy process 3 more times.

*Now you should have 5 radio buttons which all point to 1 of the 5 array elements of our variable.*

**NB:** The copy operation does not change the toggle title setting. You have to do this manually by editing each object’s settings. In this case, let’s add a single digit to the end of each toggle field title.

### Transforming data entry fields

A variable may also have various other representations:

- Numeric variables may be represented as Scrollbars or multiple-choice « radio buttons » or - in Amadeus-3 terminology - « Toggles ».
- Data entry fields may have dropdown lists attached

To change the field type:

- Double-click the **right** mouse button on top of the field.

*A popup menu appears.*

- Select one of the available « Make ... » options.

To try this with the « Make Scrollbar » option:

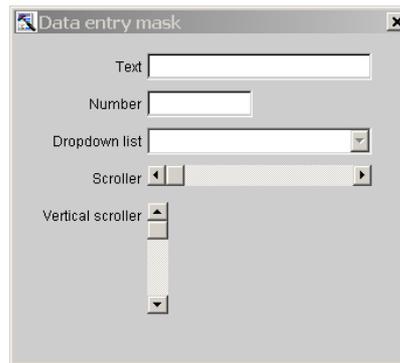
- Create a numeric variable (ShortInt, Integer or LongInt only) and name it « scroller ».
- Drag&drop it on the data entry mask, twice.
- On the second field, double-click the right mouse button to display the object menu.
- Select the « Make Scrollbar » option.

The field is changed into a scrollbar.

When a numeric field is displayed as a scrollbar, then any change in the position of the scrollbar box changes the value of the associated variable is set corresponding to a value chosen proportionately between the minimum and maximum values as specified in the scrollbar settings mask. If the numeric variable is also linked to a normal data entry field, this field will immediately be synchronised with the scroller position.

**For the fun of it:** you may also try placing several scrollers and several fields in the same mask, all linked to the same variable. When we get to compiling and running our application, you will be able to experiment with this.

By this time, your window will look something like this:



**Figure 59 - "Data entry Mask"**

Both windows were included, to attract your attention to the fact that Amadeus-3 allows you to simultaneously view and manipulate multiple windows. This goes even much further, as we shall see now. Windows are treated just like other object (or almost...).

## Manipulating windows

Window manipulation operations are greatly restricted: All the simple mouse actions are already exhausted for manipulating objects inside the window, e.g. left button click-hold-drag is used to « rubber-band » objects within the window. Right button click-hold-drag performs drag&drop on the window itself. So we need the « Ctrl » key to signal that we want to move/resize the window instead (except where a title bar and thick frame are available, which allow these manipulations through the GUI's mechanism).

- Click and hold the left mouse button while pressing the « Ctrl » key.
- While the mouse button is pressed, you may release the « Ctrl » key again.
- Now move the mouse to move the window.

Of course, if your window has a title bar, you may move it the standard way, too.

**Attention:** You may select multiple windows in the same way as you select multiple display objects, i.e. by « rubber-banding » them (using the design area as activation zone). When several windows are selected (signalled by the grey handle boxes at the corners of the window), they will all move together, although in this case only an outline will be moved. When many windows are open, you may not even see the outline moving, as it is drawn on the design area. This may be fixed in a future release, but it requires a low-level programming approach.

To resize a window, first select it, again by clicking it with the « Ctrl » key pressed. When the handle-bars are turned on, you may resize the window by pulling on it's border, just as with any display object, no matter what the frame type is<sup>12</sup>.

<sup>12</sup> Windows only allows resize operations on windows with thick borders and move operations on windows with a title bar.

If your window has a thick frame, you may also resize it the standard way. In this case, the area of resize-sensitivity is slightly further out than the standard frame activation zone.

You may also copy windows the same way you copy display objects: by drag&drop, while pressing the « Ctrl » key. The newly created window will be empty and the name will be changed by adding an automatically incremented number for proper distinction of the two objects. The copied window will be empty. You can copy all display objects of the original in the standard way.

### Child windows

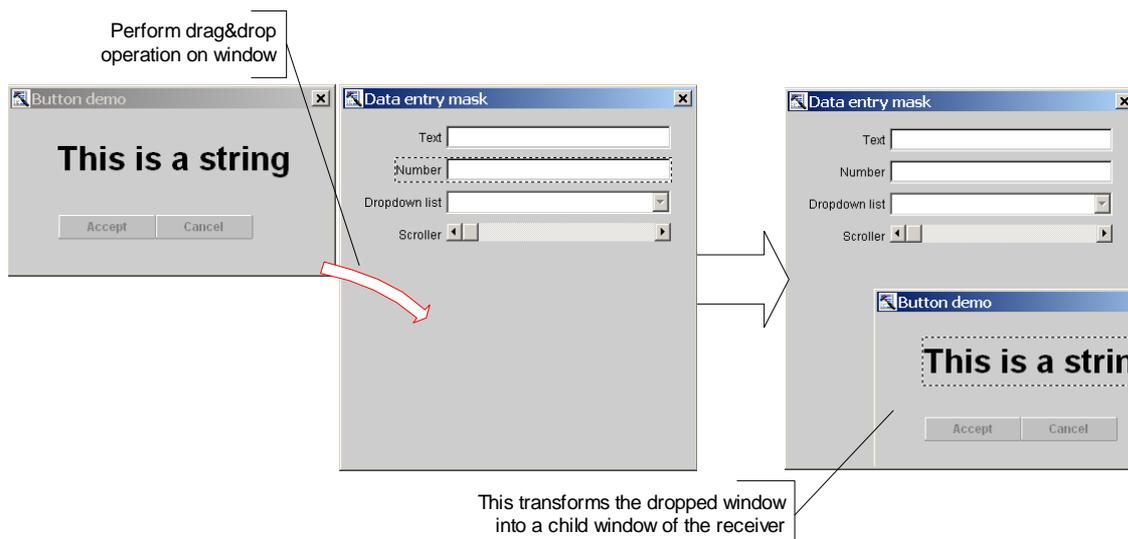
There is one more special feature concerning windows. A window may be free-floating, owned by no one. Or it may have a parent window (specified in the window settings dialog). When the parent window is specified, this simply means that *if* or *when* the window is displayed, it will be placed in front of it's parent window. It is not necessarily shown when the parent window is displayed.

A true child window, on the other hand, is basically just a special type of display object, i.e. it is really tied to it's parent window. To create such a window, do the following:

- Grab the window with the right mouse button, as usual for drag&drop operations.

**Surprise:** the entire window becomes an object that you may drop anywhere, including on the trash can. *This is one of the definitely unique Amadeus-3 features!*

If you drop a window onto another window, the dropped window becomes a child window of the receiving window. It will disappear from the object list, as it is now integrated into the hierarchy of the parent window. It may still have it's own declaration in a the generated Oberon-2 declarations, but it is definitely bound to the parent window.



**Figure 60 – Window Drag & Drop**

You may render it independent again by dropping it onto the design area.

Child windows may have smart « attachment » constraints, that are similar to Motif attachments. They allow auto-sizing of child windows of a same parent window. Very powerful, but a bit advanced for this tutorial.

## Using objects created with A3Edit

The next question is how to use all the objects created under A3Edit. The answer is: it depends. You can load objects generated by A3Edit in any Amadeus-3 application, without knowing much about them. A higher-level system, such as an interactive database manager, where you create and view data, would be very easy to write. Parts of A3Edit could be integrated as "Screen and database designer", as any Amadeus-3 application can handle dynamically created objects by using pointers and object names.

For programs with a more limited scope, such as specific end-user applications, there is a much easier way to gain access to A3Edit-created objects. You can use the code generation feature to create Oberon-2 declarations for all objects that need to be referenced in your code. Please note that only declarations and a simple object-initialisation procedure are generated, no actual application code. All functionality will be provided through your own Oberon-2 code, in which you are encouraged to make extensive use of the Amadeus-3 library and code from existing applications. This system may require a little more effort at the outset, but as you will soon realise, it confers much greater control and flexibility. It also avoids all the pitfalls of automatically generated functions, such as the need to distinguish between generated code and user code etc. etc.

Code generation is accessed through the  icon. Try it now on your Tutorial application.

=> The result will be an Oberon-2 source code module named (by default) TutorialApp.ob2. This name is generated based on the application name, appending the suffix "App". You can change this default name, but let's assume for now that you keep it as « Tutorial ».

Have a look at the declarations that were generated. The initialisation procedure assigns dynamic objects to the matching variables by name. This procedure must obviously be called only after the actual objects were loaded, which is done through the PStore.LoadProject method, which looks for and loads the application Tutorial.AP3 or - if not available - Tutorial.TX3 or shows a warning message and quits the application if not found, cf. the code in the InitInstance method below.

Here is the generated code as found in « TutorialApp.OB2 »:

```
MODULE TutorialApp;
IMPORT
  StrVal, NumVal, Persist, Projects, Dialogs, Values, SYSTEM, Menus, Colours, Fonts, WinMgr, Fields, DropList, Scroller, Buttons;

CONST
  CmdBase* = 1000;

VAR
  thisProject*: Projects.Project;
  mainMenu*: Menus.Menu;
  demoBrush*: Colours.Brush;
  demoFnt*: Fonts.Font;
  dataEntryWin*: WinMgr.Window;
  buttonTestWin*: WinMgr.Window;
  test*: ARRAY 50 OF CHAR;
  number*: INTEGER;

  __ok:BOOLEAN; __po:Persist.Object;

PROCEDURE Find(name:ARRAY OF CHAR;id:INTEGER);
VAR ow:Persist.Object;
BEGIN
  IF ~thisProject.Find(name,id,ow,__po) THEN Dialogs.Error ("NotFound", name);
  __ok := FALSE END;
END Find;
PROCEDURE Asgn(name:ARRAY OF CHAR;VAR data:ARRAY OF SYSTEM.BYTE);
BEGIN Values.AssignByName(thisProject.dict,name,SYSTEM.ADR(data),__ok) END Asgn;
PROCEDURE Init*(prjAsParam:Projects.Project):BOOLEAN;
BEGIN
  __ok:=TRUE; thisProject:=prjAsParam;
```

```

Find("main",Menus.menuId); IF __ok THEN mainMenu:=__po(Menus.Menu) END;
Find("demo",Colours.brushId); IF __ok THEN demoBrush:=__po(Colours.Brush) END;
Find("demo",Fonts.fontId); IF __ok THEN demoFnt:=__po(Fonts.Font) END;
Find("dataEntry",WinMgr.windowId);
IF __ok THEN dataEntryWin:=__po(WinMgr.Window) END;
Find("buttonTest",WinMgr.windowId);
IF __ok THEN buttonTestWin:=__po(WinMgr.Window) END;
Asgn("test",test);
Asgn("number",number);
RETURN __ok;
END Init;

END TutorialApp.

```

Now let's use the generated objects in our Tutorial application:

```

<+main*>
MODULE Tutorial;

IMPORT
  Startup,PStore,WinMgr,Fields,Commands,Events,Keys,App:=TutorialApp;
  (* App will be an alias for TutorialApp *)

TYPE
  Project* = POINTER TO ProjectDesc;
  ProjectDesc* = RECORD (Startup.ProjectDesc) END;

VAR
  project*: Project;

PROCEDURE (p: Project) InitInstance* (): BOOLEAN;
BEGIN
  IF ~PStore.LoadProject (p.name^, App.thisProject, App.Init) OR
    ~p.InitInstance^ ()
  THEN RETURN FALSE END;
  WinMgr.mainW.Show (WinMgr.SamePos);
  App.buttonTestWin.Show (WinMgr.SamePos);
  App.dataEntryWin.Reset (WinMgr.ToDefault);
  RETURN App.dataEntryWin.AcquireFocus (TRUE);
END InitInstance;

BEGIN
  NEW (project); project.Init; project.SetName ("Tutorial"); (* create project *)
  Startup.WinStart (project); (* Start windows interface *)
END Tutorial.

```

As you see, the A3Edit generated objects may now be manipulated just as any other object, but their settings are imported from the application file.

## Handling Events: Command codes - Menus, Buttons etc.

It is time now to look at how to handle the interaction with the user. We have windows and data entry fields, buttons and menu options. But how does the application program know when the user selects a menu option or activates a button? How can it retrieve data that the user entered into a field?

The basis for all this is summed up by one word: « Events ».

Whenever the user performs an action, such as pressing a button, he generates an event, which traverses the full event handling sequence of any **Amadeus-3** application program. The details of this process are outlined in chapter « 2.8 Message handling ». For now, let's concentrate on the simple things.

The demo program in its current version starts by setting the input focus to the window `dataEntry`. The user is now free to enter data into the various fields found in this mask. He may also press the keys « Ok » or « Cancel ». The program will get specific event messages whenever one of the buttons is pressed. To act on such an event, the application program first must be able to intercept the event message. For this, it must install an event handler procedure or method. As we did not declare any new object class, we'll use a dynamically assigned procedure.

```
IMPORT (* existing import list .. *),Events,Commands;

(* ... *)

PROCEDURE HandleDataEntry (w: WinMgr.Window; VAR ev: Events.Event);
BEGIN
  IF ev.tp = Events.Command THEN
    IF ev.code = Commands.Ok THEN
      (* save data *)
    ELSIF ev.code = Commands.Cancel THEN
      (* discard data *)
    ELSE
      RETURN; (* command code unknown in this procedure *)
    END;
    ev.done := TRUE; (* signal that command was handled *)
  END; (* if ev.tp *)
END HandleDataEntry;

PROCEDURE (p: Project) InitInstance* (): BOOLEAN;
  (* ... *)
  App.dataEntryWin.handler := HandleDataEntry;
  (* ... *)
END InitInstance;
```

## Handling Events: Keyboard input

Contrary to non-GUI programming, your program will only very rarely have to handle keyboard input directly. Most objects that are text/data related will take care of reading the keyboard by themselves. But occasionally, you may wish to react to some specific keyboard input, such as when the user presses “Enter” or “Esc” or a function key etc.

You may check the actual keyboard action in `ev.action` (Press, Release), the key code in `ev.key` and the key modifiers that apply in `ev.state` (Shift, Control). The character value assigned to a particular key may be found in `ev.chKey`. Usually, you should check for `ev.action = Events.Press`. This action may be repeated if the user keeps pressing the key. The release action will be sent only once.

```
IMPORT (* default import list .. *),Keys;
(* ... *)
PROCEDURE HandleDataEntry (w: WinMgr.Window; VAR ev: Events.Event);
BEGIN
  IF ev.tp = Events.Command THEN
    (* ... *)
  ELSIF (ev.tp = Events.Key) & (ev.action = Events.Press) THEN
    IF ev.key = Keys.Esc THEN
      (* discard data *)
      ev.done := TRUE; (* signal that event was handled *)
    END; (* if ev.key *)
  END; (* if ev.tp *)
END HandleDataEntry;
```

## Handling Events: Mouse input

Mouse events occur frequently and come along with coordinates as well as keyboard modifiers, just like keyboard messages. The latter may be found in `ev.state`, while `ev.action` contains a code describing the actual mouse activity, i.e. `Move`, `Press`, `Release`, `DblClick` (for Double Click) etc.

Any button-related command (`Press`, `Release`, `DblClick`) will also set the `ev.button` field to `Lbutton`, `Rbutton` or `Mbutton` (for Left Button, Right Button or Middle Button).

Coordinates are returned in `(ev.px, ev.py)` and `(ev.sx, ev.sy)`. `px` and `py` are relative to the receiving window, adjusted for the window’s current scrolling offset. `sx` and `sy` are the original coordinates as sent by Windows.

Let’s add a little piece of code to our handler procedure, that will display the click coordinates for any double-click event anywhere within the dateEntry window:

```
IMPORT (* default import list .. *),Str,Convert,Dialogs;
(* ... *)
PROCEDURE HandleDataEntry (w: WinMgr.Window; VAR ev: Events.Event);
VAR s,t: Str.ShortStr;
BEGIN
  (* ... *)
  IF ev.tp = Events.Command THEN
    (* ... *)
  ELSIF (ev.tp = Events.Key) & (ev.action = Events.Press) THEN
    (* ... *)
  ELSIF (ev.tp = Events.Mouse) & (ev.action = Events.DblClick) THEN
    Convert.IntToStr (ev.px, 1, TRUE, s); (* Convert integer to string *)
    Convert.IntToStr (ev.py, 1, TRUE, t);
    Str.AppendC (s, ', '); (* Append comma and space *)
    Str.Append (s, t); (* Concatenate values *)
    Dialogs.Message ("Click", s); (* Display in message box *)
    ev.done := TRUE; (* Event is handled *)
  END; (* if ev.tp *)
```

```
    (* ... *)  
    END; (* if not done *)  
END HandleDataEntry;
```

## Handling Data Entry

Your application needs to handle the above messages only when it wishes to achieve some specific result, not for normal user input. Most application-defined message handlers will only have to treat command messages. Keyboard and mouse messages will be taken care of by the data entry fields, buttons etc. that you placed into your windows.

Data that the user enters into any application fields will be directly transferred to the variable attached to the field,

The field and the attached data could be dynamically assigned, but will typically have been created and assigned under A3Edit, so that you may access it directly via the corresponding application module variable generated by A3Edit.

If we use the above example, let's say you placed the field "date" in the window "dataEntry". Your application displays the window it using the Window.Show method. At some point, the handler procedure (cf. the section above on Command codes, the procedure HandleDataEntry) receives the message Events.Command/Commands.Ok. You don't need to perform any further operation to retrieve the data that was entered by the user. You will know that whatever data he entered into the field "date" will be found in the variable "TemplateApp.date", as soon as it was entered by the user.

You can of course handle data as soon as the user enters or leaves a field, by attaching specific handler procedures to individual fields, called Access procedures, cf. module Fields. This allows you to respond to any new data before the user even selects some handling procedure by pressing a button etc.

## Handling system tray notification icons

Your application may want to signal various events via the system tray icons, e.g. when new information is waiting for the user. You can do this via the `Dialogs.Notify` function. You need to provide the required icons in your resource file, then proceed as follows:

- Call `Dialogs.Notify` and specify the icon name, the icon ID and the window you'd like to receive mouse messages from the notification icon, the message to be sent and the tool tip name.

Example:

```
Dialogs.Notify (NotifyOrder, "NewOrder", WinMgr.mainW.hWnd, Events.Notify, "");
```

The tool tip may be an actual string or the reference of a resource entry in the `Msg` (message) section of the application resource file. If the tool tip is not specified, the icon name is used instead.

To handle icon events, add some handler code to the receiver window, e.g.:

```
PROCEDURE Handler (w: WinMgr.Window; VAR ev: Events.Event);
BEGIN
  IF ev.tp = Events.Notify THEN -- user event, really always a mouse message
    Dialogs.NotifyMsg (ev); -- Translate the message
  IF ev.action = Events.Release THEN
    Dialogs.RemoveNotify (ev.key); -- remove the icon from the tray
    ev.done := WinMgr.mainW.AcquireFocus (TRUE);
    IF ev.key = NotifyOrder THEN WinTools.ShiftFocus (App.pendingWin) END;
  END; (* if *)
  ev.done := TRUE;
END; (* if ev.tp *)
END Handler;
```

## Creating a database

Based on the data dictionary we've created, we may now proceed to implement a database:

First, let's create a record that we will use with the database:

Go to the data dictionary and press "Add record", name it "customer" and press "OK".

Open this record (press "Enter" on the name in the list) and press "Add" to insert fields:

- "id" as LongInt
- "name" as Text with length 40
- "first" as Text with length 20
- "address" as Text with length 200; go to the "String" Tab and set "max. nb. of lines" to 3

Sort these fields by dragging the lines so that "id" is first, followed by name, first and finally address.

Go to the object list and press the "Add" button.

Enter the name "main" and select "Database" as object type to be created.

Open the database definition by pressing "Enter".

Specify the default path "Data".

From the "Available fields" window (may be hidden behind database window), select the "customer" line and drag it to the "File list" window by clicking and holding the right mouse button. This will create a line "customer" as new database table.

Open the "customer" table by double-left-clicking it. Click in the "Keys" window and then press the "Insert" key (it may be labelled "INS" or "Insert", in French "Insertion", in German "Einfuegen" etc.).

Now pick the field "customer.id" from the "Available fields" list and drag it into the "Segments" window. Double-click it and check the "Autoincremented" option, then press "Enter" to confirm. Press "Enter" again to close the Key definition window.

Now press the “Insert” button again to add another key. This time, drag&drop the “customer.name” key to the segment list. Double-click the segment and specify that it is a modifiable and duplicate key with .

### Creating windows based on Templates

Use the  button to access the create the creation of windows with the template function. Select the “Standard scroll list” window, enter the name “customer” and press “OK”. This will generate the two top-level windows “customerTop” and “customerEdit” in the object list. You may then edit these windows to change their appearance or other attributes.

After insertion, make sure that the MAIN window precedes the newly added windows or sort them accordingly by dragging.

You can change and add templates by editing the Template.tx3 file in the A3Edit directory. NB: Make sure you save your templates before installing a new version of A3Edit!

### Creating variables based on Templates

When adding variables, you may also use templates, in particular for standard record structures which you use often. Simply press the “Dictionary template” button after entering the name of the new dictionary object to be created and choose the desired template.

As for window templates, the dictionary templates are stored in the A3Edit\Template.tx3 file. By editing the dictionary of that application, you can add more dictionary templates.

### Creating a scroll window based on a database table

We already have the window we want to use for scrolling access to the database. It is “customerList”, a child window of “customerTop”. We just need to transform it so that it can be used to access the customer database file.

First, display the customerTop window. Then click on the “Class” button .

Search for the class “DB source”. Drag & drop this class on the scroll window “customerList”.

Then double-right-click the mouse in this window and select “Source settings” from the popup menu.

As source name, specify “customerSrc” (this is optional for now, but it’s a good practice to name all relevant objects). Select the “customer” file and the “name” key, so that customers will appear sorted by name. Set “use filter” to off. Accept these specifications by pressing “Enter”.

Finally, we need to define what contents should be displayed in this window. The first element are the scroll columns. Press “Alt-V” (or press the button for the data dictionary) and open the customer record (should actually still be open).

Drag&drop the “name” field to the “customerList” window. A scroll column will automatically be defined.

Drag&drop the “first” field to the “customerList” window.

Drag&drop the “address” field to the “customerList” window.

Now we may want to add column titles. Open the “Command” list (press the corresponding button or “Alt-C”). Then press the “Add” button and specify “Dummy” / “100” and accept with “Enter”.

Then D&D the resulting command to the “customerList” window over the first column. You will be prompted with the question “Do you want to add a column title?”. Answer “Yes” and you will see a button at the top of the “name” column. You can now change the title and other attributes of this button by double-clicking it.

Repeat for the other two columns.

At this point, your scroll window is complete.

Save the application (check the “Generate code” option). We are now ready to write the necessary application code.

## 5. Elements of AMADEUS-3 APPLICATIONS

All Amadeus-3 applications have a few things in common, starting with a certain directory organisation and required files for development, code elements and finally files to be distributed with the application.

You will find that the overhead of Oberon-2 code and the Amadeus-3 framework do not make it suitable for tiny applications as you may write with Visual Basic™ or other such tools. You only start reaping the real benefits of object-oriented programming when your application grows, which is true for all comparable frameworks, such as Delphi™ or any C++ library. So don't compare Oberon-2 with Amadeus-3 to a simple application generator. Amadeus-3 based development will start paying off when data structures become more complex and your source code goes beyond a thousand lines.

First of all, it's important to keep your development environment organised and consistent. Precisely the limitless possibilities of how to write and build an application, it is better to find a system one is comfortable with and then to deviate as little as possible for as long as possible in each new application. If you always recognise a consistent organisation and a certain number of files, which follow a consistent naming convention, it will be much easier to maintain many different applications, mostly when you did not work on some applications for a long time.

### 5.1 Directory organisation

It is convenient to stick with a certain directory organisation, making it easier to maintain your source code and other development-related files. Many utilities and batch files will be easier to write if they can make certain assumptions about your directory organisation. If we name the application « Demo », an example organisation would be the following:

Your application directory

**Demo\**

*Demo.OB2, Demo.INI, Demo.AP3, Demo.TX3, Demo.RC, Demo.MAP,  
Demo.ICO, DemoApp.OB2,...*

*other source code and application-specific files*

**BITMAPS\**

*Bitmap and icon files used by the application*

**OBJ\**

*Compiler generated object and symbol files for application-specific modules,  
compiled RC (RES) files*

The complete library with source code and other Amadeus-3 specific files

**A3\**

*Amadeus-3 library modules*

**OBJ\**

**A3Lib\**

*Platform-specific library and runtime support files, DLLs (for Windows, OS/2) etc.*

**A3.Doc\**

*Amadeus-3 documentation files*

The Amadeus-3 application editor

**A3Edit\** *A3Edit.EXE, A3Edit.AP3, A3Edit.INI*

*plus source code etc. for the full development version*

XDS Oberon-2 compiler installation

**XDS\**

**BIN\**

	<i>Compiler executables, command files, configuration files etc.</i>
<b>DEF\</b>	<i>Sub-directories with library definition files</i>
<b>WIN32\</b>	<i>Windows-specific definition files</i>
<b>SYM\</b>	<i>Library symbol files</i>
<b>LIB\</b>	<i>Sub-directories with platform-specific library files</i>

#### Extacy Oberon-2 compiler installation

<b>Extacy\</b>	
<b>OS2\</b>	<i>OS/2 compiler and other executables</i>
<b>BIN\</b>	<i>Compiler executables, command files, configuration files etc.</i>
<b>RTS\</b>	<i>Windows 3.x platform-specific library and runtime files; should be on PATH</i>
<b>SYM\</b>	<i>Standard Library symbol files</i>
<b>OBJ\</b>	<i>Standard library object files</i>

#### Example of a backend C compiler installation (where required)

<b>WATCOM\</b>
<b>BIN\</b>
<b>BINB\</b>
<b>BINW\</b>
<b>H\</b>
<b>LIB\</b>

*etc.*

Please note that it is very desirable to place all Amadeus-3 related development directories at the same level within one single directory hierarchy, i.e. if you work on a system with multiple volumes, place everything on the same volume. While not absolutely required, this will very much simplify your life, as it allows the use of **relative** path specifications in most circumstances. E.g. the x2c.red compiler redirection file contains path specifications for all the above directories. If you have to include absolute paths and volume names, it will be very difficult to move the entire development directory hierarchy to a different location (e.g. when installing on a different workstation). By using only relative paths, this becomes a snap, as you do not have to change any directory reference. Good advice: try to assign the same name to the development directory, the main module etc. It will make your life much easier.

## 5.2 Files used in Amadeus-3 Developments

An application is composed of source code, of course, but there is more. You will find the following files with most standard Amadeus-3 applications development directories:

File name / Category	Description
Application.OB2	The main module of the application, containing Oberon-2 source code; contains at least some Amadeus-3 specific elements.
Other .OB2 files	Application specific source code modules
Application.AP3	The main application project file (object script), in binary format. Contains definitions for most user-interface objects, database elements etc. This file is usually generated with the A3Edit resource editor.
Application.TX3	This is the text version of the .AP3 file of the same name. It may be edited manually. It contains the complete description of all the user-interface objects in printable format.
*.AP3 and *.TX3	Your application may use more than one project file
Application.INI	This is a file for string resources. The format follows Windows and Motif conventions. cf. Module Resource for more information about this file type. A certain number of options must be defined in a file of this type for every application. Details specified below.
*.FMT and *.INC	Various report and report include files, as required by your application

The following files may be found on some platforms. They are derived from template files and may require maintenance.

Optional / Platform dependent files	Description
x2c.red	<i>C Backend compilers only.</i> The « redirection » file tells the compiler where to find specific files.
xc.red	<i>Native XDS compiler only</i> The xc.red file is optional, when you use the XC native compiler, provided you use the same directory structure for all your applications. Then it is enough to set up a single xc.red file in the \XC\BIN directory. Otherwise, place this file in the main directory of your application.
Application.ICO	Windows, OS/2: Icon file for the application
Application.RC	This is a Windows specific resource file, so not found on other platforms. It contains GUI-format resources. Use it to store embedded resources, such as bitmaps, cursors and icons.

The following files are generated by some utility program and may usually be discarded without risk, as they may be re-generated. They are usually only useful during development.

Automatic files	Description
any .OBJ files	Windows, OS/2: Object files, generated by front- or backend compiler
Application.SYM	Windows, OS/2: This file is generated by the linker and contains symbol information used by the debugger.
Application.MAP	Windows, OS/2: generated by the linker and used by some debugges.
Other .SYM files	Windows, OS/2: These files are generated by the Oberon-2 compiler when compiling source code modules. They are required for compiling higher-level modules.
any .C and .H files	<i>C Backend compilers only</i> Generated by front-end Oberon-2 compiler, when using a C code generator such as the Extacy/XDS compiler. NB: .H and .C files distributed with Amadeus-3 should not be deleted!
Application.RES	Windows: Generated by the Windows resource compiler, based on the Application.RC file. This file is used by the linker when building the final executable program.

In addition, your application may require icon files, bitmaps, cursors and very application-specific files. For Windows: The former may all be included in the Application.RC file; in this case, they will actually be included in the final executable file by the linker and do not need to be distributed separately to the end user.

### 5.2.1 The main module "Application.OB2"

The main module of every application must contain some elements specific to Amadeus-3 applications. The most important one is the declaration and proper initialisation of a `Project` object. This object is derived from the class `ProjectDesc` as defined in module `Startup`. It is responsible for the initialisation of an application and defines the main loop of the application (cf. « 2.10 Message handling »). The design goal behind this approach is to make it possible for every application to substitute its own methods where the standard behaviour is not appropriate, which includes the main event handler loop, program initialisation etc. etc. For details, please refer to module `Startup`.

Here is the main module as defined by the template application (slightly abbreviated and annotated). The application name was set to « Demo » and the code generator produced module « DemoApp ».

The “<+main\*>” tag at the beginning of the program is required for the Extacy/XDS compiler, to designate the module as a main program with an entry point. This is probably necessary in one form or another outside an Oberon OS, as only that OS supports direct execution of any piece of code, without specific program entry point.

The body of the module creates the project object, which we mentioned. This object manages application initialisation, execution flow and a top-level event handling through the method `Handler`. Typical examples of top level events include main menu commands, global shortcuts, main window resizing and program termination.



In this context, it may be useful to mention that closing the main window of an application closes the entire application under Windows.

Include a call to `Actions.Handler`, if you want to make standard mouse actions (drag & drop, moving and resizing objects etc.) functions available to your application.

The same goes for help support. If your application provides a help file, you should call the procedure `Help.Handler (w, ev)` to automatically react to help events, i.e. either the standard help key (under Windows F1) or one of the help commands, as defined in module `Commands`. You also need to include the name of your help file and the help topic index file in your application resource file (`Application.INI`).

In addition to the items mentioned here, most applications will add code to do any of the following during initialisation, i.e. in the `InitInstance` method:

- start and open your database(s) (module `Db`, `Btr` etc.)
- assign report preview windows
- check the user licence (module `Licence`)
- limit the number of simultaneously executing application instances etc. etc.

```

<+main* > (* this line required for Extacy/XDS compiler *)
MODULE Application;

IMPORT
    WinMgr, Startup, Events, Str, Keys, Commands, Dialogs, Actions,
    App:=DemoApp;

TYPE
    Project* = POINTER TO ProjectDesc;
    ProjectDesc* = RECORD (Startup.ProjectDesc) END;

VAR
    project*: Project;

PROCEDURE (p: Project) Handler* (w: WinMgr.Window; VAR ev: Events.Event);
BEGIN
    IF (ev.tp = Events.Close) & (w = WinMgr.mainW) THEN
        WinMgr.mainW.Destroy; (* standard program termination *)
    ELSIF ev.tp = Events.Command THEN
        ev.done := TRUE; (* anticipate command is handled *)
        IF ev.code = App.ExitCmd THEN
            WinMgr.mainW.Destroy; (* program termination *)
        ELSIF ev.code = App.AboutCmd THEN
            Dialogs.About; (* display copyright information *)
        ELSE
            ev.done := FALSE; (* our guess was wrong, not handled *)
        END; (* if ev.code *)
    END; (* if ev.tp *)
    Actions.Handler (w, ev); (* standard object manipulations *)
    Help.Handler (w, ev); (* standard help handler *)
END Handler;

PROCEDURE (p: Project) InitInstance* (): BOOLEAN;
BEGIN
    IF ~PStore.LoadProject (p.name^, App.thisProject, App.Init) OR
        ~p.InitInstance^ ()
    THEN RETURN FALSE END;
    WinMgr.mainW.Show (WinMgr.SamePos); p.Logo;
    RETURN TRUE;
END InitInstance;

BEGIN (* this code is executed before the main loop gains control *)
    NEW (project); project.Init; project.SetName ('Application');
    Startup.WinStart (project, 0);
END Application.

```

## 5.2.2 The string resource file "Application.INI"

Most applications require a minimum amount of string resources and variable parameters. Under most GUIs, it's become quite common now to use some form of .INI file with the purpose of storing parameters for future reference.

Under Windows and Motif, this is done in .INI files that are simple text files where information is grouped with headers into specific information sections. OS/2 uses a binary INI file format that requires specific tools for inspection and maintenance.

Actual string resources, on the other hand, are not necessarily stored in this form. Windows applications usually specify strings in the .RC resource file, so they get linked with the executable file.

Amadeus-3 again goes it's own way here. It uses the standard Windows/Motif-style .INI format, but it uses this file format to store string resources, too. Being a simple text file, this kind of .INI file is easy to inspect and maintain. It may also be printed for direct inclusion in the documentation, which is quite an advantage.

Placing string resources outside the executable file makes it possible to translate an application with nothing more than a text editor and maybe the A3Edit resource editor. If the linker is required, then the complete development system must be available before a new language-specific version of the program can be built.

Dynamically changing the active language becomes possible, too, a feature that has been used many times since the days of Amadeus-2.

Here is the Template.INI file, which contains all of the essential fields expected by basic Amadeus-3 applications:

<pre>[Project] App=Demo Icon=Demo Help=Demo HelpMap=Demo Author=... Year=... Hash=97 Font=Arial;h=13;x;p=v;a=3  [Str] RealInfo=.,' Yes=Yes No=No Ok=Ok Cancel=Cancel Abort=Abort Retry=Retry Ignore=Ignore</pre>	<pre>Help=Help Landscape=LANDSCAPE  [Msg] Done=Operation complete SaveOk=File saved Error=Error occurred SaveErr=File not saved Warn=Warning NotComplete=Incomplete data  [Err] Error=Error occurred in application  [Prompt] DelOk=Do you want to delete this item?</pre>
--	--

The `[Project]` section is essential. This is where the required object script is specified. If the `App=` entry is included, then the program initialisation will fail if no valid `Application.AP3` or `Application.TX3` file is found.

The `Icon=` entry specifies the name of the basic application icon that should be displayed when the application is ionised.

The `Hash=` entry specifies the dimension of the system hash table for object names. Given the nature of hash tables, this number should always be prime. It may (and should) be smaller than the number of actual object names stored in the system hash table, but it should not be too small. Usually, it should be about 1/3 as large as the expected number of names. If you do not specify a number for this item, then no system hash table will be allocated and your application may not be able to operate properly.

The `Font=` entry defines the system font for your application. Instead of using the user-defined system-font, it is usually preferable to define a specific font as standard application font, as most screens were probably designed with a specific font in mind. While it is possible to adjust the screen layout dynamically, this does not often yield pleasant results. A good choice for Windows programs is 13pt. Arial Bold, which is specified above. For more information on font definition format strings, please read the chapter « 7.54 Fonts ».

The next section, `[Str]` contains a few standard strings that may need translation, when an application is localised.

The `RealInfo=` entry defines the characters to be used for real numbers; first 2 characters that may be used for the decimal point, then a character that defines the tick mark.

The next section, `[Msg]` contains strings used by the standard `Dialog.Message` functions, which includes titles and message bodies.

The next section, `[Err]` contains strings used by the standard `Dialog.Error` functions, which includes titles and message bodies.

The section `[Prompt]` also specifies strings, but those that are related to prompts via the `Dialogs.Prompt` function, such as « Do you really want to delete this item? », as included above. The default prompts are those accessed by some standard procedures, e.g. the “DelOk” string is requested by module Scroll when the standard Delete command of a window is activated. Given that any prompt may require translation, a string of this kind has to be located outside the source code.

### 5.2.2.1 Variable Parameters

You may want to have a group of parameters that are specific to a particular installation or use. You may simplify your life by using the calls `Resource.GetParam` and `SetParam`, after setting the group name with `Resource.SetParamGroup`. Inside the INI file, the different parameter groups are identified as, for example:

```
[Param]
[Param.Server]
[Param.Client]
```

`Startup.project.LoadParameters` automatically scans the command line supplied when starting the program for a parameter identified as “env:”. The following string will be used as parameter group name. To run your application on a server machine, you may hence start the program with “Application env:Server”, telling the application to use those parameters under `[Param.Server]`. From a client machine, you would run it with the “Application env:Client”, indicating that the program should use parameters supplied in the “Param.Client” section, particularly directory names etc.

If you use `Licence.ob2` with the utility 05.4 Licence Editor, you may also supply the environment name in the “env” field of the licence record. This will be used by default, the command line parameter will override it.

If you use the `Resource.GetSystem` and `Resource.SetSystem` calls, the search will affect the WIN.INI file in the Windows System directory first (or the closest matching file in another OS), which allows for machine-specific settings, e.g. on a network, where many people share the same application directory.

In WIN.INI, the parameter section will be named

```
[Application.Param]
[Application.Param.Server]
```

where `Application` is the name of your `Startup.project`.

### 5.2.3 The Standard Resource File Application.RC

This file is only required under Windows and OS/2. It may contain any of the following:

*Standard dialogs, Menus, Bitmaps, Icons, Mouse Cursors*

and much more, in fact. But we are not really concerned by many of these, as most of it is provided in a platform-independent form with the **Amadeus-3** object scripts, with the corresponding advantages. But as this file is mandatory for all applications, we may as well use it at least for a few things:

- Bitmaps, icons, cursors etc.: By specifying any required bitmaps and other bitmapped files in the .RC file, these resources will be integrated directly into the executable file and therefore don't have to be distributed separately.

The standard dialogs can be edited with any of the current crop of widely available tools, for example with the standard Windows SDK Dialog editor, the Watcom WDE, the Delphi resource editor etc. You may of course also use a simple text editor to make small modifications that do not require layout changes (difficult to do in the code).

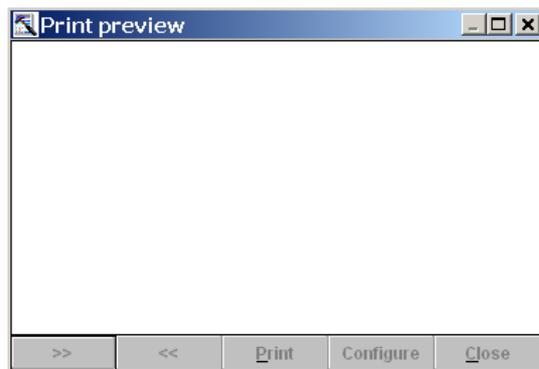
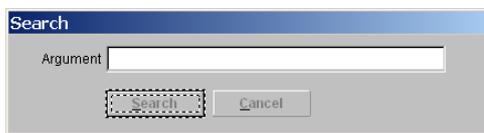
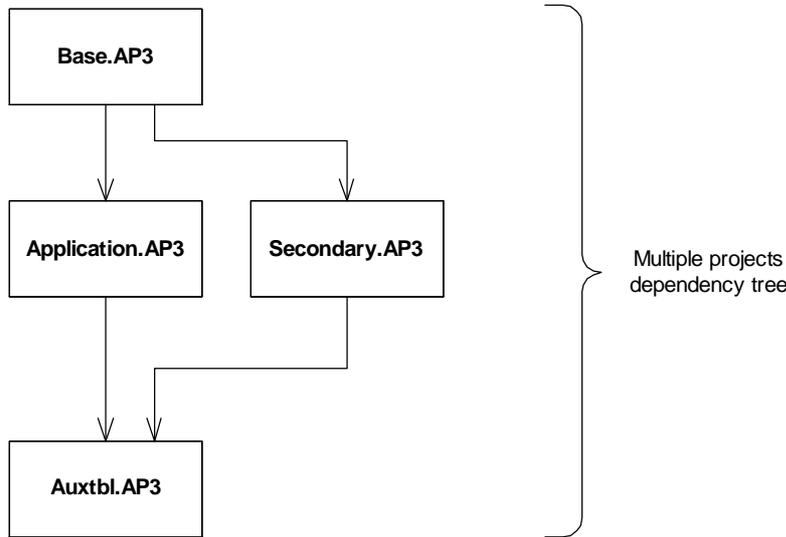
To display such a standard dialog, use the procedure `Dialogs.Process`, as demonstrated in all sample programs (cf. procedure `WinEvent.defHandler`, usually assigned to a procedure in the main module of each application).

Also note that bitmaps and other such files are specified with a relative path name, i.e. « `bitmap\` », for the reasons already exposed earlier.

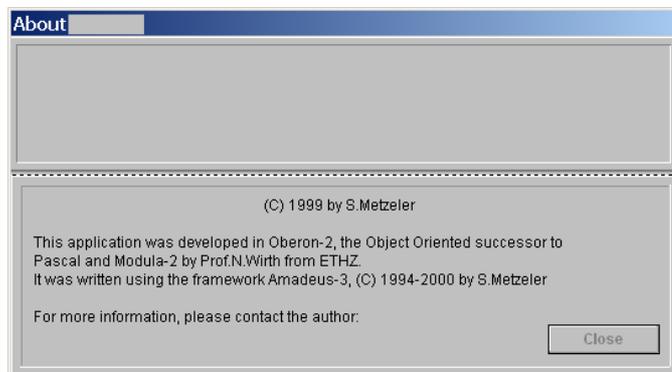
### 5.2.4 Basic application objects

Many user interface objects and other elements that are defined in object scripts are common to most applications. You may want to re-use them for each new application that you write. The easiest way to achieve this, while maintaining a lot of flexibility for presentation aspects and extensibility, is to create a basic object script, which you copy to each new application you create.

Then you create one or more additional scripts for all application-specific objects, which can then be loaded as dependent or independent projects.



**Some typical windows you might place in Base.AP3**



### 5.3 Files to be distributed with end-user applications

Naturally, you would not want to distribute all of the above files to the end-user of your applications. If you do not use an object script (.AP3 or .TX3 file), you don't need string resources and you don't use external .DLL libraries such as Btrieve, then your Amadeus-3 executable program may well be distributed as a stand-alone single .EXE file (or whatever is appropriate on your platform).

But more often, it is likely that the end-user will need at least some of the following files to run your application properly:

File	Description
Application.EXE	Windows, OS/2: The actual executable / UNIX-style Executable
Application.AP3	Binary version of object script
Application.INI	String resources

If you made use of the Pervasive database manager, the user has to install one of the versions of the database, either a workstation engine, a server engine or a client, if it is only used to access a database server engine on another machine.

If other libraries are required, you will have to make sure that the end-user has the appropriate executables and dynamic link libraries in a place where the system will find them. You could of course install these support files into the same directory as your application, but this would be wasteful if other applications use the same tools. Under Windows, for example, these files may be installed in the Windows\System directory or some other directory that is included in the path.

File	Description
<code>tx32.dll</code>	If your application uses multi-line fields and you wish to use the TX library, to gain full text editing capabilities, instead of the very limited standard text fields as supported by Windows. When you distribute applications that use this library, you <b>must</b> purchase a licence for the TX library.
<code>vic32.dll</code> <code>victw.dll</code> <code>etc.</code>	If your application imports module ImgFiles, which you will want to do when it uses extended graphics functions, loads or saves various bitmap formats (JPG, TIFF etc.); you only need to include victw.dll for scanner function support with Twain interface. When you distribute applications that use this library, you <b>must</b> purchase a licence for the ACCUSOFT Image library.

All of the above DLLs are included with the Amadeus-3 library for *development only*.

NB: If your application does not use any of the above tools, you don't have to distribute these files with your application, nor do you have to purchase the corresponding licences.

Additionally, your application may require specific files, such as report files, bitmaps, etc.

You'll have to make your own list as you develop the application.

### **5.3.1 Installation tool**

You can find many excellent platform-specific installation support tools on the market, particularly for Windows and OS/2. These tools usually cost no more than 100 to 500 US\$, with varying levels of sophistication. You can expect all to work just fine with your Amadeus-3 application. The result is a very professional end-user installation. The Microsoft SMS Installer is perfectly acceptable.

You can also make use of the Windows built-in tools for the creation of setup tools, which is a bit more involved. Most professional installation tools are almost overkill for Amadeus applications, as they are so easy to install and require almost no changes to the system. Such simplicity is one of the strongest points of Amadeus applications over the typical Windows development tools, which create huge and complex applications, which also require a large number of registry entries, utilities etc.

## 5.4 Licence Editor

Amadeus includes a licence editor, which works in conjunction with the module Licence.ob2. It allows you to edit a database file named (per default LICENCE.DAT), which is encrypted and contains user- and application- and installation-specific information.

You can use it in various ways, especially to include data about the licence holder, his coordinates and his specific environment. This is very far from a perfect anti-piracy protection tool, but by including some essential data elements, you can make the application harder to distribute, especially when some elements from the licence data file are used in reports, such as the customer's name, cf. below. This will make it hard to use your applications by professional users without a minimum amount of customization, which you will likely provide against payment of the appropriate licence fee. According to the threat level – probable competence of the average user, cost of legal licence against effort to “crack” the application, likely volume of distribution – you may want to add several other elements to make it harder to break, such as not using clear-text reports or checking report files against tempering.

In the suggested configuration, you create a sub-directory named LICENCE, into which you place the user's LICENCE.DAT and LICENCE.DFI files. You may use a test licence file, which resides permanently in the application's main directory. When no user licence is present, this file will be used instead. It will only contain dummy information for testing purposes.

The licence file also contains a default field named "env". If this field is filled in, it will be used as the default environment, for use with module Resource.OB2 and its parameter mechanisms: GetParam, SetParam etc. See paragraph 5.2.2.1 Variable Parameters for more details.

On the second tab of the editor, you can freely define a list of application-specific variables, which you can add, modify and delete. Each is composed of name and assigned string value. When you include the application data with Licence.Load or Check, these variables and the default fields of the licence record will be loaded into your Startup.project.dict data dictionary, where you can access them normally, e.g. through the StrFmt.Translate command.

As an example, let's assume that your application prints invoices for the client company. Instead of hard-coding the client's name and address or including it from some other source, you use @[licence.name], @[licence.adr], @[licence.city] etc. A custom-made field might be the company VAT number (as would be required in the UK). You would add the field “vat” to the custom field list and set the value to the customer's VAT number. In the invoice report, you would include it as @[licence.vat].

NB: If the environment name is supplied on the command line, this overrides the definition found in the licence file.

To edit licence data:

- launch LicenceEdit from the target directory, where you wish to create or edit the licence file
- launch LicenceEdit and specify the target path as parameter
- launch LicenceEdit and use the function "File / Open" to specify the target directory

Fill in licence data and confirm with ACCEPT to save the changes or CANCEL the entry, which is the same as "File / Exit".

There is a connection with the Application Launcher, cf. 5.5 Network Application Launcher :

The parameter licence.checkExe labelled “User control over new EXE files” in the editor. If you check this field, then the user is prompted before a new executable file is copied to the user's installation from the source directory. This may be desirable where security is of greater concern, but you still want to allow a certain amount of automatic updating.

## 5.5 Network Application Launcher

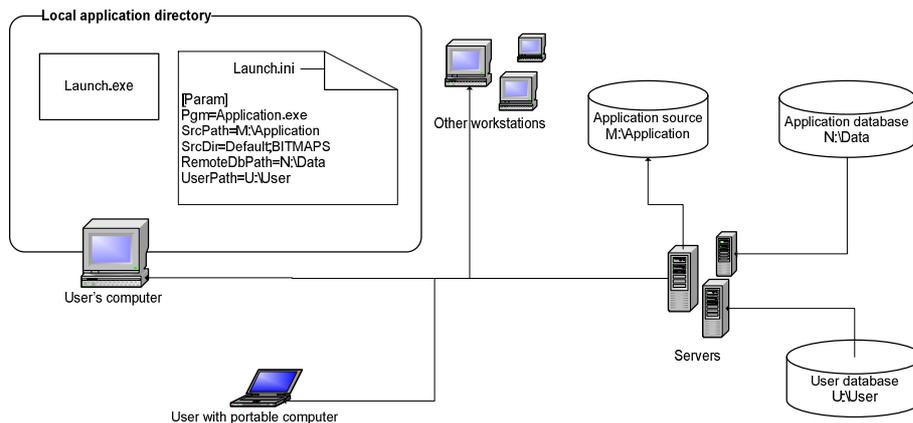
Network applications may be run either from a shared network drive or from the local computer of each user:

Running the application directly from the Network drive has the advantage that everyone is always using the same version, but it can be slow and might add substantially to network traffic, if many users start running their application at the same time. It also makes it more difficult to manage local / temporary files, since each user still has to have an individual location for such files.

Alternatively, you can install every application locally on each user's computer and share only the data. This has the advantage that the application will load much faster, that local an temporary files can be created directly in the local application directory. The disadvantage is that the user's copy of the application may not be up to date and might require updating before running, to avoid conflicts with a new database structure or changed communication protocols etc.

The Amadeus Network Application Launcher combines these respective advantages and eliminates the drawbacks. In fact, Launch.exe uses parameters supplied during installation - either via a desktop icon, a command file or the file Launch.ini - to locate the source directory on the network, from which it will copy any new or changed files, removing any that are no longer necessary and then starts the application with the proper parameters.

NB: The interpretation of the user- and application database are application-specific.



### 5.5.1 Parameters

These are the parameters supported by the launcher application:

INI Parameter	Command line	Description
Pgm	pgm:	specifies the application to run (with or without extension)
SrcPath	src:	specifies the location of the user database
SrcDir	dir:	specifies source directory of the application
UserPath	user:	specifies a list of sub-directories to copy from the source directory
RemoteDbPath	last paramter on line, no name	specifies the location of the database
Env	env :	specifies the environment parameter set name, e.g. "Server" or "Client"; matching parameters are found in INI file under section [Param.Test] etc. Per default, [Param] section is used; this section may also be found in LICENCE.DAT file, cf. information on licence editor
DevPath		Root of development directory tree structure

The launcher application will copy all new or changed files from the source directory and remove all those files from the target directory that don't exist in the source directory, effectively synchronizing the two directories. It will also delete all source and other standard development files, so you can use the development directory as source.

To protect your source files, the launcher will never accept as target a directory that is located in your development environment. You may specify the name of the development directory root in LAUNCH.INI as DevPath. The default is "\\dev\".

There is a connection with the licence editor, cf. 5.4 Licence Editor. NB: When a Licence.DAT/DFI file is already present on the local machine, it will never be updated automatically by the Launcher.

## 6. Important Programming Rules

In programming, rules are everything, this is no news. Some of them are so basic that they are hard to point out, because we don't even think about them anymore, they've become second nature. Often we establish our own set of rules through long experience, but we never think of writing them down. And often, we have to re-invent and redo things the hard way because we didn't turn working ideas into a good set of rules for future reference.

To break with this tradition, please take the time to read this chapter and add to it out of your own experience. This is where some rules and good practice for working with **Amadeus-3** are explained.

### 6.1 Guiding principles in coding for Amadeus-3

The code for **Amadeus-3** was written based on long experience with 2 such systems for MS-DOS and Modula-2, not to mention the many years of application and system programming on various other platforms.

One of the most important things learned was the importance of elegant, readable code where a clean conceptual design takes precedence over apparent efficiency and flashy performance. It is true that an application that was written by following strict coding standards is often slightly slower than one optimised for speed. On the other hand, optimisations that are based on fine-tuned code rather than improved algorithms are usually very hard to change, whereas well-designed code is much more flexible and forgiving. The performance penalty is usually very slight (assuming that the algorithms used are efficient) and more than compensated by the constant improvement on the hardware side.

Don't miss-read this statement: any application program should always be written so as to run with acceptable performance on existing hardware. Faster hardware in the future should not be used as a justification for sluggish code now. No end-user should be forced to buy new hardware just to be able to use a given application due to bad design and bad coding. This is unfortunately exactly what most industry leaders keep doing all the time, not to mention any particular billion-dollar companies, who seem to specialise in user-frustration.

In coding **Amadeus-3**, priority was therefore given to:

<b>1. Correctness</b>
<b>2. Functionality</b>
<b>3. Readability</b>
<b>4. Performance</b>

in that order!

#### 6.1.1 Correctness

For obvious reasons: if it isn't correct, the code has no value at all. So occasionally, you will find strange fixes that were simply required to get the code working despite problems with a particular compiler or the GUI system.

#### 6.1.2 Functionality

If the code doesn't deliver the desired functionality, it may be severely limited or unusable. So functionality is important. This led to a few more than sub-optimal design-decisions, which were simply forced by the underlying GUI or compatibility issues.

A good example for this is module « 7.64 Controls » and all the related modules. You will have a hard time understanding what is going on in there, because it is occasionally quite messy. Unfortunately this was unavoidable, due to constraints of the Windows message-passing and subclassing system, but it does work!

To enhance portability between 16 and 32 bit compilers, it was also necessary to introduce quite horrible code due to differences in the result type of the standard functions `SIZE` and `LEN`. This might no longer be necessary as soon as all compilers agree on the result type of these functions (`INTEGER` or `LONGINT`). Differences in how addresses are treated also created problems. Due to the attempt to have only the most minor differences at least between versions for different compilers on the same platform, some code is a bit awkward to read, although this is only at the presentation level.

### 6.1.3 Readability

Next came readability. Although some local optimisations were used, which may not be easy to read, the readability of the underlying structure and design was kept at a very high level. Without readability, code is simply impossible to maintain.

### 6.1.4 Performance

Finally, performance was enhanced where this did not interfere with the other aims. Performance should not really be a serious problem, where correct algorithms were selected. The performance gain through the use of a better algorithm may improve performance by a factor of 100 or more, whereas small optimizations will only provide a few percent improvement.

## 6.2 Compiler settings

Always compile with the same set of options, in particular for data alignment. `Amadeus-3` tries to find out about data alignment (Module `MemOps`, `GetAlignment` and variable `align`). But if this module was compiled with a different alignment option than some other module that you link it with, the information in module `MemOps` may not be correct.

## 6.3 Linker settings

At link time, you must specify the amount of stack space that you will require. The usual value should be between 30 and 60KB. You should avoid placing large structures on the stack. Create them dynamically on the heap instead!

Your program should only require more than 60KB of stack space if it uses intense recursive algorithms with deep nesting. But be careful with this, as GUIs based on co-operative multi-tasking won't like applications hogging resources and processor time for the evaluation of deep-nested recursive algorithms. Try a different approach if you run into this kind of problem.

## 6.4 Coding standards

When you read and write code, it helps if you stick to strict rules, beyond those imposed by the compiler. Although the Oberon-2 language does a superb job checking your code for consistency and enforces strict type compatibility and cross-module calling conventions, there is still room for improvement.

### 6.4.1 Naming convention

Variables, types, constants etc. should be easy to recognise by their name. The language only enforces the convention to spell standard identifiers in all uppercase. A system proposed for Modula-2 has shown to produce very readable code:

- All variables start with a lowercase letter
- All constants, types and procedures start with uppercase letters

This allows the easy distinction between direct and indirect calls of procedures:

Examples:

The statement

```
RunSomeProc ('abc');
```

is – according to our convention – a direct call to a local procedure, whereas

```
runMyProc ('abc');
```

is an indirect call to a dynamically assigned procedure.

Following the same idea, you can tell that in the expression

```
result := 5 * FrameWidth + margin;
```

`FrameWidth` must be a constant, whereas `margin` and `result` are variables.

By applying this convention consistently, you can ensure an even better code readability at zero cost (except for the effort of getting used to it).

Other rules, such as how to actually choose your identifiers, is another matter. It's quite usual though, to use the following *within local procedures only!*

a,l,n,m,i,j	for <code>Integers</code> , <code>ShortInts</code> and <code>LONGINTs</code>
x,y	for <code>INTEGERS</code> when coordinates are meant
b	for <code>BOOLEANS</code> , in particular dummy results that may be ignored
ok	for <code>BOOLEANS</code> with a longer lifetime
r,x,y	for real values
s,t,u,v	for <code>ARRAYs OF CHAR</code> (Strings)

#### Special case:

Class and object naming. When naming an object class, use the ETH standard, i.e.

```
WindowDesc = RECORD (base class) ... END;
```

is the base class record,

```
Window = POINTER TO WindowDesc;
```

is the pointer to the base class record.

**More specifically to Amadeus-3**, there are a few conventions that will be used implicitly by the A3Edit code generator:

- Window objects will have a name ending in 'Win'
- Database objects have `Db` added to their name
- Database Files end in `F1`
- Database key constants end in `Key`
- Command constants end in `Cmd`

Examples:

If you name a window `userEdit`, it's going to be exported as `userEditWin`.

The constant representing the command code `AddUser` will be exported as `AddUserCmd`.

The database `main` will be exported as `mainDb`.

The database file `user` will be exported as `userFl`.

### 6.4.2 Parameter passing

Although parameter types are checked by the compiler, it is better to use a convention that helps you decide the ordering of parameters. Sometimes, parameters are all of the same type and very often, you have just too many parameters to make sense if not placed according to some logically consistent system.

The most important rule here is to write input parameters at the beginning of the list and output parameters at the end of the list, with modifiers in between: **IN + MODIFIERS -> OUT**

Example: `Convert.IntToStr` takes an integer and attempts to convert it into a string.

```
Convert.IntToStr (i, 5, result);
```

where 'i' is the integer to be converted, 5 is the number of characters expected in the resulting string (a modifier) and 'result' is the resulting string.

**Special case:** One parameter is used as input to a procedure and will also receive the result. In this case, the target parameter should be placed first in the parameter list, to make it's predominant role easy to understand.

Example: Selection of a file name through a dialog box, where the current value of the file name parameter will be used as default selection:

```
Dialogs.FileName (fileName,title,dir,filter,
                 defExtension,write,create): BOOLEAN
```

The `fileName` parameter is input as well as output of this procedure, whereas the other parameters are modifiers.

### 6.4.3 Code formatting

This is less vital, but still a strong factor influencing the readability of your code. Although a pretty-print utility could easily rearrange your code, this is not very desirable for several reasons:

- The output of any pretty-print utility is usually less than perfect
- A pretty-print utility does not know about internal dependencies in your code
- It requires that you run a utility before you can distribute your code
- It doesn't help while writing code
- It might confuse you, as the result may look different from what you worked on
- Automatic file comparison will generate many more differences, a handicap for proper version management (unless your version manager works on character- not line-base and ignores white space, which would help)

For all these reasons, it seems advisable to spend a little time and effort on developing a consistent coding style, with the emphasis on *consisting*.

Style used in Amadeus-3:

- Indent always by 3 blanks (not tabs, as those tend to be interpreted very differently by various utilities)
- Opening parentheses and brackets always preceded by single blank
- Closing parentheses and brackets followed by a blank
- Operators (+ - \* / & etc.) always surrounded by blanks
- If logical consistency and readability are maintained, more than one instruction per line used, mostly in the case of many short instructions
- Variable declarations: placed on single line with VAR statement if only one variable type used, otherwise indented normally with no declarations on the same line as VAR statement
- Record declarations: RECORD name aligned with matching END statement
- Constants, Variables etc. align on '=' or ':' sign, where possible
- In long IF, LOOP, WHILE, WITH, RECORD statements, place comment after matching END, specifying at least the type of statement being matched, usually enhanced with further comments
- When local procedures are used, add comment with procedure name after BEGIN of upper-level procedure
- Always leave one blank line before procedure declarations (not more than one line either)
- Comments describing a procedure and its parameters should be placed immediately after the procedure header
- Don't overuse comments, as they tend to hide the code; concentrate on making your code very readable instead; use comments to clarify the ideas behind the code

**NOTE:** Feel free to change any or all of these rules to suit your own taste, as long as you *do* have rules.

There are good reasons for this formatting style. Some people prefer to place no spaces around parenthesis and operators, others put blanks before parenthesis and brackets, not after. Logically though, the code within parenthesis and brackets forms a single, coherent unit, which is why it should be offset from the surrounding code. Keeping operators detached also incredibly enhances readability.

Next page: Example of typical formatting style used in Amadeus-3 with formatting comments added after the '--' symbol.

```

MODULE WindowDemo;

(* ----- *
 * >> Copyright, Version management codes etc. placed here << *
 * ----- *)

IMPORT
  Persist,WinMgr,Files,Paths,Db,DbView,Scroll,Graphics,Colours,Fonts,
  Str,Convert,SYSTEM; (* import list in compact format *)

(** ----- *
 * Place module description here, just after the IMPORT statement, *
 * ----- *)

(* NOTE: Exported comments start with '***', as by the Oakwood report *
 * The order of declaration for constants, types and variables is *
 * defined by the Oberon-2 standard. *)

CONST
  Max          = 200; (* Comments concerning constant placed here *)
  FrameStyle = 1;

TYPE
  Window* = POINTER TO WindowDesc; -- short, compact declaration
  WindowDesc* = RECORD (WinMgr.WindowDesc) objId: LONGINT END;

VAR windowId: INTEGER; (* single variable *)

PROCEDURE (w: Window) Init* ();
BEGIN w.Init^; w.objId := 0; -- compact style more readable here
END Init;

PROCEDURE AttachObject* (w : Window; personId: LONGINT;
                        fmt: INTEGER): BOOLEAN; -- note alignment!
(* Procedure and parameter description goes here *)
VAR
  n : INTEGER; (* multiple variable types, so split lines *)
  array : ARRAY 3 OF INTEGER;

  PROCEDURE SetPrevious ();
  VAR wp,prev: WinMgr.Object;
  BEGIN prev := wp; wp := wp.next;
  END SetPrevious;

BEGIN (* AttachObject *) -- include matching procedure name
  SetPrevious;
  (* compact FOR statement; use LEN to find upper ARRAY boundary *)
  FOR n := 0 TO LEN (array) - 1 DO array [n] := 0 END;
  (* Example of expression with lots of spaces to make it readable *)
  array [0] := Max - 5 * (array [1] + array [2] + 2) DIV 7;
END AttachObject;

PROCEDURE MakeWindow (VAR po: Persist.Object);
(* Procedure doesn't do much, so keep it as compact as possible *)
VAR w: Window;
BEGIN NEW (w); po := w;
END MakeWindow;

BEGIN Persist.Register (MakeWindow, windowId);
END WindowDemo.

```

## 7. Modules

Here is a brief description of all the modules that are part of the **Amadeus-3** framework. As the designation “framework” implies, there are some strong dependencies between at least a certain set of modules, i.e. they could not easily be used outside the entire framework. They have all been carefully crafted to be as small and flexible as possible, mostly grouping only logically related objects and functions together in the same module, except where internal dependencies forced certain object classes to co-exist, as for example in module `WinMgr`, which finally became much larger than any other module.

**Amadeus-3** does a few things for the sake of efficiency and programming power that should be strictly avoided by application programs. The time and effort spent on making these modules safe would be wasted on any code that is not expected to be re-used frequently.

Great care was taken to make **Amadeus-3** as portable as possible, even where certain low-level functions are being used. Even the recommended 3<sup>rd</sup> party libraries were selected with an eye on their portability. This will only work if your application does not attempt to use system-specific information. If portability is of concern to you – even as a future eventuality - you should always use **Amadeus-3** services, where available, instead of system-specific calls, as otherwise you will sacrifice portability. If you are willing to do so, go ahead and use any system-specific features that you need, including non-portable 3<sup>rd</sup> party libraries.

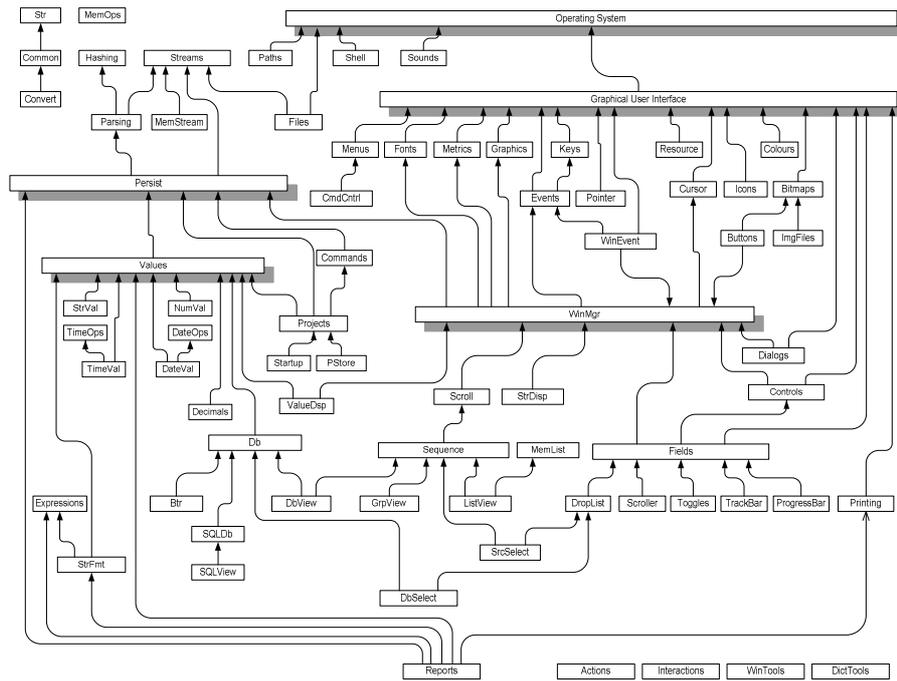
The modules are arranged by complexity. Simple, low-level modules come first. Please refer also to the module diagram, which should give you a clearer picture of the major dependencies in the **Amadeus-3** framework. As you can easily see in this diagram, some modules are not included as they are far too general to show meaningful dependencies graphically. Examples are the module `Str` for string operations, which gets used all over the framework and throughout applications. There are also the very low-level modules `MemAlloc` that manages the heap on systems that do not have proper support built in (such as Windows...) and `MemOps`, that simply supplies very low-level memory manipulation functions.

You will also come to realise that not all dependencies could be depicted without making the diagram much less readable. But be assured that all the arrows would still be pointing in the same direction! There are no circular references in Oberon-2, contrary to Modula-2, where definition modules make circular references possible, and most other languages, where such rules have never even been considered; in C/C++, nothing could stop you from building a library with chaotic cross-references. In Modula-2 the initialisation sequence is determined by the compiler (as in Oberon-2), which takes at least some of the risk out of circular references. The fact that Prof. Wirth decided to discard this option from the language shows quite clearly that even the controlled risk under Modula-2 was deemed to great and the reward almost non-existent.

There is no situation, where you could not replace 2 modules with some circular references with 1 to 3 modules with no circular references. Either you re-arrange their elements between them, or you regroup the two modules into one, or you extract any common elements and place them in a third module, that is referenced by both.

NB: The following chapters are in order of ascending complexity and dependence on preceding modules.

**Module overview**



**Figure 61 - Module Overview Diagram**

Please note that the above diagram is really just an overview. Dependencies are only hinted at and by no means complete! But it does convey an approximate indication of the hierarchy of modules. Some modules, such as Str and Convert, are used almost universally. Others are at a high level of abstraction and use a lot of the lower-level modules, such as WinTools, DictTools, Interactions etc.

## Common declarations, memory management and related modules

### 7.1 Common

Generic constants, types, regional codes (imported from OS) etc.®

### 7.2 MemAlloc

*Only with C pre-compiler versions*

This module implements decent memory allocation and heap management for the Microsoft Windows® 3.x environment with the Extacy compiler and should never be called directly by the application. It will probably not be necessary on most other platforms.

In fact, this module is inserted into the compiler runtime library so that all calls to `memalloc` and `free` actually go through `MemAlloc.Alloc` and `MemAlloc.Free` then manage memory blocks allocated from Windows as a true heap. The only limitation of this is that memory blocks so allocated are not movable in a Windows Standard Mode set-up.

Algorithm used: this module is based on a discussion of the heap manager in the Oberon System by Prof. Wirth and Prof. Gutknecht. It shows nicely how clean even low-level code can look under Oberon-2, which is why is still included, despite the fact that on most platforms, it is not required.

### 7.3 MemOps

Very low level memory operations.

*This module implements functions that may make your code unsafe or non-portable. Only use it if all else fails and when you know precisely what you are doing – meaning that you understand all the implications of these functions and encapsulate such calls properly.*

To make this restriction a little more precise: DON'T use it, where you can achieve the same results by using standard language methods, such as pointers, assignments, loops etc. Among others, it is used to cope with C programs that use a integers and long integers as sets of bits. Never use it this way when you write your own code, define a `SET` variables instead!

`MemOps` also tries to determine a few basic parameters concerning your platform, in particular: the alignment factor (usually set through a compiler switch, but sometimes platform invariant) and the memory overhead for empty records.

The memory alignment, offsets and other functions were used to establish the connection from externally declared variables to their internal representation in the data dictionary (cf. module `Values`) and then again in the database interface (cf. `Db` and `Btr`).

One particularly tempting functions is `MemOps.Clear`, which clears an entire data structure. The most typical error with this operation is that you use it to clear not the data structure but the pointer to a data structure, because you forget to add the little reference marker “^”. As a pointer is just as compatible with `ARRAY OF BYTE` as any other data type, the compiler will it let slip. This is something that usually happens in C, where it is very easy to confound pointers and objects they reference. Don't let it happen to you<sup>13</sup>.

Consider any module that imports `MemOps` as equivalent to a module that imports `SYSTEM`.<sup>14</sup>

**NB:** Please read the note on "Compiling" in section « 6.1 Compiler settings ».

<sup>13</sup> Under compilers that support `ASSERT`, `MemOps.Clear` will at least reject any array that is of the same size as a pointer, which should avoid this type of problem or at least help you pin-point it quickly.

<sup>14</sup> Unfortunately, even `DISPOSE` ended up in module `SYSTEM`, as the basic Oberon system supplies a garbage collector.

## String manipulation and formatting

### 7.4 Str

Basic string functions, portable and extensive. Remember that many string support functions are part of the Oberon-2 language, in particular string assignment `COPY` and string comparison (implemented through the standard operators "`<>=#`").

`Str` defines a special data type, the `PStr` or `Pointer to String`. This is just a `POINTER TO ARRAY OF CHAR`, but it does allow the consistent use of variable-length strings. There are also several support functions for this data type.

There are a few extended ASCII support functions. They should help you write programs with support for most European languages. In particular, there are variables defining lower case and upper case equivalents of accented letters, as well as their plain ASCII representation.<sup>15</sup>

### 7.5 Convert

Numeric to string conversion and formatting functions. Includes every numeric type, from `SHORTINT` to `LONGREAL`, with the notable exception of `REAL`, which is considered quite useless (who wants a real number with a precision of 5 digits). Of course, `LONGREAL` should really be called `REAL`. If you wish to save memory by using short `REAL` numbers, you should only perform the conversion when you are sure that you will not lose any precision.

There is even support for serious real-number formatting, i.e. what engineers and scientists have come to expect since the first HP scientific calculator. You'll find precisely the same formatting styles, `STD`, `FIX`, `SCI` and `ENG`. For non-HP users, this is the meaning of these modes:

Mode Name	Description and Samples
<b>STD</b> Standard mode	Show whatever is defined, truncating trailing zeros, switching to scientific notation when required: 2                    540.3454                    7.43E53                    -47.5
<b>FIX</b> Fixed comma	Always show the specified number of digits, switching to scientific notation when required; for 2 digits displayed: 2.650                    1.00                    -58.35                    2.00E19
<b>SCI</b> Scientific notation	Always display numbers in fixed comma, scientific notation: 1.0000E0                    5.354E-10
<b>ENG</b> Engineering mode	Always display numbers in scientific notation, but exponent always set to multiples of 3, moving the comma to compensate: 10.0E3                    573E-6

Another feature for real number formatting is the possibility to include high-commas, i.e. separators for groups of digits before the comma. You may freely define the character that is to be used for this function. Default: ' ' ', set in variable `tickMark`. You may define the local comma sign by setting the character in the variable `dot`.

### 7.6 StrFmt

This module contains several high-level string functions, which are closely related and universally used throughout Amadeus-3.

The formatting codes are especially important, when used with reports and data import and export.

<sup>15</sup> This won't help if you want to move into the Asian market, but if that is your target, you should probably choose a tool developed by an Asian company in the first place. For programs targeted at the US and European market, Unicode characters make about as much sense as taking a Ferrari to pick up your groceries.

### 7.6.1 Output Formatting

StrFmt.Translate supports the translation of strings with embedded variables into result strings, where the embedded variables have been replaced with their contents.

Example: Your application supplies a string to be converted of the following format:

```
"Client name: @[client.name], born on: @[client.birth]"
```

where "client" is a record defined in a data dictionary, with fields "name" and "birth"

(let's assume that `client.name` is of type `StrVal.Value` and `client.birth` is of type `DateVal.Value`, although it this is not important in this context).

If `client.name` = "Jim Brown" and `client.birth` = "30.4.1973" then the above string will translate as: "Customer name: Jim Brown, born on: 30.4.1973"

Here is a list of the various formatting parameters:

@[name;format] or @[name,format] include a data dictionary item in output string

format elements may be separated with either ";" (semicolon) or a "," (comma):

The format may specify the following:

"#n" designates the index into an array variable, eg. embed variable result.table [5]: "@[result.table;#5]"

Output alignment specification: (adjusts the output relative to the maximum field length)

"<" Align to the left; "^" Align to centre; ">" Align to the right;

"&" Output compact, no alignment

"-" Output without formatting, i.e. raw data

all of the above may optionally be followed by the field length; default is maximum variable length

"+n" Number of characters to use for real number representation

"x" or "x" Filling character; NB: actual quote characters, ' or " MUST be included !

"~x" Discard the following character(s) from the entire string, eg. "~." discards any "." from the output string; you may quote the string to be discarded, if it contains other special characters:

'@[varString,~,;,"]' discards the characters ";," and does not read them as field separators;

You may discard quote characters, too:

'@[varString,~']' or '@[varString,~"]'

If necessary, you can use intermediate strings for successive removal operations:

```
VAR Str temp LEN 100 END SET temp "@[varString,~]" PRINT '@[varString,~"]'
```

will remove both quote character types from the output string.

==>> This method of using intermediate variables may be used in many other circumstances, to be remembered!

String trimming:

/"x" Removes the specified character only to the left of the string, otherwise works like ~"x"

\ "x" Removes the specified character only to the right of the string, otherwise works like ~"x"

More examples: '@[event.time;^10;"."]' and '@[amount;+15]"

Global string formatting instructions; must be at the beginning of the format string, followed by a ";" (semicolon) or a "," (comma):

@^ Maximum length in average character width for entire text block

@> Maximum length in average character width, right aligned

Examples:

"@^15;@[title] @[name]" -- together not more than 15 char.

"@>20;@[amount]" -- print in right-aligned field, 20 char wide

Numeric variable with string representation for values 0..n

"@[varName:String A:String B:String C;other format info]"

Numeric variable with string loaded from option list

"@[varName?optionListName;other format info]"

Special:

`@[*variable]` return the length of array instead of returning the contents of the variable.  
`@[?+value?list]` return the string associated with the specified option value from the specified option list  
`@[?-value?list]` return the name associated with the specified option value from the specified option list  
`@[#option?list]` return the value associated with the specified option string from the specified option list  
`@[%variable]` return the contents of the specified environment variable.

### 7.6.2 Parsing of input strings

Interpret performs the inverse operation. It takes a format string and an input string and attempts to extract values from the input, which are then assigned to the variables specified in the format string. The format is similar to the above, except that a more limited range of options is supported. Extended formatting instructions may be included, but will be ignored.

The following meta-characters are supported:

- \* - Means that the preceeding character may be repeated 0 or more times in the input stream
- [ - Accept any of the characters included up to the next bracket ']' NB: [...] and may be combined
- \ - Ignore the following meta-character

Example:

The following format string:

```
Input: @[agenda.date] *+ *@[agenda.priority] => [.;!()]* @[agenda.type]
```

and the following input string: "15.3.2000 + 1 => ...!(.); Meeting with customer!"

Will set the embedded variables as follows:

```
agenda.date    = 15.3.2000
agenda.priority = 1
agenda.type    = Meeting with customer!
```

### 7.6.3 Accelerated translation through pre-compiled templates

The parsing of format strings is fast for normal use, such as in reports, but it may consume significant computational time, if you use it for formatted data output of large data sets. To speed up this process, you can use a pre-compiled Template. Use the `StrFmt.Translator.Compile` method to generate a `StrFmt.Template`, which you may then re-use with the `StrFmt.Translator.Apply` method. The speedup may be anywhere between 50% to 97% (as found in one large application).

## 7.7 Expressions

This module supports the generic parsing and evaluation of arithmetic or Boolean expressions from Parsing.TokenStream sources. The parser creates a data structure representing the expression, which can then be evaluated. Variables may be embedded in the expression as well as constants and application-specific strings, which can be evaluated through a procedure that is supplied by the application.

Expressions can be: arithmetic =  $a + b / ((c - d) * 4) ^ e$

or boolean =  $a \text{ AND } b \text{ OR NOT } (c \text{ XOR } d)$

which is identical to:  $a \& b | \sim (c \text{ XOR } d)$

& = AND, | = OR, ~ = NOT; both representations can be freely mixed

or date-based = date + day

or date-based = date - day

Adding or subtracting numbers from dates increases or decreases the date by the corresponding number of days

or string-based = "abc" + "def" + variable - "cd"

Adding strings is equivalent to concatenating them; this also works with variables, which are transformed into a standard string representation. Subtracting strings means that any sub-string in the left string matching the right string will be removed.

A logical expression can also be a comparison between two expressions:

expression < | <= | = | # | >= | > expression

string\_a IN string\_b

The values for TRUE can be: TRUE, YES, 1

The values for FALSE can be: FALSE, NO, 0

Precedence rules are respected: \* / ^ are evaluated before + -

AND is evaluated before OR and XOR

NOT and - can be used as unary operators

Arithmetic expression syntax:

num\_expr := term [ + | - term ]

term := factor { \* | / | ^ factor }

factor := [ '+' | '-' ] value | ( term )

value := decimal | variable | string | LEN variable | TBLSZ variable

string := any string that converts with Decimals.Decimal.Parse

Precedence rules are respected: \* / ^ come before + and -

LEN returns the length of a variable, typically a string.

TBLSZ returns the number of elements in an array.

Boolean expression syntax:

bool\_expr := term { OR | XOR term }

term := factor { AND factor }

factor := [ NOT ] value | ( bool\_expr ) | compare

compare := expression < | <= | = | # | >= | > expression

value := true | false | variable | command | string | eval

command := CMD typeName:objectName "command"

true := 'TRUE'

false := 'FALSE'

string := any string that converts with Str.ToBoolean

String expression syntax:

`str_expr := string { + | - | IN string | value }`

NB: values other than strings are converted to strings first

NB: adding strings with + means appending them

NB: subtracting strings with - means removing the corresponding sub-string

Date expression syntax:

`date_expr := date [ + | - number ]`

NB: result is date + or - number of days

`variable := 'A'..'Z' { 'A'..'Z', '0'..'9', '_', '! } [ '[' number ']' ]`

`number := '0'..'9' { '0'..'9' }`

`decimal := '0'..'9' { '0'..'9', '.', ',', 'E' }`

`date := DATE dateString`

`expression := arithmetic_expr | bool_expr | date_expr | str_expr`

`eval := EVAL string | string`

NB: when using eval strings without the EVAL keyword, they must not be delimited strings, which are interpreted as part of a string expression instead!

Option string and values:

Option name to numeric value := `OPTION optionListName:optionName`

Option number to string := `OPTION optionListName:optionNumber`

Standard boolean functions for strings include:

`VAREXIST "varName" -- Check if specified variable exists in dictionary`

`EXIST "filename" -- Check if specified file exists, string is translated`

`EVAL "string" -- send string to the EvalProc of the expression, if any`

## 7.8 NumberAsString

This module performs the translation of numbers into fully spelled out strings in various languages. At the moment, it supports English, German and French. Additional converters may be added as classes derived from the Converter class. The application can step through all installed converters to find the appropriate one. Each language is identified by its standard 2 letter identifier (EN=ENGLISH, DE=GERMAN, FR=FRENCH etc.).

Note that it can handle large number, up to  $10^{21}$  or a SEXTILLION.

## Parsing and supporting functions

### 7.9 Hashing

This module implements a generic hashing table. Hashing is a technique for speedy search operations. Each element you wish to store in a hash table is assigned a code. This code is not necessarily unique. It might be calculated as the sum of the ASCII values of all characters in a string that identifies the element. This sum modulo the size of the hashing table is then used as the index into the hashing table, which is a simple array of pointers. Collisions are resolved by searching sequentially through the list of elements starting at the calculated array entry.

Hashing can be very quick, requiring an average of 1.5 access operations, but it is no good for sequential searches, only for direct access to elements with known hash value.

As an extension to the standard hash item, you can also use assignable hash items. These elements may accept assignable LONGINT values, which is great for storing values that are to be retrieved by name, such as constants or variables.

### 7.10 Parsing

Through this module, you may parse extensible syntax languages, such as the *Amadeus-3* object script. Parsing supports the tokenising of any input stream. For this, it will read elements of the same type (numbers, strings, operators) and return them with an appropriate identifier. If a string was read, the parsing procedure will automatically attempt to match this string to known keywords.

All the application needs to do is read in tokens and maybe ask for complementary information, such as parameters for a object or procedure.

The inverse process, the generation of a token stream, is also possible.

An interesting aspect is the fact that the source or destination stream may be in text or binary format, controlled simply by a flag. Binary data is faster to read and slightly more compact, while text data may be viewed and modified with any standard text editor. The main reason to use binary data is to stop users from modifying the data.

## Date and time related

### 7.11 DateOps

Defines the type "Date" and all related functions, including `Current`, `DateToStr`, `StrToDate`, `DaysToDate`, `DateToDays` and `DayOfWeek`. These functions allow your application to input, format dates and to use them in various computations, such as number of days between 2 dates etc.

Many different formatting options are supported, including `Day.Month.Year`, `Year.Month.Day`, `Month.Day.Year`, including the month by partial or full name, including the day name etc.

Of course, all month and day names are defined through variables which you may redefine when needed.

### 7.12 TimeOps

Defines the type `Time` and all related functions, including `Current`, `TimeToStr`, `StrToTime`, `Compare` etc. These functions allow your application to convert and format time and duration values in various ways and to make calculations based on time values.

### 7.13 DateTime

Combines Date and Time into a unit of type UTC – Coordinated Universal Time, which also included the time zone. With tools supplied in this module, you can handle Date and Time as a unit. It is also able to encode and decode UTC values, as well as to format and read UTC strings.

## Universal modules

### 7.14 Common

This module regroups some common constants and other elements, which may be used throughout the library. The first use is to have a system-wide definition for constants, such as “Greater”, “Equal”, “Less” etc. These constants are used in DateOps and TimeOps, Values, Db and many more.

### 7.15 Resource

Resources are not quite what most GUI systems define as such. In the Amadeus-3 context, the usual GUI resources are in fact represented by persistent objects, whereas resources designate items that are stored outside the program and only loaded on request from a file.

The most common resource type is a string that may encode various types of information, under the control of the application.

Under MS Windows, resources are implemented as .INI files. The same format will also be supported under other GUI systems, as it is very readable and easy to maintain.

Example of INI file entry:

```
[Project]
Name=Demo
```

There are 3 ways to access resources in INI files:

- The private resource for the current application; per default, this is automatically initialized to the name of the main project of the application (the application control project) in Startup.WinStart. You may change this name by calling SetPrivateName.
- A named INI file; use GetNamed and SetNamed, specifying the actual INI file name for each call
- The System INI file, which is WIN.INI in the Windows installation directory for all versions of Windows. Use it for installation-specific information only.

#### 7.15.1 Variable Parameters

You may want to have a group of parameters that are specific to a particular installation or use. You may simplify your life by using the calls Resource.GetParam and SetParam, after setting the group name with Resource.SetParamGroup. Inside the INI file, the different parameter groups are identified as, for example:

```
[Param]
[Param.Server]
[Param.Client]
```

Startup.project.LoadParameters automatically scans the command line supplied when starting the program for a parameter identified as “env:”. The following string will be used as parameter group name. To run your application on a server machine, you may hence start the program with “DbApplication env:Server”, telling the application to use those parameters under [Param.Server].

The set of parameters may also be defined via a Licence file, cf. module Licences.

#### 7.15.2 Registry access

For Win32, you'll also find support for the Registry, the big database that MS Invented to keep system parameters in. Registry Crash course:

All registry values are stored as sub-keys to one of the standard keys (cf. below). To access a value, you first need to open it's owner key with either RegOpenKey or RegCreateKey (which either opens or creates a key).

Once you have a key handle, you can either use it to create, open or delete sub-keys or you can read or write values with RegGetValue or RegSetValue. Note that the registry supports multiple data types. For now, only the string types REG\_SZ and REG\_EXPAND\_SZ are supported.

NB: Strings with embedded environment variables to be expanded are expanded automatically.

You should close keys when you don't need them anymore. When you close them, their contents will be written to the registry.

Here are the main registry access functions:

- `RegTopLevel`: Extracts top level key name from string `s` and converts it to key value; return `FALSE` if no match, key invalid
- `RegCreateKey`: Create a new key in the registry
- `RegGetKey`: Get a key from the registry; the key is identified by its name starting from the `mainKey`, which is an already open key; the `mainKey` may be one of the standard keys as noted above
- `RegDelKey`: Delete a key from the system registry; `keyName` specifies the name of a sub key of `mainKey`
- `RegCloseKey`: Close an open registry key
- `RegSetValue`: Set a key value in the registry
- `RegGetValue`: Get a key value from the registry
- `RegDelValue`: Delete a key value in the registry

## 7.16 Commands

Defines command tables that relate strings to project-specific command codes. Each project contains such a command table for buttons, menus, object identification etc.

A command is a `LONGINT` with an associated name; each command name should be unique throughout the application, but numbers are assigned separately within each table.

## Persistent objects and values

### 7.17 Persist

Supplies the basic definition for persistent objects and object groups. Manages the list of registered object classes, derived from `Persist.Object`. The important object classes exported by module Persist are the following:

Class	Description
<b>Object</b>	Abstract class defining a persistent object with <code>Encode</code> and <code>Decode</code> , as well as <code>Name</code> and <code>SetName</code> methods.
<b>ObjectClass</b>	Object class definition. Contains a field for the object "make" procedure. This procedure will be called whenever a new object of the specified class is to be created. In application programs, you may take advantage of this by substituting your own make procedure for any class, creating application-specific objects for generic object classes, such as recognised by the application editor.
<b>TokenStream</b>	An extended version of the basic <code>Parsing.TokenStream</code> class, adding the top object and the <code>useTables</code> field
<b>Group</b>	Defines the concept of the object group, that may contain any persistent objects, including other groups

For a more extensive discussion of the topic of Persistent objects, please see the overview in chapter «2.3 Object Persistent ».

### 7.18 Values

Values supplies the basic facilities to handle input and output of standard data types, such as strings, numbers, dates etc. This module also supplies an actual data dictionary. Complete Oberon-2 data structures can be represented within this dictionary.

The basic module supports the following abstract and concrete types:

Class	Description
<b>Dictionary</b>	Derived from <code>Persist.Group</code> , accepting only <code>DictEntry</code> items. In addition, a <code>Dictionary</code> defines a constant table, that may be referenced by it's members, wherever an item size or array size is required (or anything else appropriate for a given object class)
<b>DictEntry</b>	Generic dictionary entry, abstract class; all following classes are extensions of this class
<b>Struct</b>	Structured variable, i.e. mapping of the Oberon-2 record structure, derived from <code>DictEntry</code>
<b>External</b>	Externally defined, only used as place holder for a structure member that does not need to be further specified
<b>Value</b>	Mapping of a simple variable with internalisation and externalisation methods; abstract class, derived from <code>DictEntry</code>

A structure may contain any other `DictEntry` type, including another structure. A value is a definition of a simple variable of any type, with the restriction that there must be some way to convert this value to and from a string representation. With this property, it becomes possible to implement almost any data input and output or import and export scheme.

The most interesting aspect is that module Values makes it possible to achieve a direct mapping between an internal variable or structure and it's description in terms of `DictEntry` definitions.

So as to be universally usable, the memory area used by a DictEntry element may be assigned either dynamically from the heap or statically, for example in order to map a global or local variable.

Here are the most important procedures and methods from module Values:

### Constant table management:

AddConstant\*(dict: Dictionary; s: ARRAY OF CHAR; VAR cd: INTEGER; val: INTEGER):  
BOOLEAN;

Add a constant to the specified Dictionary.

(dict: Dictionary) Constant\*(cd: INTEGER): INTEGER;

Return the constant that has the specified code. If cd is positive, it will be returned unchanged. If it is negative, the constant of code ABS (cd) will be returned.

(dict: Dictionary) ConstantByName\*(s: ARRAY OF CHAR): INTEGER;

Return the constant with specified name.

(de: DictEntry) ConstantToStr\*(cd: INTEGER; VAR s: ARRAY OF CHAR);

Based on the constant table assigned to the dictionary that owns de, return the name of the constant specified by cd.

### Dictionary and structure management etc.:

(de: DictEntry) GetDictionary\*(VAR dict: Dictionary): BOOLEAN;

Return the dictionary that owns de.

(de: DictEntry) Name\*(VAR s: ARRAY OF CHAR);

Return the full name of de, including records it may belong to in standard Oberon-2 notation.

(de: DictEntry) RemoveOwner\*;

Remove de from its owner, if it is member of a structure.

(dict: Dictionary) Find\*(VAR name: ARRAY OF CHAR; cid: INTEGER; VAR po: Persist.Object;  
VAR owner: Persist.Group; VAR idx: INTEGER): BOOLEAN;

Standard lookup procedure for groups, but taking the special record member naming conventions into account.

(de: DictEntry) DataType\*(VAR tp: SHORTINT);

Return the basic data type to which the object belongs. This is of interest where data must be represented externally, such as in a file or database.

### Memory allocation etc.:

(de: DictEntry) MarkModified\*(on: BOOLEAN);

Mark de as being modified or non-modified. This is mostly used in conjunction with displayed values. When a value has been modified, it should be marked, so that display procedures will be aware of the change.

(st: Struct) Index\*(de: DictEntry): INTEGER;

Returns the index of de within the structure st or NotFound.

(de: DictEntry) Assign\*(adr: LONGINT; dynamic: BOOLEAN);

Assign a data area to be used by the variable defined in de. If dynamic is TRUE, the corresponding area will be allocated dynamically in the buffer field. Otherwise, the area found at adr is supposed to be large enough to hold the complete structure defined by de.

AssignAll\*(dict: Dictionary);

Assign dynamic memory to all elements of dictionary that are not already assigned to a memory area.

AssignByName\*(name: ARRAY OF CHAR; adr: LONGINT; VAR ok: BOOLEAN);

Find a variable by name in the dictionary specified in dataDict (the default dictionary) and assign it to the specified address.

AssignExternal\*(name: ARRAY OF CHAR; adr: LONGINT; sz: INTEGER; VAR ok: BOOLEAN);

Same as for Value, but accept the actual size as additional parameter.

(v: Value) ItemSize\*(): INTEGER;

Size of one element of this value.

(v: Value) ArraySize\*(): INTEGER;

Return the number of array elements in v. Only values may be defined as arrays.

(de: DictEntry) MemSize\*(items: SHORTINT): INTEGER;

Return the size of the data de defines. items specifies the number of array elements that should be considered, where applicable. If you want to know the size of all elements of an array, specify Values.FullSize for items.

(de: DictEntry) Offset\*(): INTEGER;

Return the offset of de from the start of the data structure it belongs to. If de does not belong to a record, this function will return 0.

### Internal buffer access for Value:

(v: Value) ItemToBuffer\*(VAR buffer: ARRAY OF SYSTEM.BYTE; start: INTEGER; idx: SHORTINT; toBuffer: BOOLEAN): BOOLEAN;

Low-level entry point: move data between a buffer and a variable. You may specify: the buffer, which must be large enough to hold the requested data, if toBuffer is TRUE; the starting point within the buffer; the item's index or FullSize if you want to transfer all array elements. This is the proper way for accessing the actual data specified by v.

Get\*(v: Value; i: SHORTINT; VAR data: ARRAY OF SYSTEM.BYTE): BOOLEAN;

Retrieve the contents of specified value and index into data buffer; simplified version of ToBuffer.

Put\*(v: Value; i: SHORTINT; VAR data: ARRAY OF SYSTEM.BYTE): BOOLEAN;

Store the contents of data buffer into specified value and index.

(de: DictEntry) Reset\*(to: SHORTINT);

Reset the contents of de to some known value; the standard, recognized identifiers for to are: ToZero - set all to zero; ToDefault - set to some specified default value.

### Support for external representation of Value elements:

(v: Value) Dimension\*(VAR w,h: INTEGER);

Return the number of columns and lines for v.

(v: Value) MultiLine\*(VAR lineCount: INTEGER);

Return the number of lines, if v may have more than one line in external representation (NB: This is *not* related to the number of array elements).

(v: Value) ToString\*(index,fmt: SHORTINT; fill: CHAR; len: INTEGER; VAR s: ARRAY OF CHAR): BOOLEAN;

Convert a value to external representation. index is the desired array index (may **not** be equal to FullSize); fmt is a general or specific formatting instruction, interpreted by the method's concrete instantiation; fill specifies the character used for filling and alignment; len is the maximum length of the output; s will contain the resulting string.

(v: Value) ToValue\*(VAR s: ARRAY OF CHAR; index: SHORTINT; store: BOOLEAN): BOOLEAN;

Convert a value from a string to internal representation.

### Parsing support:

DecodeConstant\*(ts: Persist.TokenStream; VAR const1: INTEGER; VAR ok: BOOLEAN);

Attempt to decode an actual numeric value or a constant name.

EncodeConstant\*(ts: Persist.TokenStream; de: DictEntry; c: INTEGER);

Encode a number or a constant name as string.

NextIsDictEntry\*(ts: Persist.TokenStream; VAR po: Persist.Object): BOOLEAN;

Attempt to decode the next token as name of a DictEntry currently registered with dataDict, the default dictionary.

## 7.19 StrVal

Defines a sub-class of `Values.Value`, `StrVal.Value`, supporting string values. `Tostring` and `ToValue` operations are basically trivial for this sub-class, but the class does add some intelligence and functionality by allowing the definition of certain string conversion procedures.

You can also define a set of valid characters against which the input should be checked.

## 7.20 NumVal

This module implements a sub-class of `Values.ValueDesc`, to support input/output value handling of numeric values with various magnitudes. You may define a value of this class to represent a `SHORTINT`, an `INTEGER` or a `LONGINT`.

An additional class is defined in this module: `Range`, which may be used to add valid range checking to numeric input. In fact, a single range object may define several valid sub-ranges with `LONGINT`. precision.

## 7.21 RealVal

`RealVal` is a step up from `NumVal` and defines `RealVal.Value`, for real number formatting, input and output support. You may define a set of attributes to specify how the real value should be convert to string representation, by default.

## 7.22 Decimals

Support for Decimals, i.e. numbers with fixed comma and great precision, in the default version up to 24 digits before the comma and 8 digits after the comma.

Decimals support the `FIX` and `STD` format.

Decimal is compatible with the SQL `DECIMAL` type. The `Btr` module automatically converts `Decimals.Value` record elements to `Pervaisve.SQL` format.

## 7.23 DecimalVal

Supplies a `Value` type similar to `RealVal.Value`, but based on `Decimals.Decimal`.

## 7.24 DateVal

This module makes dates accessible as `Value` objects. You may use it to include date variables as data entry fields or database objects and all the other things you may do with `Values`.

Of course, date values may define attributes that define for each value what format should be applied when converting to and from strings. For possible formats, see the module `DateOps`.

## 7.25 TimeVal

This module makes time and duration items usable as `Value` objects. You may use it to include time variables as data entry fields or database objects and all the other things you may do with `Values`.

## 7.26 NumToStr

Combines a virtual value and a numeric variable to provide a field, which may display any string based on the numeric value through the use of a translation procedure. Another method may provide the translation from a string representation into a numeric value.

A typical use would be the translation of a unique id of a database item into a string that represents this item, such as a name. The translation from numeric value to string is usually unproblematic, but the translation from string to numeric value may be ambiguous, such as when a name is not unique.

This kind of ambiguity must be resolved by the translation procedure provided by the application. NumToStr only supplies the framework for such a translation.

## **7.27 DictTools**

Dictionary handling tools. This is a higher-level module, which integrates functions from other modules.

## Project related

### 7.28 Projects

Defines the class `Projects.Project`, from which `Startup.Project` is derived. A project groups a data dictionary and its own constant table, a command table and an object group. Any Amadeus-3 application may use one or more (sub-)projects. Each such project may or may not have a matching definition module, as generated by module `Generate`.

Using individual projects makes code re-use much easier. As User Interface Code may be connected with some interface objects, such as windows, variables and data entry masks, re-using the attached code means also re-using these user interface objects. This is much easier, when you can manage blocks of external objects as units, which is what you can do with the `Projects.Project` class.

### 7.29 Startup

This module declares a class `Project` derived from `Projects.Project`, which is in charge of any application that you write for Amadeus-3. In fact, this is one class that your application must always instantiate, usually in the main module of your application.

Instantiating this class allows your application to modify any default behaviour, up to and including the main loop of the application, but usually rather the `InitInstance` and maybe the `CreateMain` methods.

Your application's `Project` variable *must* be created before any GUI related operation takes place, so that the main loop is properly started. Please refer to the supplied example programs and the Template application to see extended examples of how to implement this. Here is just an outline of the required code:Commands

Defines a command table, i.e. a hashing table containing strings with associated values. You may search for commands either by name or by index.

Commands are used throughout the interface to signal events such as button presses and menu selections.

A project (see `Projects.Project` class) should always contain a command table. This command table may then be used to associate command numbers to interface objects by specifying the command name instead of the number.

The code generator will produce a constant with the command name and the extension "Cmd" for each command when applied to a command table.

### 7.30 EndUp

This module accepts registration of termination procedures. Any module that requires some specific cleanup to take place before program termination (such as closing files, saving data etc.) should register a termination procedure. The registered procedures will be played back in the inverse order of registration on program termination.

### 7.31 PStore

When you wish to save/restore a persistent object to/from a file, it would be tedious if you had to set up all the variables and perform all the required file operations within your application, mostly as this is a very standard situation. `PStore` supplies these operations in an easily accessible format.

**Data structures and sequential access sources****7.32 Sequence**

Defines the concept of a sequential data source without further specifying the actual implementation of this data source.

This is a very useful concept, as you can map any sequential structure (memory lists, database files, arrays) etc. to such a data source. Sequential data sources are used for scrolling, for database filtering and any other place where it may be useful to substitute variable sequential input sources for a common destination.

Sequence also defines a Stepper, which simplifies the use of sequences from within an application. By defining a Stepper, you can access the sequence directly, without having to bother with the necessary infrastructure and temporary variables.

A Sequence may be extended to include tags. Tagging elements allows such functions as selecting multiple lines from a scroll list. To enable tagging, you have to supply the SetTag, GetTag, AnyTagged and ClearTags methods.

Instructions =====

Inherit the Source type and supply the the following methods classes and methods, to match your data structure:

**CLASSES**

- Sequence Defines a set of methods that access some underlying data structure that you wish to access sequentially; it's the central abstract class of this module
- Item Identifies an element of a sequence in a unique way; must allow the retrieval of the underlying data, be comparable etc. You may attach any required information to implementations of this class that are required for such information
- Stepper This is merely a utility, that allows automatic stepping through a sequence

**BASIC METHODS** required for the management of any sequence: -----

- SetInfo Prepares the sequence for a state change; base parameters are NewFilter and Clear. Use Clear to reset the data attached to the sequence. Use NewFilter to set data attached to the sequence which is required for filtering.
- SetStart Is called before invoking Get for the first time with First / Last command.
- Get Should be able to return an index to the first, last, next or previous element of the data structure
- GetNext Utility method for Get; returns the next or previous item within a given sequence; allows more control over the sequence in which elements are retrieved by Get. Your source is responsible for implementing both, Get and GetNext; you should use GetNext to allow external applications to gain control over the access without having to re-implement the entire Get method.
- Load Loads the data pointed to by a given item without changing position information; you can request various sub-sets of data, which is mostly useful when a full load operation is expensive in terms of time or resources.
- Save Stores or fetches the external data buffer for nested use; should be called before the source is accessed, if linked data is to be preserved; a typical use is the building of a scroll window display: Save is called at the beginning and the end of the display building process, so that data is left unaffected by the process.

**OPTIONAL METHODS** -----

- Count Should return the total number of available element, where sensibl and MAX (LONGINT) otherwise.
- Index Should return the absolute number of an item.
- NewItem Used to generate an item that is compatible with the sequence
- Compare Compares two items for equality or rank; must return at least

Command Equal / NotEqual for various other methods to work  
 Pass a command string to the sequence. Syntax:  
 SETINFO NEWFILTER | CLEAR => set filter information

NB: Source.Count should return the total number of items in the current view. IF this number cannot be given, res should be set to MAX (LONGINT) and item index numbers won't be considered. Otherwise, the item index will be used to return the current position.

#### EDITING METHODS -----

Store Opposite of Load; saves data from external buffer to sequential data structure  
 Clear Clears and resets the external data structure to default values

#### TAGGING SUPPORT METHODS -----

Some support for tagging of elements of the source is also supplied through the following methods:

SetTag Set a tag for the specified item (TRUE, FALSE, TOGGLE)  
 GetTag Return the state of the tag for the specified item (TRUE, FALSE)  
 AnyTagged Return 0 (no tags set in sequence), 1 (exactly one tag set) or 2 (more than 1 tag set)  
 ClearTags Reset all tags in sequence.

All these methods are virtual and require a concrete instantiation to work. The most common use for tagging would be the selection with marking of several elements in a list, followed by the application of some function to the tagged elements.

#### STEPPER METHODS -----

The stepper class is intended to simplify the administrative overhead of using a sequence directly from within your code, such as to step through all elements in a given data sequence:

```
-----
VAR src: YourSource; st: Stepper;
NEW (src); src.Init; .. NEW (st); st.Init; st.Assign (src, TRUE); WHILE st.Next () DO ... END;
-----
```

Assign Must be called to assign a sequence to the stepper  
 Reset Reset the stepper, loosing the current position  
 Next Return the next (or first, if just initialised) element in the sequence  
 Again Keep current positioning for following call to Next  
 Command Pass a command string to the stepper. Syntax: (passed on to sequence, if required)  
 FIRST | LAST | NEXT => access data source; read either first, last or following element  
 ASSIGN 'sourceName' => search for source in all open projects and assign it to stepper

### 7.33 MemList

Implements a generic data structure representing a double-chained list. It's easy to abuse this, technically speaking, but it's safe if you stick to a few rules.

Basically, you store records of a given type in such a list and then use the list to manipulate them. This requires no extra fields within your record. Any kind of object may be stored in this way, except for pointers, if you want maintain proper garbage collection. Strings and other variable-length objects may easily be stored in a space-saving way by defining a procedure that will compute the total number of actually used bytes.

This module is inherently *unsafe*, as you may store any kind of information in a given list. In this it is very similar to module Files. But just as for Files, it is possible to make good use of generic memory lists if you stick to a few rules. First of all, you should be very careful with your naming conventions. If you give a sensible name to a memory list, it will be easier to spot any misuse.

Example: if your code first reads

```
topicList: MemList.List;
...
topicList.Add (0, topic);
```

it should strike you as odd if you find a line like:

```
topicList.Add (0, gadget);
```

Though the compiler won't catch this mistake (if it is a mistake), you will easily spot the error when reading your code

The use of memory lists is not recommended. You should usually prefer to build groups (see [Persist.Group](#)) or other pointer list structures for your objects. But [MemList](#) may be just what you need when you are looking for a quick and comfortable way of managing an ordered list of data items.

Command syntax: as [MemList.List](#) is a [Persist.Object](#) class, it implements the [Command](#) method with the following options:

- GET FIRST | LAST | NEXT | PREV variable | record  
Access list and store result in specified variable or record
- COUNT  
Return the number of items in list

### 7.34 ListView

This is an instantiation of a [Sequence.Source](#), with the actual data source being a list in memory. This means it's ideal for access to simple elements, such as numeric variables, strings or records, but not usable for any pointer types (cf. also the chapter on « [MemList](#), 7.33 »).

Access methods are fast and natural for sequential data access.

```
TYPE
  Item*          = POINTER TO ItemDesc;
  ItemDesc*     = RECORD (Sequence.ItemDesc)
    idx*,id*: INTEGER;
  END;
  Source*       = POINTER TO SourceDesc;
  SourceDesc*  = RECORD (Sequence.SourceDesc)
    data- : LONGINT;
    l-    : MemList.List;
    owned-: BOOLEAN;
    PROCEDURE (src: Source)
      Assign*(l: MemList.List; VAR data: ARRAY OF SYSTEM.BYTE);
  END;
```

Other methods do not diverge from basic ones supplied by [Sequence.Source](#) and [Sequence.Item](#).

Special features:

- The field `src.data` points to an array of bytes that contains the data for display
- The field `src.l` points to the actual data source, a memory list
- Both are set through the method `src.Assign`

### 7.35 GrpView

This is an instantiation of a `Sequence.Source`, with the actual data source being a persistent object group.

```

TYPE
  ExchProc*   = PROCEDURE (Source, Persist.Object, BOOLEAN): BOOLEAN;
  Item*       = POINTER TO ItemDesc;
  ItemDesc*   = RECORD (Sequence.ItemDesc)
    po*: Persist.Object; (* item is identified by the object reference *)
  END;
  Source*     = POINTER TO SourceDesc;
  SourceDesc* = RECORD (Sequence.SourceDesc)
    grp-      : Persist.Group; (* Actual group serving as data source *)
    curr-,svCurr-: Persist.Object; (* Current object being accessed *)
    make*     : Persist.MakeProc; (* Make new object *)
    exch*     : ExchProc;
    PROCEDURE (src: Source) Assign*(grp: Persist.Group);
    (* method used to assign an object group as data source *)
  END;

```

Other methods do not diverge from basic ones supplied by `Sequence.Source` and `Sequence.Item`.

Special features:

- The field `grp` points to the actual data source and is assigned through the method `src.Assign`
- While stepping through a group, the field `src.curr` is updated, so that it always points to the currently accessed item

### 7.36 DbView

What `GrpView` is for `Persist.Groups` and `ListView` is for `MemList.List`, `DbView` is for `Db.File`: it supplies a uniform sequential access method to database files. Among others, this allows the display of database elements in scrolling windows. `DbView` defines an object class of type `Sequence.Source`, which is a sequential data source.

### 7.37 SQLView

Supports sequential access to SQL Data sources.

## Stream input and output

### 7.38 Streams

Basic definition of the "Stream" class. This is an abstract class and should not be instantiated. A stream is basically an object to which you can write bytes or from which you may read bytes. No further interpretation is made by the basic methods. Additional methods allow the structuring of the input data, e.g. by viewing the data as text.

`PosStream` is an extended version of `Stream`, defining additional operations for specifying the read/write position.

A stream often is a disk file, but it may be instantiated as any other data structure with similar properties. A typical example of a stream could also be a communication line, where you receive and send basically unstructured bytes and you have no control over positioning or even flow.

Please note that the end-of-line sequence, stored in variable "eol", may be redefined with up to three characters, which should cover all operating standards.

### 7.39 MemStream

This module supplies an implementation of a `Streams.PosStream` based on memory buffers., i.e. arrays of bytes. They can grow or shrink dynamically and support all the methods for sequential and positional access.

## File and Directory access

### 7.40 Files

Files are implementations of `Streams.PosStream`. Remember that many methods and fields are inherited from module `Streams` and may therefore not be visible when inspecting the source code of module "Files". You should therefore either refer to the definition module generated by the browser (Under Extacy / XDS: `xc =browse ModuleName`) or consult the module `Streams` as well as `Files`.

Supports all standard file access operations. Keeps track of open files and is able to temporarily close all files, later reopening them when operating conditions permit.

Closing and Re-opening of files is performed automatically whenever you give up control by invoking a message-retrieval function which may allow another program to gain control of the CPU. This should avoid access conflicts in a badly multitasked environment such as MS-Windows until version '98. It is no longer required for Windows 2000 and later.

### 7.41 Paths

`Paths` implements support for path and file names. This module takes care of all system-specific file naming conventions and conversions.

It supplies an easy way to parse file names and access individual components of a file name.

It also supports directory services, i.e. the creation and deletion of directories.

## Database access

### 7.42 Db

This module implements a generic database interface. Only minimal requirements are imposed on conforming database engines. This minimal set includes: Creating, Opening and Closing a data file, Inserting, Retrieving, Updating and Deleting of records.

It is assumed that there is some way to specify composite keys. Without composite keys, a database would be fairly useless, as one would constantly have to sort through the database elements.

For additional information on database handling under **Amadeus-3**, please refer to the chapter «2.14 Databases» and to the next chapter about module **Btr** (for Btrieve).

### 7.43 Btr

A concrete implementation of a database interface as defined in module **Db**, based on the Btrieve engine, now also known under the name of Pervasive.SQL, which perfectly fills the list of requirements.

Please refer to the Btrieve / Pervasive.SQL documentation for some of the special features. Most of the time, you will not need Btrieve-specific functions, you should get around with the default **Db.methods**, i.e. you should only call and use file methods as well as procedures and constants from module **Db**.

When you use module **Btr** directly, you gain access to additional, Btrieve-specific features, but you lose portability and flexibility. Therefore, you should use module **Db** most of the time, except where access to Btrieve features is required.

### 7.44 SQLDb

Supports ODBC database access. SQL Connections can be opened and when associated A3 databases are opened, they can be automatically synchronized.

SQLDb supplies Connections, Statements and a Manager interface, which tracks open connections. SQL expressions used in statements may contain references to variables in the A3 Framework. Access structures for statements may be created automatically.

## GUI interface modules

### 7.45 Pointer

Access to pointer manipulation functions. The pointer is the mouse cursor, in some GUIs just called "Cursor", which we avoid here to prevent confusion with the text cursor (see next section).

Most of the time, you will only refer to this module when you have to change the current shape of the pointer or when

### 7.46 Cursor

Text cursor manipulations, in particular cursor shape modification and cursor activation and deactivation.

### 7.47 Sounds

Sounds implements some minimal sound feedback procedures. It should at the very least allow your application to generate warning or error beeps. In a more advanced version, you will probably find procedures to play back entire melodies and maybe spoken language.

### 7.48 Graphics

This module is the door to the underlying graphical system. It supplies the most fundamental graphics-related functions, but it has no notion of windows. It does define the type "Context", which may refer to a window or to some other logical graphics entity, such as a virtual page formatted for output on some printer or plotter device.

You may inquire a graphics context for certain important parameters, such as resolution, physical dimensions, aspect ratio etc.

### 7.49 Metrics

This module exports a procedure to access all important system metric information with standard keywords. This is the place to look for information about window frame width, title bar height, screen resolution and other information of this kind.

### 7.50 Colours

Colours supplies definitions for the most important colours and colour related objects.

Inspired by the MS Windows system, it defines pens and brushes, which seems to be a nice way to distinguish the foreground and background colour used for any given drawing function. Pens and brushes will automatically be managed by module colours, i.e. they will be allocated and released where required. Functions to save and restore the active definitions for a given graphics context are supplied.

### 7.51 Menus

This module handles all the basic menu functions, from the definition of menu structures to the use of popup menus

### 7.52 Keys

This module defines constants for important function keys and allows the translation from numeric code to string name and from name to numeric code of any key value.

### 7.53 Events

This module implements a high-level GUI Event translation. All standard GUI events are translated into Amadeus-3 compliant codes and information attached to or derived from an event is directly made available in a standard format, through the record type Event.

## 7.54 Fonts

Fonts are extremely important for the interaction with the user. This module defines a high-level description of fonts, as well as support for a default font. Font mapping is very hard on most systems. Finding a precise mapping for a given font even on 2 different installations running the same GUI system is not obvious. You always have to expect some deviations, so be not too surprised if display or printed output does not always look identical on all machines.

You can improve the output fidelity by distributing specific font files with your application programs, but this is definitely a platform-dependent operation.

Fonts have a special `DecodeStr` method, which allows the quick translation from an encoded string to an actual font and vice-versa, using the `EncodeStr` method. The syntax for such font description strings is the following:

```
"Name;H=h;W=w;X;x;Z;z;#s;O=o;M=m;I;i;U;S;F=f;B=b;C=c;P=p;A=a;"
```

Where: **Name** stands for the font name (platform dependent)

**h** stands for a number representing the height of the font

**w** stands for a number representing the width of the font

**p** may specify either V (for Variable pitch) or F for fixed

**M** may specify either T for Transparent or O for Opaque

**o** defines the orientation

**X** specifies that size is in pixels (default is Points)

**Z** indicates that size represents cell size

**c** defines the colour as one of the standard colour types

**f** defines the foreground colour as RGB value

**b** defines the background colour as RGB value

**I** specifies the Italic attribute

**U** specifies the Underline attribute

**S** specifies the Strikeout attribute

**a** stands for the font weight as THIN=0,LIGHT=1,BOLD=3,BLACK=4

NOTE: Codes without value turn the specified attribute ON if present

The format is case insensitive. The order of all items except the name is free.

Example: « Times Roman;h=12;a=3 » defines a font of class « Times Roman », 12 points high, bold.

## 7.55 CmdCntrl

Implements the object class `CmdCntrl.Control`, which assists a top-level application in managing access to various functions, via manipulation of the accessibility of buttons and menu options. It can among others make a complete copy of a menu and then re-create only those menu options that are actually in use, hiding the others completely.

Via the functions `ToggleAll` and `ToggleInPrj`, it is possible to modify the status of fields, buttons and other display objects.

## 7.56 Dialogs

Some standard dialogs are supported through this module. This helps in giving the user the impression that he is working with a perfectly standard application and allows him to perform certain operations with more ease.

- In particular, there are the
- File selection dialog
- Colour selection dialog
- Confirmation prompts (Yes, No selection)
- « About » information window display and handler
- System tray notification icon handling

and more.

## 7.57 WinMgr

The heart of the Amadeus-3 GUI interface (along with module Events), you won't get anything done if you don't have at least a minimal understanding of this module. This is where the basic classes `Window`, `Object` (abstract class) and `ChildWindow` are defined, along with all the basic methods and procedures to manage windows and window display parts.

Please refer to chapter « 2.5 Windows and display » for an in-depth discussion of the principles behind the GUI interface and window objects etc.

## 7.58 WinEvent

This module extends WinMgr in significant ways. It supplies the default main event loop, handles basic and extended events, manages modal windows and their local event loop, dispatches events to windows and objects, adds window and display object utility functions and controls the display of notes. The latter function may be switched on or off by setting the variable `showNotes`. The timer for displaying a note window is also started by WinEvent. You may obviously use the same timer for other purposes as well, if you just need a timer signal.

## 7.59 WinTools

A collection of useful procedures for window and window object manipulation. When you need to perform some typical window operation, such as setting attributes globally or performing some other function on all objects within a window, you may want to have a look at this module first. You may find what you are looking for.

## 7.60 Scroll

Scrolling is based on module Sequence that defines a generic class Source. A sequential source is a data structure that may be accessed sequentially and that must be able to return at any moment any of the following: The first element, the last element or the successor or predecessor of a given element. This minimalist definition now may be implemented based on an array, a memory list, a database or any other sequential structure.

Module scroll defines a window class that knows how to handle scrolling through sequential elements, but has no idea of how to represent them. This knowledge is carried by the column

elements that may be added to a scrolling window. Each such element is a standard WinMgr.Object display item that simply knows how to draw itself within a graphics context.

Scroll can easily handle any type of element and even variable-height lines without further coding. Based on other Amadeus-3 modules it is very easy to assemble scrolling lists that show icons, bitmaps or simply strings or variables.

Examples: Module DbView implements sequential access to on a database, module GrpView allows scrolling based on a group of persistent elements and module ListView does the same for memory lists.

This combination of modules demonstrates nicely that is not only possible but desirable to work without multiple inheritance. The absence of multiple inheritance forces a better design and ultimately a cleaner library and greater flexibility without the overhead and the intrinsic problems of multiple inheritance.

### 7.61 ScrollDecor

Defines extended scroll background decors. So far:

- Ledger, 2 alternating colours as used in accounting books. The alternate colours may be defined, as well as the use of vertical and horizontal lines as in the standard Grid décor class.

### 7.62 Actions

This module takes care of all the standard object manipulations supported by Amadeus-3. To gain access to these functions, your application must call the function Actions.Handler in one of it's event handlers, usually the DefaultHandler defined in the main code module.

This module knows how to:

- Select objects by clicking them
- Select objects with the « Rubberbanding » method
- Move selected object(s)
- Resize one object
- Drag&drop one or more selected object(s)

For more information, please read the chapter “2.13 Drag & Drop, Amadeus-Style”.

### 7.63 Interactions

A high-level module, which implements scroll line dragging (as used in A3Edit), feedback for long processes and other interactive user interface functions.

## User interface objects

### 7.64 Controls

This is a low-level module, sub-classing all the major GUI-based user-interface widgets such as data entry fields, selections and scrollbars. Only those widgets which are GUI dependent are derived from module Controls. You may define stand-alone widgets, which only depend on low-level Amadeus-3 modules – such as Graphics, Fonts, Colours, etc. – to avoid excessive dependence on the GUI and making the widget portable.

### 7.65 Fields

The standard data entry element of most applications is the field that is linked to a variable of some type. The user can usually directly type text or numbers into such a field. Some « fields » derived from the standard class may have other input mechanisms, but they all share the common denominator, i.e. the fact that the value of the variable they point to may be represented as a string in a natural way. This rules out BLOBs (Binary Large Objects), which cannot efficiently fit into a string representation that could be handled as a simple variable.

To take the two variants of fields into account in the basic design, there are 2 classes with this purpose defined by this module: `Fields.Field` and `Fields.EditField`. The former is an abstract class that defines the relationship between a field and a value and the basic methods used by fields. The latter is the actual implementation of the standard string-entry field (text or numbers).

### 7.66 Toggles

This is a class derived from the standard data entry field `Fields.Field`. As such, it inherits all the standard behaviour of fields, except the string data entry. It replaces this with various forms of checkboxes, which the user may click to turn flags on and off or to make selections from multiple choice lists.

Toggles may be grouped together, so several mutual-exclusive or co-existent choices may be handled as a logical unit. This is achieved through the class `Group`, which has its own particular management methods.

Toggles may be bound to `Toggles.Value`, i.e. Boolean variables or to `NumVal.Values`, i.e. numeric variables and this only if the toggle object belong to a group. The interpretation of the actual value in the case of numeric variables depends on specific factors: It may either return the index number of the selected option or a `SET` (by using the `SYSTEM.VAL` function) with bits set according to the index numbers of co-existent choices.

### 7.67 Buttons

This module manages text- and graphics-based button objects, including their representation (via `Draw3DButton`) and their interaction, with keyboard and mouse interface.

### 7.68 Scrollbar

Scrollbars are another type of user interface widget based on values. Here, only numeric values are allowed, i.e. a Scrollbar *must* be linked to a value of type `NumVal.Value`. Each scrollbar comes with a minimum and maximum value, representing the scrolling range. Whenever the scrollbar is activated, its position will reflect a value between the minimum and maximum, proportionally to its position.

A scrollbar may be either horizontal (by default) or vertical, which is defined with `sb.SetAttribute(Scrollbar.Vertical, TRUE | FALSE)`.

### 7.69 DropList

`DropLists` are derived from `Fields.EditField`. They are linked to a list that contains allowable, previously used or explicitly defined values of the linked variable in the form of

OptionList structures. The methods `DropList.Load` and `DropList.Get` build the OptionList that will be attached to the field.

### 7.70 DbSelect

`DbSelect.Field` is based on `DropList.Field` and is linked to a database file, which is used implicitly to generate the contents of the list of valid strings. For this, it needs to be associated with a `Db.File` and a key, along with a variable that is linked to the `Db.File`.

You can either limit the field function to just the retrieval of existing values or you can allow the user to add new values by setting the `AllowNew` flag.

`DbSelect` is smart enough to deal with certain special cases, such as numeric or Boolean `NoNullKeys` and `NoSegmentNull` segments, that are typically added to a key to limit the records that are included on its search path. When it finds such segments, it sets the corresponding variable to one or `TRUE` before starting the search when checking the validity of a user's entry.

### 7.71 SrcSelect

Derived from `DropList.Field` and `Sequence.Source`, this field retrieves its data from a standard sequential data source (which, hence, can be linked to a database, a list or any other data structure) to assemble the contents a drop list.

### 7.72 TextEdit

At this point, `TextEdit` is a bit over-ambitious, as it just implements a way of formatting and displaying blocks of text as `Object` elements. It does not allow text entry and sophisticated formatting. For now, this function will have to be fulfilled through the use of multi-line `Edit` controls, whereas this module here just supplies a simple way of displaying text blocks.

### 7.73 ValueDsp

This module provides a simple way of displaying the contents of data dictionary values. Whenever you wish to display a variable (global or dynamic) that is associated with a `Values.Value` class object, you can do it through a `ValueDsp.Object` object.

## Graphics- and Bitmap-related modules

### 7.74 Bitmaps

Just as its name indicates, this module interfaces your application with the underlying GUIs bitmap support. The main object class exported is `Bitmaps.Object`, based on `WinMgr.Object` and therefore directly attachable to a window.

If you declare an object as `bm: Bitmaps.Object`, then `bm.name` will be interpreted as the name of an actual bitmap to be loaded. You may specify a complete or relative path name and include or omit the extension. As bitmap loader procedures may be installed dynamically, the extension may give a hint which loader is to be used. The default is `.BMP` (for Windows). If no extension is given, each loader is asked to try loading a file with its own extension.

Further support is given for the manipulation of bitmaps, i.e. for stretching and shrinking a bitmap, for copying it etc.

### 7.75 ImgFiles

This module uses an external library to define loaders for some standard image file formats, such as JPG, TIFF, PCX etc. It works transparently, so you don't have to know anything about the underlying graphics library, you simply use the standard bitmap functions to load or store your bitmap. The `ImgFiles` module will take care of accessing the external library properly.

### 7.76 PCXFiles

Similar to `ImgFiles`, this module interfaces with the libraries of the GX series, for loading and saving PCX bitmap files. It is still included to ensure continuity with previous releases, but probably quite outdated by the time you read this.

### 7.77 Icons

Icons are little pictures used mostly for user-interface objects. Most systems supply special functions for icons which are faster than standard bitmap manipulation functions, as they use the fact that the icon will always have fixed dimensions. The storage format of icons is usually different from standard bitmaps. Therefore, `Amadeus-3` supports special functions for loading and displaying.

In addition, you may create objects of type `Icons.Object`, which are window elements displaying icons. You may attach an entire array of icons to such an element. It may also accept the input focus.

When an element of type `Icons.Object` has the input focus, the active icon may be selected by clicking the mouse on top of this item or by pressing the space bar. This action has the effect of cycling through the available icons, assigning the index number of the selected icon to the associated variable.

### 7.78 DrawObj

This module supports basic graphical objects, i.e. lines, rectangles and ellipses as window objects, including full encoding/decoding, making them suitable for use in an object script. If the owner window specifies the use of 3D encoding, then rectangles and lines will be drawn with a 3D effect.

## Reports and printing

### 7.79 Reports

`Reports` takes the job from where Module Printing left it. This module allow easier generation of reports based on report source files (report scripts). A user-defined support method may be required, to prepare data for printing.

Module `Reports` uses `StrFmt` for the formatting of output strings. This allows the easy inclusion of dictionary entries into report output.

Any display object class may be included for printing, except for those based on GUI controls. It may be necessary to prepare the `Display` method for use with printer output, as the constraints may be different.

Reports are built using the following syntax: (>> leads in new element)

>> A note on string handling: Strings are sent through `StrFmt.Translator.(Full)Translate`, which will convert embedded variables and perform other formatting; cf. each command and module `StrFmt` for details. `Translate` converts the string once, `FullTranslate` converts it repeatedly, until all references to variables have been translated.

```
>> Comment : anything following the "--" sequence will be ignored
>> Font table: FONT defines a font table entry, followed by a number and the font parameters
>> For font parameter strings, please refer to module Fonts.
-- Font entry 0 is the default font; if undefined, it will be set to
-- default .INI file entry RptFont or - if absent - "Arial;h=11"
-- At most MaxFont fonts may be defined (now: 0..9), e.g.
-- Font string may contain embedded variables
FONT 0 "Times New Roman;h=-11;p=v;a=3" -- (default font)
FONT 1 "Arial;h=-11;p=v;a=3"
FONT 5 "Logo;h=18" -- Specialized font for printing company logo
```

To change font definitions dynamically during script execution, use `SETFONT` instead with identical syntax. The font parameter string is translated either during parsing (for `FONT`) or during execution (`SETFONT`).

```
>> Specify page orientation: PORTRAIT | LANDSCAPE
>> Specify duplex printing : SIMPLEX | DUPLEX
>> Specify paper tray : TRAY AUTO | UPPER | LOWER | MIDDLE | MANUAL | ENV
>> Specify paper format : PAPER A4|A5|C4|C5|C6|C65|LTR|LTS|LGL|EXE|E09|E10|E11|E12|E14
>> where: LTR = Letter, LTS = Small Letter, LGL = Legal, EXE = Executive, E09..14 = Envelope 9..14
```

```
>> Internal variables - Definition:
VAR Str name LEN n END
VAR Num name [SHORT|INT|LONG] [LEN n] END
VAR Real name [STD|FIX|SCI|ENG] [LEN n] [DIGITS d] [TICKS] [EXP] END
VAR Decimal name [STD|FIX] [LEN n] [DIGITS d] [TICKS] END
VAR Date name { DMY|MDY|YMD CENTURY MONTHNAME APPENDDAY|PREPENDDAY } END
VAR Time name { SECSINCE AMPM MINUTES SECONDS HUNDREDTHS DURATION } END
VAR Bool name [INT] END
-- Other variable types may be available, depending on installed classes !
>> Internal objects of other types - Definition:
DEF class_name "object_name" { object definition as by it's type } ENDOBJ
NB: the quotes around the object name are required !!!
```

```
>> Internal predefined variables (case-sensitive, as usual!):
RptDate = report date (defaults to current date on call of rpt.Start)
RptTime = report time (defaults to current time on call of rpt.Start)
```

RptName = report name (the file name used to load the report; may not be defined)  
 NbLines = total number of lines per page using FONT 0  
 PageNb = current page number      MaxPages = Highest page number of report  
 Margin = currently active margin (pixels) Column = current column,  
 Footer = Footer margin for page      Copies = Number of copies printed  
 PX = horizontal position in report   PY = vertical position in report  
 MX = max horizontal pixels      MY = max vertical pixels  
 MaxY = max vertical position reached   LineNb = current line number  
 AvgWd = average character width      AvgHg = average character height  
 Answer = string returned from latest call to CMD  
 CmdRes = numeric result returned from latest call to CMD  
 >> All of the system variables can be used in expressions and modified with SET !  
 -- You may add other report-specific variables to rpt.dict

>> Set variable to a string expression  
 >> string is sent through StrFmt.Translate; if it starts with "@" through FullTranslate  
 Assignment (with translate): SET name [ = ] [ "[" index "]" ] Value  
 Assignment (no translate) : STORE name [ "[" index "]" ] Value  
 Increment numeric variable : INC name [ "[" index "]" ] [ Value ]  
 Decrement numeric variable : DEC name [ "[" index "]" ] [ Value ]  
 Value = Number | "string" | Boolean | { Date }  
 >> NB: If the variable is numeric, the string will also be sent through Expressions.EvalToStr  
 >> The following example demonstrates this feature:  
 VAR Real x FIX LEN 16 DIGITS 3 END  
 SET x = "@[operation.rate] \* @[operation.cnt]" -- will be translated  
 or SET x = operation.rate \* operation.cnt -- is interpreted directly  
 PRINT "@[operation.rate] \* @[operation.cnt] = @[x]" -- prints operation and result

>> Define procedure : PROC name ... END  
 >> Call procedure : CALL name | "name" -- call via translated string  
 >> Define header for following code : HEADER ... END  
 >> Define Footer for following code : FOOTER ... END  
 >> Define code for beginning of line: LINESTART ... END  
 >> Define code for end of line : LINEEND ... END  
 >> Define page margin offset : MARGIN [n] | [variable] -- in nb. of characters

To use a footer, you have to include this code:

```
SET Footer [number of lines from bottom] FOOTER [ code for footer ] END
```

>> Boolean expressions      meaning of TRUE:  
 CMD class:object "Command string" -- search for object and pass it cmd string; sets answer variable  
 VAREXIST "varName"      -- Check if specified variable exists in dictionary  
 EXIST "filename"      -- Check if specified file exists, translated  
 ToPrinter      -- Output is being sent to printer object  
 ToFile      -- Output is being sent to file  
 Text      -- Output is being sent to flat text file  
 "FMT=XYZ"      -- Output is file with extension XYZ  
 Duplex      -- Duplex mode is active  
 CanDuplex      -- Duplex mode is available

plus any boolean expression allowed as defined in module Expressions !

>> Repeated execution; repeat following statement n times: REPEAT [n] | [variable] statement

>> Conditional execution (expression cf. above, Boolean expression):

```
IF expression THEN .. { ELSIF expression THEN .. } [ ELSE .. ] END
WHILE expression DO .. END
REPEAT .. UNTIL expression
```

>> Call application subroutine by name

```
CMD object "Command string"          -- search for object and pass it cmd string
PERFORM "string"                    -- "string" is passed to Report.Evaluate
```

>> Output statements:

```
PRINT [Font ID] "string"           -- Print string in specified font after translation
SPACE [n] | ["string"]            -- Leave blank space between items (number of blanks or length of string)
OBJECT [Font ID] "class name;parameters" [x [y]]
-- Insert object of specified class (sub-class of WinMgr.Object); assign string after ";" as
-- object name; x and y designate the extent of the object, i.e. the coordinte of the opposite end
-- If you included the sequence "@!" at the beginning of the object name, the entire string will be
-- translated a first time before being interpreted, which allows for "double-indirection",
-- i.e. you may specify the print format through a report variable
-- You may also include the sequence "@*Command", where Command is the name of a command string,
-- to be activated by the object's Command method
```

```
FIND object_variable "object name" [ "class name" ]
```

```
-- Find an object of specified name [ compatible with class ] and assign it to the object variable
```

```
BREAK [n] -- Page break; if n specified, only if less than n lines remaining
NEWLINE [n] -- Insert [n] line breaks, where line height is measured in FONT 0
NEWLINE -n -- Leave fractional blank space = 1/n * single line height in FONT 0
MOVETO n -- Go to specified column number in average character width in FONT 0
INSERT -- like object, but insert object in page, moving all lower items down by object size
DROP -- Remove latest object from page, shifting following items up by object size
LINE [x] -- Draw a line of thickness "x" (default 1)
INCLUDE "filename" -- Load and include text from specified file as PRINT statements
PARSE "filename" -- Parse code from specified file and include at current point in code
DYNPARSE "filename" -- Parse code from specified file at runtime and execute as procedure
-- allows for the dynamic inclusion of code via a variable name for example
```

>> Column printing:

```
INCOLUMNS n,m -- Start printing in columns of n characters width and up to m columns wide
COLUMNLEN n -- Set maximum number of lines per column before an automatic column change
NEXTCOL [n] -- Go to next column, which is of width n (default same as current width)
ENDCOLUMNS -- End column printing mode (required!)
```

NB: If your report contain large strings, you may want to re-define the variable `maxStrLen`, which is set by default to 600 characters. Very large tables may be split up over several pages, which may require a little additional code, cf. demo code.

## 7.80 DbReport

Provides extension of basic report class, which automatically saves and restores all database files before printing a report page. This means that your application doesn't have to worry about disturbing report data records, when it is printing reports.

The module also adds database-specific commands and variables. The commands can be accessed as usual through the PERFORM and conditional statements.

## 7.81 RptDebug

A utility module that you might include if you want to debug a report. You can use it by calling RptDebug.Evaluate from your main report evaluation method or procedure and then including the line "PERFORM ListRpt" in your report. This will produce a simplified representation of the contents of your report, as it was parsed.

## 7.82 Printing

This is a bare-bones printing support module. It just establishes the link between the graphical context of windows and of printers with the least possible changes to application code.

In printing, you actually draw to a special window type, that is then able to reproduce its contents on a supported printer. Usually, this will happen through the GUI's own support for banding or any similar way of translating graphics into the huge bitmaps needed for printers.

Process of printing:

```
p: Printer; NEW (p); p.Create (); then add elements with p.Add (wp)
when done building page, call p.Update;
```

You may attach another window to a printer window, which will then paint its contents on the printer device. Then either change just some elements or call p.Clear and start over. You may set a document name by calling the Printer.SetTitle method. When done with document, call p.Destroy;

When done with printer, call p.Dispose or just let the finalizer handle it.

Set printer parameters in printer object, e.g. paper, tray & duplex settings

## 7.83 RptTable

RptTable.TableObj Defines Display object, which shows tables with multiple columns and multi-line fields and

with the explicit intent of being included in a report.

The object name defines the column contents and layout (for use in Reports).

Example: OBJECT "Table;@[cntryTbl;#0]##30;@[cntryTbl;#1]##20;Note://@[note]"

- Defines a table with 3 columns, the first of which occupies 30% of the available space (from the current x-coord. on creation of the object to the right hand border of the display area per default), the second one 20 % and the last one the rest (or 50% in this case).
- The ## may be followed by either a number or the name of a numeric variable, which will be interpreted as the % value for the column width
- The variables "cntryTbl [0]", "cntryTbl [1]" and "note" will be inserted.
- The "/" means "insert a line break here", i.e. before the contents of the variable "note".
- To define line attributes, insert "@LINE=[-][n/t/s/m]" after "Table;"
  - - = only horizontal lines
  - | = only vertical lines
  - n = line spacing: where n is the actual line spacing in pixels
  - t = thickness of lines in pixels
  - s = style of lines (0=Null 1=Solid 2=Dash 3=Dot 4=Dash Dot 5=Dash Dot Dot)
  - m = minimum number of lines for block (may be more, if text overflows)
 to be used in pixels. For table with no visible lines, include @LINE=//0
- To define the global background colour, insert "@BG=brush name or @BG=#RGB value"
- To define the background colour for a single cell, insert "@BG:brush name or @BG:#RGB value"
- To specify that a single cell should not be framed, add @NF: (for NoFrame) at the beginning
- To specify a special font for a specific cell, include the string "@Fn:" at the beginning of the cell definition, where "n" stands for the font number you want to use for this column.
- To specify the column alignment, use the standard StrFmt syntax to indicate global alignment
  - at the beginning of the cell, eg. "@^ " for centered, "@> " for right aligned; the space is required, if the next element starts with a digit, to avoid interpreting it as format width
- To include a display object other than a format string, the syntax is @ {ClassName;Parameters}

OutlineObj:

Defines a report object that draws a box around a string, fitting the string with some margin

NB: If your tables contain large strings, you may want to re-define the variable reportStrLen, which is set by default to 512 characters. Very large tables may be split up over several pages, which may require a little additional code, cf. demo code.

## Others

### 7.84 Debug

Debug supplies very simple functions for producing debug trace information in the form of output to a text file. The module is auto-managed, i.e. you don't have to bother about opening, closing or clearing the debug output file.

All output goes to a file named «DEBUG.TXT» in the local directory where the application is being executed. After a program run, you may inspect this file for information your application generated.

Functions supplied are the following:

```

VAR
  f-: Files.File; -- Output file, opened after any call to a debug write function
  name* : Str.PStr;
-- name for debug file, default is "Debug [date time].txt"
  tmStmp*: BOOLEAN; -- If TRUE, date & time stamp the debug file name
  dbgOff*: INTEGER;
-- control over debug output; if dbgOff # 0, no debug output is generated;
-- NB: Use only Suspend / Resume to control dbgOff
  mode* : BOOLEAN;
-- Global mode: TRUE = Debug functions operational; FALSE = deactivated

PROCEDURE PosInfo*(x,y: INTEGER; str: ARRAY OF CHAR);
(* Format « x, y: » coordinate pair and append string str *)

PROCEDURE WriteDate*(d: BaseType.Date);
(* Format and output the date d as string to the debug file *)

PROCEDURE WriteEOL*;
(* Send End-of-Line character(s) to debug file *)

PROCEDURE WriteInt*(l: LONGINT);
(* Format and send SHORTINT/INTEGER/LONGINT to debug file *)

PROCEDURE WriteReal*(r: LONGREAL);
(* Format real value and send to debug file *)

PROCEDURE WriteDecimal*(d: Decimals.Decimal);
(* Format decimal value and send to debug file *)

PROCEDURE WriteStr*(s: ARRAY OF CHAR);
(* Send string s unmodified to debug file *)

PROCEDURE WriteTime*(t: BaseType.Time);
(* Format and output the time t as string to the debug file *)

```

Example of use: You suspect an error in incoming data that arrives under specific timing constraints, precluding the use of a debugger. Instead, you insert Debug.Write statements, to trace the actual data received by the program, so you can compare it to expected values:

```

PROCEDURE ReadData (st: Streams.Stream; VAR cnt: INTEGER);
VAR s: ARRAY 100 OF CHAR;
BEGIN
  WHILE st.err = Streams.OK DO
    st.ReadLine (s); INC (cnt);
    Debug.WriteStr ('Line #'); Debug.WriteInt (cnt);
    Debug.WriteStr (' '); Debug.WriteStr (s); Debug.WriteEOL;
  END; (* while *)
END ReadData;

```

You can then retrieve the resulting listing in DEBUG???.TXT. This file will always be cleared the next time you start the observed application and more output is generated.

Per default, the file name is also date and time stamped, i.e. the ??? is replaced with the date and time it was first created. In this way, multiple instances of the same program don't access the same file and you can maintain multiple execution results. Just remember to remove those debug files when you no longer need them.

You can also set the output file name to something else - even temporarily - and switch forth and back between the different files.

## 7.85 Licence

This module operates in conjunction with the licence editor, LicenceEdit, which you can find in the directory A3\Tools\Licence. Via this module, you can include a slight protection of your application along with user-specific information. The protection is provided by an encrypted Pervasive.SQL data file, which contains information about the user. If the file is absent, your application should refuse to be executed. Furthermore, you should apply the user data, so that it is visible and makes the application unusable for professional users, by including licence-specific information on reports, letters, invoices etc.

Besides for the simple protection, this mechanism can also be used to provide other user-specific information, that is useful in the context of the application. Besides for the basic address data fields, the licence editor can define application-specific fields, which will be transformed into global values after loading the licence information. These values can then be used as any other value, pre-set to the data entered in the licence editor. All values found via the licence module will be members of the record "licence". Please refer to the module text for the details on default fields and usage.

## 7.86 GenCode

Code generation is only meant to create a mapping from dynamic objects to declared variables, types and constants. It will not generate any other executable code. You may create such a mapping module by analysing, then generating a module based on a group of objects.

The following objects will be declared and initialised by the code generator:

- CONSTANTS
  - Command table objects
  - A data dictionary's constant table
  - All key numbers from database files
- RECORDS
  - Based on data dictionary
- VARIABLES
  - `thisProject: Project.Project` variable
  - Variables defined in the project's data dictionary
  - Database files
  - Windows
  - Child windows
  - Menus and PopUp menus
  - Brushes and pens

NB: Windows of the classes Scroll.Window and Scroll.Mask are exported with their own type, not as WinMgr.Window.

## 8. Version Information

In this chapter, you will find information on new versions and how to upgrade existing code to take advantage of the new features.

### 8.1 Adapting existing code to support window layers

Layers are a very powerful new feature of the WinMgr module, but they do require a few changes to your code if you want to take advantage of them. If you do not use layers – except statically through object scripts – you have to make sure your code will know how to find layered objects.

In existing code, to step through all the objects in a window, it was sufficient to write a while loop such as the following:

```
VAR wp: WinMgr.Object;
...
wp := w.parts;
WHILE wp # NIL DO
  (* do some processing on wp *)
  wp := wp.next;
END; (* while wp *)
```

This code still works for the default layer, but if you wish to access objects in other layers, you should use the new methods Window.FirstObject and Window.NextObject:

```
VAR st: WinMgr.Stepper;
...
IF w.FirstObject (TRUE, TRUE, st) THEN
  REPEAT
    (* do some processing on st.wp *)
  UNTIL ~w.NextObject (st);
END; (* if first *)
```

For further information regarding layers, please read the corresponding chapter 2.8 Window layers.

## 8.2 TX3 Syntax change

Due to the change implied by the client-sized window handling etc. some changes in the TX3 syntax became necessary, for clarity and removal of old features, the TX3 syntax has been changed slightly. For use of the version of June 1, 2006 or later, please remove all your AP3 files, apply the following transformations on your TX3 files and then regenerate the AP3 versions:

1. Replace all instances of MKCLIENT; in windows where it is used, insert CLIENT between the first and second group of numbers after POS; as replacement expression for MultiEdit, use:  
 “MKCLIENT {?\*POS [-+0-9]\* , [-+0-9]\* }{[-+0-9]\* ,}” => “#0CLIENT #1”
2. Replace FRAME THINK / THICK DIALOG / TOPWIN / CLIENT / STATIC with THINFRM, THICKFRM, DIALOGFRM, TOPWINFRM, CLIENTFRM, STATICFRM, as search expression:

```
“{FRAME      }{{THIN}}{THICK}}{DIALOG}}{TOPWIN}}{CLIENT}}{STATIC}}”    =>
“#1FRM”
```

Please verify all these changes, they might not have worked for some cases, e.g. where the parameters are on multiple lines.

## 8.3 Major new features overview

2008

- Multiple reports can be opened at the same time; automatic report window copy etc.
- DropLists standardized on SrcSelect
- Revised and improved event handling
- New guideline option inner / outer attachment
- Easy web / email interfacing via run command & run note on object
- New window class TreeView
- Completely revised display context handling with caching of object handles
- Enhanced class management in A3Edit

2007

- Major improvements to DbViewer utility
- Many new features for report generation for much better visual results
- Widow position & guideline saving

2006

- Added support for multiple layers in Windows
- Added A3Edit support for layer editor
- Added editable list of objects for current window
- Added more line-based controls for scroll windows
- Improved focus handling, especially with Tabbed window etc.
- Accelerated display of scroll windows etc.; are now 2 – 3x faster
- Display works fine with Object Desktop xkinning
- Many powerful extensions for report syntax
- Universal complex expression support

A3Edit:

- Improved project tree view and interface
- Changed shortcuts and improved shortcut logic, cf. A3Edit chapter
- Added layer support (Alt-L key)
- Added per window / per layer object list with direct editing access
- Improved guideline editing

2005

- Reports: add complex expressions for tests and assignments => module Expressions.ob2
- A3 programs can now be started from any directory; default files now automatically loaded from application directory; works with resource file, licence file, report files, include files etc.  
⇒ Support via WinTools.DefaultFileName
- If your application loads specific files it assumes are located in the current directory, use WinTools.DefaultFileName to ensure they can also load when running in a different directory
- Added Commands.std standard command table; the standard command codes are now no longer added to each project's table

- Added Persist.ElementsReferenced method to find all reference to other objects
- PStore: SortGroup now works correctly, sorting elements in group by all references

#### A3Edit:

- cf. Pstore above; Elements in project correctly ordered before save
- Separated standard and project-specific command codes in display
- Display standard brushes and pens in list, separately from user-defined ones
- Added column index editing and display for scroll window columns that display variables

#### 2004

- Added ODBC support: Generate standard ODBC definition files; add direct access to databases via ODBC; automatic table and key definition

#### 2003

- Added MAPI interface translation
- Single/Double quotes replacement in Str.Quote added, to avoid use of same symbol as occurring inside the string
- Shift-Tab works also with child scroll windows
- Printing now checks the spooler service, if present, to ensure it is running and otherwise restarts it
- Dropdown list with use of standard Sequence.Source: SrcSelect
- HTML Help support
- Template for dictionary objects
- Improve window attachment and extend to all display objects; tabbed windows should also be resizable
- Align field properly when changing label; don't keep global length
- Keep author name when defining new class modules; store in system INI
- StrVal.Compare with specification of format for compare op. (uppercase, identical, ascii...)
- Copy of scroll source with window ok

#### A3Edit

- Full guideline attachment editing; linking guidelines and scroll columns per mouse click; Scroll Guidelines: attach scroll columns to guidelines; enhanced editing, visualising list of defined guidelines
- Add process revised for objects and Dictionary: list with graphical buttons instead of radio buttons; keep name when using template
- Add creation of List, NumVal.Range etc
- Keyboard interface for project tree
- Save/Restore attributes in A3Edit when copying whole windows
- Edit project group name; used for code generation, allowing for fully user-defined module name in generated code

## 8.4 Change log

10.8.2008

- Fixed error handling in A3Edit; error list is now correctly displayed when an error occurs.
- Fixed loading process; now displays loaded projects correctly even if load was partial.
- Enhanced class management in A3Edit.
  - On all windows
    - Shift+Left click on any window now prompts for the new window class
    - Ctrl+Left click on any object prompts for the new object's class
  - On a scroll windows
    - Ctrl+Left click prompts for the new class of the current scroll column
    - Ctrl+Right click prompts for the class of the associated scroll source.
- Also added new popup menu options for windows and objects to handle the class changes.

1.8.2008

- Added RptTbl.Value, which returns a format string for use with reports that is based on an associated scroll window. Based on the specified index, it returns either a table title, header or data column format. Allows the generation of complete reports based on a scroll window with minimal amount of code; cf. Reports chapter "2.20 Automatic report generation from scroll window"
- Moved Match, BulidMatch and CopyByMatch from Db to DictTools and enhanced the functions with new options.

7.7.2008

- Windows are now sorted in positional order in WinMgr.displayList, i.e. the most recent to be displayed with position TopPos will be first, the latest one displayed with BottomPos will be last; this fixes a bug in window activation
- Improved DateOps.StrToDate to handle multi-lingual strings of very different formats, e.g. "July 4th, 2008", "1er Septembre 2007", "5-11-05", "2006.03.17". For purely numeric formats, you have to indicate the order, i.e. Day / Month / Year, Month / Day / Year or Year / Month / Day.
- Added Str.CaseSensitive as option for convert set; should be used in all string compare operations where convert options are passed
- Added support for case sensitive setting in A3Edit

25.5.2008

- Added A3Edit ImportCSV function wich allows the import of one or more CSV files that contain column names. See corresponding chapter in A3Edit documentatio
- Completely revised GUI Context handling, caching all handles while they are valid, then disposing them to free up correctly all user and graphical device objects; this was necessary to overcome a bug in the internal handle management of Windows. Brush & pen handling has also been completely revised. This should avoid any further GDI handle losses.

5.5.2008

- Added TreeView module to handle tree view extension of Scroll.Window
- Added Shell.Documents, which returns the user or common document folder
- Paths.CheckReplace treated ReplaceAll like ReplaceOlder; fixed

- Fixed refresh in report window based on Constant => should not update constant
- Added Shell.Run and Shell.RunProcess for simplified execution interface
- Added Persist.NoteRunCmd
- Added Commands.Run command constant
- New function WinEvent.RunObj runs the command that is stored in the specified object's NoteRunCmd string; typical use might be to associate an object with a program, a web site or an email address; by clicking the object with ID Commands.Run, the associated command string is automatically translated, then executed; (e.g. a text block, label or bitmap)
- => This latest feature allows the inclusion of active elements such as email and web links in the About window of an application or anywhere else; add a logo or text block with special colour and font pointing to your web site and set the object run note to the web address, make it accessible and set the object ID to Commands.Run
- Added support to attach run note to objects in A3Edit
- Added support for TreeView windows in A3Edit
- Fixed list search behavior in A3Edit (used characters typed anywhere in the interface)
- Added handlers with menus for more interface elements in A3Edit
- Added Fonts.WriteAligned, which writes blocs of text with alignment
- Added Fonts alignment constants
- Fonts.WriteClipped uses WriteAligned with default alignment Left & Top
- ValueDsp now uses Fonts.WriteAligned; changed field align to alignH and alignV to support horizontal and vertical alignment
- Rewrote some routines in Streams to fix ReadLn bug

#### 9.4.2008

- Added options to Str.CountLines procedure to specify if empty or trailing empty lines should be counted as well; is now function, not procedure and result is LONGINT
- Added StrVal.Value.CountLines method with the same semantics as Str.CountLines
- ValueDsp.Display now per default ignores empty lines
- Expressions now supports LINECNT as operator for strings and string variables; returns number of lines in string
- Added Inner / Outer guideline encoding to WinMgr.Window.Encode and Decode
- Slightly changed behavior of WinMgr.CheckCommand
- Improvements and bug fixes in SrcSelect and DropList
- Persist.ParseDecodeCommand now accepts numbers, not only identifiers
- Keys for Scroll.Window editing (Ins = Add, Del = Delete etc.) only accepted if sent to scroll window itself; new broadcasting could lead to keys being broadcast to other windows
- Scroll.Window - fixed line updating after editing
- Added SQLDb.Connection.SourceStr; used in GlobalDb to avoid using connection when connection string is empty after translation
- Fixed WinTools.Dispatcher to always handle current window first

#### 8.3.2008

- Added Guideline option "Inner / Outer frame" (Guidelines.AxisOuter) to specify that objects should attach with their outer, not their inner frame; the "inner frame" is object's position adjusted by the offsets returned by Object.AlignOffset. Example: for data entry fields, this means that the field attaches to the left side of the field label as outer frame, not the left side of the actual field, which used to be the default behaviour (attaching to the inner frame).

21.2.2008

- **Major changes – recreate all AP3 files! Syntax change for all DropList fields.**
- Rewrote Reports & Printing to support multiple reports, removing the infamous "currentReport" variable; same report can now be used directly from memory in different window context
- Added several top-level functions to Reports, simplifying use
- Support for copying report viewer window
- Added module RptObjects, grouping several objects formerly in module Reports
- Completely revised event handling
- Properly handling modal windows on application switching or with multiple top-level windows
- Generally improved handling of multiple top-level windows
- Ctrl+Tab and Ctrl+Shift+Tab now works as by standard for Tabbed windows
- Command Keys (Alt+key) now ignore shortcuts on hidden tabs
- Command Key handling now hasFocus owner window forward from hasFocus object; skips hasFocus object completely; allows for multiple uses of same shortcut; still exercise caution!
- Added Options.Source
- Rewrote DropList as sub-class of SrcSelect
- Fixed various issues in SrcSelect
- Added Scroll.FollowMouse attribute
- Fixed many bugs in A3Edit
- Fixed old bugs in module Scroll; FillDisplay returns result indicating degree of filling
- Occasional crash of A3Edit after standby fixed

04.11.2007

- Scroll Window, column resize: if column is linked to guideline, the guideline is moved along with the column; if the guideline is proportional, then the proportion is automatically adjusted
- Removed WinMgr.PaintBgDel, PaintBgAdd and IsPopup as being useless now
- Added instead WinMgr.UserChg as window attribute, signalling every window that was changed by user
- WinMgr.Window.EncodeGuidelines and DecodeGuidelines now exported as methods
- Added WinTools.UserLayout (save: BOOLEAN) to load or save the position of all windows and scroll window column guidelines that were changed by the user. This allows the user to position windows or resize columns and reload his own configuration when he next starts the application. The data will be saved to the user's application data directory under Amadeus\ApplicationName\UserLayout.apu (application user data).
- Note that this feature is independent of changes in the application data; it will not fail if a window isn't found or if any changes were applied. The only caveats are:
  - the configuration will not work if scroll columns and guidelines were added or deleted
  - if you wanted to force a new window position via the application file, it won't work if the user re-positioned the window; you'll have to force the new position via program code

## 03.10.2007

- Added expression support to StrFmt.Translate format: you may include a conditional in a format string as “@?{condition}:{TRUE section};{FALSE section}”  
The condition will be evaluated and then either the TRUE or the FALSE (if and else) section of the string is executed. Such a conditional section can be included anywhere in the string, even recursively. Note that the “:” denotes the TRUE section, while the “;” denotes the FALSE section. Either of them may be absent.
- Made DbSelect a sub-class of SrcSelect
- StrFmt: @[?+-\* now works with Options.Value variables and string variables; with numeric variables which will be considered as index into following table; with string variables, which will be considered as specifying the name of an option which will be looked up in the table
- New format option Str.ExtFormat can be used to access a type specific format, e.g. Options.Value variables in extended format will return the option name instead of the external string
- Fields.Field has a new field fmt which can be used to format the field representation; before, the option “Trimmed” was used which is now the default
- DropLists associated with option lists and string variables can now store either the external string or the internal name while displaying the external string representation as is done for numeric variables, i.e. the option name is used as an index into the options table.

## A3Edit

- Option lists may now include special characters in the external string representation, e.g. “=” or “[“ and “]” as it is being read right to left
- Support for format specification for fields
  - 15.07.2007
  - Improved SrcSelect to make it fully functional as dropdown replacement
  - Made DbSelect a sub-class of SrcSelect
  - DbSelect now works with a default window, unless a specific selection window is attached; the default window will list the basic select field, allowing easy replacement of all standard dropdown lists linked to database files

## 25.04.2007

- Added standard command line flag +logo to control the display of the application logo window
- Added the following flags to A3Edit (in addition to +logo):
  - +tx3: load only TX3 files, ignore AP3 files when reading a project tree
  - +save: store all projects; if +tx3 was specified, store only to binary AP3
  - +genCode: generate code for each project
  - +backup: create backup files of TX3 files
 These options allow the use of A3Edit as a “compiler”; when either +save or +genCode are specified, the program will quit immediately after saving the requested files.

## 05.04.2007

- Added support for regional information via module Common for region data and Startup for loading fields from OS & adding interface variables:  
Description                      Windows      Amadeus Common.      In dictionary

```

-----
Country name      : sCountry  region.cntry    osRegion.cntry
Country id number : iCountry  region.cntryId  osRegion.cntryId
Currency code     : sCurrency region.currency osRegion.currency
Default language  : sLanguage region.language osRegion.language
List separator    : sList     region.listSep  osRegion.listSep
Date separator    : sDate     region.dateSep  osRegion.dateSep
Time separator    : sTime     region.timeSep  osRegion.timeSep
Time AM           : s1159    region.am        osRegion.am
Time PM           : s2359    region.pm        osRegion.pm
Decimal separator : sDecimal  region.dot       osRegion.decimal
Thousands tick mark: sThousand region.tickMark osRegion.tickMark

```

- Reports export now converts the format string with `StrFmt.Translate`, which means that you can use regional variables (among others), e.g. for the CSV export, you can specify:  
`CSV=(!)(!)(!)(!)"(!)"(!)[osRegion.listSep](!)[osRegion.listSep](!)1(!)1`  
and it will use the list separator as defined in windows for the current installation. This will allow users to export to CSV and import into Excel on the same system without any problem, despite the Excel bug that doesn't allow it to recognize a specific character, e.g. the comma, as defined in the CSV standard...
- ATTENTION: As a consequence, the following substitutions must be applied in this order:
  - `DateOps.separator` with `Common.region.dateSep [0]`
  - `TimeOps.minuteChar` and `secondChar` with `Common.region.timeSep [0]`
  - `Convert.dot2` with `Common.region.decimalAlt [0]`
  - `Convert.dot` with `Common.region.decimalSep [0]`
  - `Convert.tickMark` with `Common.region.tickMark [0]`

Note that these character variables are now represented as 2 character strings to be useable as interface variables.

### 16.03.2007

- Added copy function to `DbViewer`; it allows you to copy records from one open file to another; the function matches record fields of equal names recursively, but it ignores the top level record name. If 2 fields bear equal names, they will be considered to be the same field independently of field type.
- Added support for editing user definition of object from within `A3Edit` ; just do a Shift-Click left on the desired object (including variables etc.) and enter different note fields, i.e. Help topic, Bubble info, User Comment or programming comment.

### 11.03.2007

- Much improved `DbViewer` utility; now allows opening multiple files; supports array objects for editing
- Added brief documentation for `DbViewer` utility.
- `Options.Value` now may be of any integer number type (`SHORTINT`, `INTEGER`, `LONGINT`); in `A3Edit`, pick a numeric data type and to transform into Option variable, simply assign the matching option list on tab page for numbers.
- When dropping an Option related numerical variable onto a window, it will automatically be transformed into an Option dropdown field.
- Added option "TypedAnswer" and function `ConfirmTyped` to module `Dialogs`; requires the user to type his answer ("Yes", "Accept" etc. depending on the text of the confirmation button) in addition to pressing the confirmation button..

## 24.02.2007

- Added Options.Value, which is derived from NumVal.Value. Allows the creation of variables that are directly associated with Options.List.
- Where option variables are displayed, the external name is displayed for all formats except FlatFormat, for which the internal name is used. ToValue (e.g. with data entry fields), accepts either an external option description (language dependent), an internal name or a numeric value.
- NB: That means that options may be entered and specified as either numbers or strings, both will be accepted anywhere
- Added integration of Options.Value into A3Edit; option values are considered as short numeric variables, but they can also be assigned an option list name.
- Added support for Options.Value in Db.ob2: option value list is exported to DFI file; changes in option list are recognized and integrated into database update, i.e. if an option name is assigned a different value, the file will be updated automatically
- Added support in DbViewer: option variables are correctly displayed and the data entry mask generates a dropdown list for each option with an associated value list.
- Improved error reporting in Db.File.Open

## 29.01.2007

- Added DateOps.InRange
- Fixed bugs in A3Edit: clearing project will also delete all open windows for child projects; deleting “.” entry while viewing a record won’t crash anymore
- Improved Report tables some more; new features; also fixed dimension when using other than standard fonts, including multiple fonts in different cells

## 15.01.2007

- Added TX3 keyword NODISABLE to Controls.ob2; also added support to A3Edit. Use NoDisable attribute to deactivate control object without putting it into the Windows disabled state.
- Bug fix in A3Edit: when editing parameters of objects in Windows with “Append” attribute, the sequential position of the object is no longer changed

## 29.12.2006

- Added Str.ReplaceNewline, which replaces any newline sequence (eol, cr or lf) with the specified replacement string
- Added StrFmt format option for replacement of newline sequences in variables; format is: @[variable;{replacement string}], where replacementString may be quoted or unquoted; if it contains separator characters - i.e. any of , ; ] – then it must be quoted (single or double quotes)
- Revised module Reports; removed some old features (BLOCK, FILL) and unused tokens and keywords, added new features (INSERT, DROP); cf. report documentation for use (quite powerful)
- New command codes for MemList.List: GET, COUNT, INDEX, APPEND, INSERT, DELETE for control over list object from within reports etc.
- Quite a few more object commands were added over time; check out the appendix 11.3 Object Commands”

## 14.12.2006

- Added Shell.Folder command to retrieve system folder names such as “Desktop”, “Startup” etc., from registry; “%USERPROFILE%” is automatically expanded
- Improved Reports.currentRpt handling; global variable now only used during generation of each page; otherwise is set to report currently viewed in report window; should avoid the funny font display if there is a dialog during a printing process
- Reports now allow up to 127 font definitions; table expands dynamically

## 21.11.2006

- Added MemList.Command method with support for GET and COUNT
- Paths no longer imports MemList; instead now supports an internal list structure for CopyDir and KillDir

## 15.11.2006

- Added report variables Copies (controls nb. of copies), AvgWd (average character width), AvgHg (average character height)

## 09.11.2006

- Added support for duplex printers in Printing.ob2 and Reports.ob2
- New commands include DUPLEX, SIMPLEX and test CanDuplex and Duplex
- Added support for full-page “header”, i.e. for carrying over printed objects to following pages if a header takes up too much space; you can use this to print backside pages on Duplex printers as “header” code
- Improved Guideline editing; also fixed 2 bugs
- Improved handling of filters with DbView.Source; substantially accelerated speed, making the use of filters feasible with very large data sets without performance hit

## 05.09.2006

- Added background colouring to report tables, module RptTbl.ob2

## 27.07.2006

- Btr.File.Open no longer resets the keyNow parameter to zero, Init does
- Added A3Edit layer editor
- Added A3Edit object list editor
- Changed WinMgr.Layer.SetAttribute to WinMgr.Window.LayerAttribute; method now updates object visibility immediately

## 30.06.2006

- Added Values.IsNumeric
- Added support for deferred window placement via Graphics.StartDeferred, SetPosWindow, EndDeferred; replace all calls to Win.SetWindowPos with Graphics.SetWindowPos to use this features; not properly working yet, comments on Windows operation welcome.

## 16.06.2006

- Revised the window size calculation to support Object Desktop, which can radically alter the proportions of frame elements and scroll bars

- Accelerated display functions by eliminating many refresh operations and improving the handling of resizing sets of windows with guidelines; especially apparent for scroll windows
- **IMPORTANT:** New window size is based on either the outer frame size or – if specified or for all child windows– calculated based on the client area size; this has implications for windows that are repositioned by the code: if you change the outer dimensions, you should always set `client.wx` to zero before creating the window or – alternatively – first create the window and then change it's size
- The new method `WinMgr.Window.CalcExtent` has been added to calculate a windows size and return the full set of GUI attributes for it's creation
- **TX3 Syntax change:** the attribute `MKCLIENT` is no longer used. For better readability, dropped the `FRAME` token and instead added the suffix `FRM` to each frame type name; do a global replace of  
`“FRAME {[A-Z]*} “` with `“#0FRM “`  
then reload all the TX3 files with A3Edit and regenerate the AP3 files
- Added support for proportional scroll bars; added `ScrollInfo.pg` for page size
- Added `WinMgr.WinToTop`, `ClearScrollBar`, `PlaceChildWindows`
- Switched a few window procedures and declarations around:
  - from `WinMgr` to `Graphics`: `PosInfo`, `RectInfo`, `ExtentInfo`
  - from `WinEvent` to `WinMgr`: `MoveGuidelines`, `PlaceGuidlines`, `,`, `IsParent`
- Scroll bars are not added at all when the `VScrollable` and `HScrollable` options are not specified; if they are present, the attribute `IsToolwindow` is ignored as scroll bars are not always represented correctly in tool windows
- Improved Reports: when adding objects it is no longer necessary to insert `BREAK n` statements in the report script to avoid page overruns (normally calculated after `NEWLINE`); when objects extend beyond the page length they are automatically moved to the next page
- Added support for alternate sorting tables for Pervasive files
- Added `Options.List.InsertOption`
- Added `Projects.ForEach`
- Improved font handling, particularly for multi-line text fields;
- `WinTools.DefUserFont` now properly refreshes all windows, including scroll lists; added `WinTools.ApplyDisplayChange`, which can also be used for changes of other global properties

## 01.06.2006

- Added `Str.Contains`, `Str.IsAlphaNumeric`; `Str.IsLetter` now also accepts `'_'`
- Fixed problem with client-sized windows; the attribute `MkClientSize` is no longer necessary; the client size is now recorded for every window, but optional; if specified, it is used to generate the initial window size, properly calculated based on the method provided internally by `Windows`.
- **WARNING:** Changed TX3 Syntax; please see version instructions above for adaptation to change.
- **WARNING:** Changed `Str.AppendEol` parameter `nonEmpty` to `onEmpty`, to make it consistent with `AppendLine`; they were different for historical reasons; invert the value of the boolean parameter on every call to this procedure! To make sure you don't miss it, the name was changed to `AppendEoL`
- Support for mouse wheel in report viewer
- Added `ImgFiles.Rotate`
- Report variables are now automatically initialized `ToZero`, so you don't need to clear them after declaration

## A3Edit

- Added window layer and object display via icon or Alt-L
- Improved handling of all shortcuts

## 01.05.2006

- Added Guideline center option, i.e. objects can be centered on a guideline; to specify, attach object left/top border to guideline and right/bottom border to “Center on axis”
- Dynamically allocated Values.DictEntry objects are now automatically initialized to zero (unless they are being copied), which means among others that you don't need to CLEAR variables allocated in reports anymore.

## 23.04.2006

- **MAJOR NEW FEATURE:** Added layers to window maangement; cf. documentation
- Improved TAB handling for moving through objects inside and between windows
- Added multi-lingual support to DateOps, used in StrToDate
- Added TextEdit.Translate to translate all lines in contents via StrFmt.Translate
- Added WinTools.FindObjOfClass to search for object of specific classId
- Moved most Guideline definitions and procedures out of WinMgr to module Guidelines
- Actions.ob2: When moving / resizing objects, guidelines are now enforced permanently
- Actions.ob2: When drag/copy action fails or is aborted, the copied objects are not returned to owner window
- WinEvent.FindChild now returns WinMgr.Stepper instead of WinMgr.Object, providing also the layer information after a search.
- Added Scroll.LineItem.Paint & Background, Scroll.Window.NewLine for better control over line items; as soon as lines become invisible, LineItem.Dispose is called, which takes care of any external objects, such as bitmaps or windows, which means that now even windows can become scroll items.
- Added Scroll.DisposeNext and Scroll.Window.WipeLines; take care of disposing external line item contents so that you can use bitmaps, child windows etc. as column items and immediately recover the corresponding resources as you scroll through a large list.
- Added WinMgr.Window.AccessOk to check and signal access to a window
- Improved documentation, integrated version information and change log

## A3Edit:

- Renamed module Guidelines to GuidelineEdit to avoid conflict
- Improved project tree view; now auto-tracks current project in smaller, fixed position windows
- Improved font editing; refresh of bold / italic attributes didn't work properly
- Improved keyboard control, new shortcuts and logic, cf. A3Edit chapter

## 20.03.2006

- Added SETFONT command to reports, allowing dynamic font changes during report execution
- A3Edit: Fixed segment adding in database editing
- Added SMTP/POP support via LeadTools library LTEmail, included in A3Lib; for commercial use, you must get a licence from LeadTools.

- SQLDb: fixed definition of decimal fields where digits were zero length; in debug mode, table create statements are now always written to debug file
- Licence: when an environment is specified by command line or in any other way taking precedence over the licence file, the current environment name will be stored in the licence.env variable.
- RptTbl: The ## in a column definition may be followed by either a number or the name of a numeric variable, which will be interpreted as the % value for the column width
- WinTools: FocusBased dispatcher improved to recognize the selected window with more precision
- Scroll: Introduced the possibility of defining a fixed line height, accelerating the display refresh as there is no need to find the line height each time; added FIXEDHI syntax element for scroll window definition
- A3Edit: Added option on scroll window definition mask to support fixed line height

### 22.02.2006

- Added Reports.ExportToStream and improved export functions; all export resource strings should now be found in section [Export]
- Fixed A3Edit bug that could crash on guideline editing after saving an application
- Added OPTION handling to Expressions:
  - OPTION optionList:optionName [returns number]
  - OPTION optionList:optionNumber [returns string]
  - OPTION optionList:@variable [returns string or number]

### 19.01.2006

- Added Dialogs tray icon notification handling via Notify and RemoveNotify
- WinMgr.Window.AcquireFocus now forces the application window into the foreground, if it is not the currently active window
- Typical usage: when a notification icon is clicked, the application window will be brought to the foreground

### 16.01.2006

- Added Db.DataErr error level
- Added ExtDebug.ob2 to avoid cyclical imports with DateTime.ob2; will contain other high-level debug functions
- In A3Edit, Ctrl+Shift+Click&Drag now ignores window objects and allows direct manipulation of the owner window
- Fixed menu editing, display problems
- Expressions LEN now returns the length of the current string for string variables, instead of the maximum item length
- Fixed report parsing error on TRAY UPPER/LOWER, when no quotes used ("UPPER" and "LOWER" worked fine)
- Fixed access to shortcuts when using Alt+Letter from within drop list fields and certain other standard controls
- Decimals.AssignStr now also works if Convert.tickMark is empty

20.12.2005

- Added column index editing and display for scroll window columns that display array variables

15.12.2005

- PStore.SortGroup now fully sorts group of objects by reference or returns error if circular references found
- Added Persist.Object.ElementsReferenced to support full sorting
- Added Values.Struct.ElementsReferenced to support full sorting for Values structures
- A3Edit stores correctly ordered data for object and dictionary lists
- Much improved brush and pen display and editing; standard brushes and pens now displayed in separate list
- Added HTML-like named colours with pens and brushes
- Refresh ok on adding brushes, pens, windows, on dragging & dropping windows etc.
- Scroll revised and improved

11.12.2005

- Further improvement to A3Edit: better menu editing; focus on drop now passed to dropped object
- Added several chapters and updates to documentation

09.12.2005

- Added Commands.std standard command table; the standard command codes are now no longer added to each project's table
- Command table base is now LONGINT, allowing for many more and larger tables; set multiplier to 10'000, for up to that many non-conflicting commands per table; the INTEGER was really a relic from Win3.x, which allowed only 16 bit command codes
- **ATTENTION: Generate OB2 files for all AP3/TX3 with the new editor**
- Separated standard and project-specific command codes in A3Edit display
- A3Edit bug changing child window attributes now really fixed

07.12.2005

- A few more improvements in A3Edit; long droplists are now editable; when editing window objects, guidelines are preserved
- Mouse actions: when pressing Shift & Control, any window object is ignored and the window can be manipulated directly (i.g. you can copy or move the window even through an area that is covered by another object)
- Changed definition of Str.IsDelimiter; now any character that is not alphanumeric or "\_" is considered a delimiter; if you need the old definition, you are advised to create your own function and replace the calls in your code

03.12.2005

- Fixed Reports.Footer handling; now works perfectly, even with complex footers.
- Moved stepping direction codes from Sequence and MemList to Common.
- Added recursive parameter to WinMgr.Window.Refresh and WinMgr.Object.Refresh; per default, replace with WinMgr.All, unless you know that refresh should not operate recursively; this parameter specifies the depth of the recursion; zero means no recursion.

- Fixed child window attribute handling in A3Edit and improved record field display; also fixed colour change while moving objects in list.
- Corrected ListView handling; now faultless.
- Added Scroll.WipeLines; use when filter and other settings for list did not change to clear current display.
- Added Graphics.CreateWindow, for consistency only.

### 01.11.2005 - Major new release

- Added Values.Value.CompareVal to compare 2 values directly without string conversion.
- Moved Decimals.Value object to module DecimalVal.ob2, along with all attending constants, types, variables and methods.
- New module Expressions handles mathematical, boolean, string and date expressions generically, with embedded variables etc.
- Module reports uses module Expressions for complex conditions and math expressions inside report scripts. ';' is now accepted as generic terminator inside a report script.
- Required changes to existing report scripts:
  - Make sure that conditional statements (IF/WHILE etc.) with a single string as condition specify either 'EVAL "string"' or leave the string without quotes. Quoted strings are now considered to start a string expression.
  - Replace assignments and checks on boolean variables where you used numeric expressions with boolean ones, e.g. 'SET quit 1' should become "SET quit=TRUE;" (the '=' and ';' are optional) and 'IF quit = "1" THEN' simply becomes 'IF quit THEN'.
  - Where an expression is followed by a comment ('-- text'), insert a ';' after the expression.
  - Replace assignments to numeric variables via strings and string expressions with proper expressions: 'SET total "@[total]+@[increase]/2"' should be transformed into 'SET total=total+increase/2;'.
  - The INC and DEC command now accept an optional ',' separator between the variable and the expression, which by default is still '1'.
- CMD Syntax in reports was changed: the type is now separated from the object name with a ':' and not with '.'.
  - Example: old 'CMD DbFile.customers "GET FIRST"' changes to 'CMD DbFile:customers "GET FIRST"'. Benefit: structured elements can be included without class name.
- Parsing.NextIsIdentifier now accepts a second separator; per default, set to 0X.
- Added Projects.FindThroughObj, errHnd Errorhandler procedure variable.
- Added Parsing.ErrorHandler type (replacing Persist.ErrorHandler).
- Added Parsing.history and improved error reporting for parsing. If input is a file, Parsing.Signal now automatically creates an error file with the same name as the input file, including extension, plus extension ".err" (i.e. for include files, the error will be written to the name of the include file + .err, not to a global error file). When errors are signaled, the parsing history is written to the error file, along with the error line, position and error number/message.

### 15.10.2005

- DateOps: one of the attributes DayOnly, MonthOnly and YearOnly may now be combined with DayMonthYear, MonthDayYear and YearMonthDay attributes; if StrToDate cannot find a complete date, it will accept just a day, a month or a year, according to the specified

attributes; the output will be either the full date or - if not available - the partial element. Typically use: birth dates, where the full date may not be known, but the year of birth is available.

15.09.2005

- Fixed bug in that could alter windows flags when editing certain fields
- Finally removed the old window attachment system; if you still used it, now is the moment to move on to Guidelines!
- Added Str.WordFromPos and added parameters to ExtractWord

30.08.2005

- Many improvements in A3Edit
- Added Startup.info.cmd, .cmdDir and .startDir
- Added WinTools.DefaultFileName, which checks if a given file exists locally or changes the name so that it points to a directory relative to where the executable is located
- The default resource file, the licence file, report and include files etc. are automatically located in this way, so that you can start applications from any directory

11.07.2005

- Changed parameter line handling: the cmdLine variable no longer exists, access all parameters via CmdParam and CmdFlag.
- The empty ID parameter name ignores any preceding parameters and just returns the portion of the command line that does not contain any identifies and flags, not the entire command line
- Improved font handling in A3Edit
- The mouse wheel now only returns VScroll for Scroll windows; elsewhere it can be accessed as event type "Mouse", button "Wheel" and action = number of increments
- Db.Manager now contains the name of the underlying database, e.g. Btr.mgr.name = "Pervasive"
- If the file DbErrors.ini is included with the application (found in A3 directory), database errors can be displayed in full; DbErrors.ini should contain messages for each installed db manager

24.04.2005

- Added Db.Database.defDir, which is required for applications with multiple databases directories and a default directory; the default name will be used to form the full path of the default database directory

27.03.2005

- StrFmt.Translate now accepts @[variableName] to return the contents of a system / environment variable, such as PATH
- Update procedure change: the name of the record in DFI files is now ignored, only the data structure is checked for compatibility; this allows you to rename a record without having to worry about remaining compatible with existing DFI files; you may obviously still have to rename your data and dfi files
- Various improvements in A3Edit, among which:

- Shift+Left click on object and dictionary lists now pops up the Note dialog, for annotation and documentation of the object; same as right-click on object in edit mode and menu selection "Set Help / Note"
- In project tree view, you can use the first character of an application name to jump to that tree branch; incremental, i.e. repeatedly pressing the character key skips on to the next application starting with the same letter; Esc closes window
- Ctrl+C, Ctrl+X, Ctrl+P and the Copy, Cut, Paste buttons finally work as they should; even empty windows can now be selected and activated

### 03.03.2005

- Added Paths.UpperDir
- Added SQLDb.connection.base; store the connection base directory here when assigning the source
- Added SQLDb.RelativePath, which makes the file name of database files relative to the main directory, where possible
- Added SQLDb.Environment, which only connects to an ODBC environment, without opening a database connection
- Added SQLDb.Disconnect, which only disconnects the database, but keeps the environment open
- Added SQLDb.DataSources, which fills a list with available data sources and driver information
- Changed Db.File.RecordCount and KeyCount to function, instead of returning VAR parameter

### 26.02.2005

- Added prefUser (prefer user settings) parameter to Licence.GetSetting and Licence.Setup
- Added Scroll.Window.DeleteLine method, which deletes the selected line and updates the display simultaneously
- Fixed ListView to properly handle line deletion
- Added possibility of declaring a Values.Struct to be based on another record via baseTp field; fields of this record will not be declared in generated code, but structure should match base record
- Added Menus.PopUp.Get method; same as Prompt, except that it is modal and directly returns the selected command
- Menus.PopUp.align field is now a SET, as it allows the combined use of parameters, e.g. AlignLeft + CenterVert

### 30.01.2005

- Added module MemStream.ob2
- Added module GroupedList.ob2
- Added functions Decimals.Parse and Extract, to support parsing of decimal numbers in strings

### 06.01.2005

- Added additional functions to Decimals.ob2, to support conversion to and from LONGINT and LONGREAL
- Fixed a few small bugs in DbViewer (scrolling etc.), added editing of multi-line fields; updated dictionary viewer to new format

- Fixed bug in ScrollDecor; didn't display correctly in horizontally scrolled windows
- Pervasive file names are now converted to the same spelling as dfi files on creation, i.e. including lowercase letters; Pervasive per default transforms all names to all uppercase
- DbView source now accepts a StrFmt string as starting and compare value for filters
- Added showZero option to RealVal.Value and Decimals.Value, same as for NumVal.Value
- Added Debug.WriteDecimal

#### 06.12.2004

- Added Decimal data type and value for use in data entry fields etc. through module Decimals.ob2
- Various A3Edit improvements, including support for Decimal type, better colour definition, foreground and background colour editing for fields (instead of just drag & drop support)
- Decimal type is correctly declared when used with SQL / ODBC
- Conversion from internal to standard ODBC format is performed automatically when accessing decimals in Pervasive database

#### 17.11.2004

- WinMgr.SetFocusProc not expects a Window as first parameter; WinEvent.SetFocus has been changed accordingly; the object parameter may now be NIL, too. In this way, the focus can be passed to a window with no objects.
- SQLDb now has constants and methods added for access via scrollable cursor
- SQLSrc is in production and will provide sequence access to an SQL/ODBC data source, for use with Scroll windows etc.
- Added support for ProgressBar control via ProgressBar.ob2 with support in A3Edit
- Added support for TrackBar control via TrackBar.ob2 with support in A3Edit
- For both, ProgressBar and TrackBar, the object is created as transformation of a numeric field - Right-click field and select "Transform to" sub-menu.

#### 12.11.2004

- Class redefinition in A3Edit object list now allows for "downgrading" classes, i.e. you can choose to change an object into it's parent class
- The use of constants for the definition of External classes could provoke problems with databases; the following changes were introduced to fix this and gain more consistency overall:
- In Values.ob2: ConstValue is now associated with DictEntry, not Value; added method DictEntry.ArraySize, long overdue; defined ArraySize also for class External

#### 22.08.2004

- Added styles for pen; outline pen is now thin dotted line, as it should have been...
- Added support for mouse wheel; for now, it just scrolls one line up and down, when activated; might add page scrolling

#### 27.06.2004

- New module ComPort for serial com port control added

#### 08.06.2004

- Added Persist.Group.CreateLink, to allow the creation of group-specific links

## 03.06.2004

- Added DateOps.NoAutoComplete
- Fixed focus change bug, not reactivating scroll window when switching back from other application
- Added parameter noAutoComplete to DateOps.StrToDate; as a rule, for interactive use this should be FALSE, for string transformation, it should be TRUE; TRUE means that an incomplete or erroneous date string will not be completed via the date returned from DateOps.Default
- Events.Translate no longer transforms the value of Ctrl-key values: instead of returning `ev.key = 1` for Ctrl-A, it returns `ev.key = ORD ('A')`; check if `Events.Controls IN ev.state` to determine if Ctrl key was pressed and remove the `ORD ('@')` bias

## 31.05.2004

- Fixed problem with Tab use for multiple child windows in A3Edit

## 17.05.2004

- Paths.Parse and other Path functions now recognize the file syntax "\\\" to indicate a network file, so you don't need to declare a share before you can use network files
- A3Edit: Improved project tree view and other functions

## 10.05.2004

- Changed project loading: Startup.project is now reserved for program-generated objects
- The load order of projects in Project.chain was inverted, which should not affect your code
- Previous main project should now be loaded like any other project, with `PStore.LoadProject ("NAME", App.thisProject, App.Init)`
- `PStore.LoadMain` has been removed, to ensure that you do update your code!
- Parameter loading order has been slightly changed
- Ensure that your main project has a Command Table base other than zero, which is reserved for the internal table

## 05.05.2004

- Expanded Licence.ob2 to handle advanced configuration options
- Enhanced LicenceEdit as well, to allow the entry of more user options

## 30.04.2004

- Variable-length records and long fields (> 255 bytes) in Btrieve/Pervasive files are automatically converted to BLOB / LONGVARCHAR / LONGBINARY, making files fully compatible with SQL / ODBC access; the attribute SQL will be added to their definition
- If you do NOT wish to convert your files automatically, set `Btr.mgr.sql := FALSE` before the first call to open a database or database file
- Startup.Init procedures were changed to allow for multiple initialization points and a method to cancel the process on error: modules that use `Startup.InstallInitProc` must change the Init procedure parameter to `(VAR start: SHORTINT)`; change the test on parameter start from "IF

start" to "IF start = Startup.OnInitInst" and from "IF ~start THEN RETURN END" to "IF start # Startup.OnInitInst THEN RETURN END"; if you used "ELSE" or "IF ~start THEN" as a cleanup section, you have to explicitly add "IF start = Startup.OnCleanup THEN"; if you want to cancel initialization due to an error, set "start := Startup.CancelInit"

#### 09.04.2004

- Debug now sets the default name to "Debug yyyy-mm-dd hh-mm-ss.txt", to avoid crashing due to simultaneous access by multiple users on same directory and also to support successive versions of output

#### 30.03.2004

- Added Report variables MX / MY, through which you can retrieve the extent of the current page in pixels
- Added Bitmaps.Object Command "FULLSZ ON | OFF"
- Str.ToBoolean now accepts ON/OFF as standard terms for TRUE/FALSE as well
- Report objects may integrate commands into their definition string: sub-strings preceded with '@\*' are extracted up to ';' and sent to the Command method of the object

#### 27.03.2004

- Added Streams.Search, a fast stream-based string search procedure

#### 19.03.2004

- Added Str.IsLetter and Str.ExtractWord
- Fixed note display for active controls, not always updated properly before
- Bubble help now properly works with Windows control elements; before, bubbles could fail to display
- Modified naming for MemOps set operations for LONGINT, INTEGER and SHORTINT variables; now using proper terms Union, Difference, IntersectNotEmpty
- Db.File.CompareDef now takes a parameter to tell it if changed data should be copied and returns FALSE if change is required but was not performed
- Btr.File.Open doesn't perform updating unless create is set to TRUE, in which case the result code is Db.FormatError

#### 5.03.2004

- Automatic generation of SQL Scripts for the creation of databases works. The databases will be generated based on standard A3 database definitions, including all fields and indexes, with the exception of variable size fields, which cannot be generated in a way compatible with Btrieve / Pervasive transactional mode.
- Added support for Ctrl-F4 key sequence to close child windows; translates key sequence into Commands.Cancel and works with windows that accept that command.

#### 30.01.2004

- Added results to Db.InsProc and Db.DelProc and a "done" parameter to DelProc; the application procedure may now cancel the insert/update process and the delete procedure may cancel the delete procedure, when done is FALSE; DelProc is now also called before delete actually took place, with done = FALSE; to maintain functionality identical, just return result TRUE for both; add a test on "done" to the DelProc and only perform current procedure if done = TRUE

- Added Str.AnyCharInStr, to check if any character in a first string appears in a second string

#### 21.01.2004

- ODBC Interface: SQL.def and ODBCInst.def give access to the standard library functions, SQLDb.OB2 adds Amadeus-3high-level interface with various object classes and integration with module Db, which allows you to easily create ODBC definitions for existing databases
- Db.File.FileName now requires an additional parameter, "full"; set it to TRUE per default; FALSE will return the file name without the full database path
- Added Str.ExtractByKeyword, which extracts a sub-string based on a keyword and terminator string
- Added Str.Undelimit, to remove pairs of delimiters

#### 12.12.2003

- Constants may be used in report expressions: assignments, conditional statements etc.

#### 16.11.2003

- Added message counter Dialogs.msgCounter, allowing for detection of error message display in other stages of the application, avoiding multiple warning messages at the top level; applied this method in module Scroll, to avoid multiple notification for source update errors.
- Fixed small bug in window updating when deleting lines from scroll windows with ledger background.

#### 4.11.2003

- Printing changes: several procedures transformed into methods of class Printer; New function GetSetup supports direct selection of printer
- Added Reports.DefaultPrinter, to get and set the application default printer via a system INI entry

#### 24.10.2003

- Added Scroll.Decor class, through which you may attach named decor objects to a scroll window, for improved presentation
- Added support for decor definition to A3Edit
- Added ScrollDecor.ob2 with initial class Ledger
- Scroll.Line now contains an index number, which is attributed based on order of line insertion and allows effects as found in ScrollDecor, i.e. coloring based on line index

#### 1.10.2003

- Added Events.SpyProc and variable spy, to allow the assignment of a procedure that reads all messages sent to the application and also able to modify them.

#### 29.9.2003

- Improved bitmap scrolling; Events.KeyToScroll now translates Ctrl-Left / Right to "Page" Left / Right

## 24.9.2003

- A few more improvements in A3Edit: menu names can now be edited with "Edit" key; list is immediately updated after editing brush and pen data
- Added Commands.Copy, Cut, Paste and Undo as standard commands
- All standard editing fields now support the popup menu with the Copy, Cut, Paste and Undo command
- You may supply translations for these command names in your resource file; in name string, add "&" character before shortcut key
- Resource.Get and GetSystem now return the ID string, if no entry was found, but the returned length is still zero in that case; your application should always take this into account, if you want to supply a different default or ignore the result string, if it was not returned from the resource file; nb: in the past, the result string was supposed to be undefined, so this should not cause any problems

## 20.9.2003

- Printing now checks the spooler service, if present, to ensure it is running and otherwise restarts it
- Shift-Tab now works also with child scroll windows
- Launcher enhancements for enhanced security, cf. full documentation for registered users only

## 16.9.2003

- **IMPORTANT:** Make sure you get the latest update of A3Edit, if you recently downloaded another version; it fixes the parent window saving problems that have beleaguered A3Edit for a while, where the MAIN window was not properly saved
- Event handling and especially keyboard handling was improved, including the handling of the ESC key and DropList objects
- SrcSelect was also improved for use with search popup window

## 15.9.2003

- Fields.setData, acceptData and abortData are now constants in WinMgr.ob2, along with GainFocus and LoseFocus as standard focus changing commands; you can substitute them one to one
- Fixed small bug with graphical buttons, where button name contained extension
- Further improved A3Edit

## 7.9.2003

- A3Edit: Guidelines are now displayed in list format and directly editable; same for window objects
- A3Edit: several bug fixes (MAIN window reference etc.)
- You may now use the "&" character in Toggles and Fields to designate a shortcut; the "&" won't be displayed, instead the following character will be underlined; the user can press ALT + the underlined character to jump directly to the corresponding field within the current window
- Also fixed Esc key usage; now works with child windows as well

## 5.9.2003

- Added StrFmt.Translate option to access registry strings, via the syntax @[ @key;subKey]
- Added StrFmt.TranslateFile

## 3.9.2003

- While viewing a report, you may now use key commands:
  - Back one page: PAGE UP, UP, LEFT, BACK
  - Forward one page: PAGE DOWN, DOWN, RIGHT
  - Move to first page: HOME
  - Move to last (already viewed) page: END
  - Send report to printer: PRINT

## 2.9.2003

- Improved focus management for report viewing window
- If you use SrcSelect, you have to ensure that the data source used with SrcSelect drop list is properly saved and restored, so that it does not conflict with simultaneous uses in other parts of the application; this will be improved in the next release

## 30.8.2003

- Fixed use of SrcSelect with modal windows

## 24.8.2003

- New function DateOps.SetRange and supporting constants; allows the setting a date range via 2 variables, including week, month, quarter, trimester, semester, year
- Added WinTools.FocusBasedDispatcher

## 20.8.2003

- Added WinEvent.focusTarget and WinEvent.whFocusTgt: focusTarget is the next window object that will get the focus and whFocusTgt is the handle of the window that will receive the focus
- SrcSelect is a new dropdown field, to which you can assign any standard scroll list and result string; when pressing Alt-Down, Alt-Up or the arrow button, the scroll list is "dropped" down and the user may select an item, as in any dropdown field. The major advantage is that any data source may be connected to it, including a full database and that the presentation can be controlled: multiple columns and headers may be displayed

## 14.8.2003

- New A3Edit option allows the creation of a new dictionary entry based on a template
- A3Edit variable editing improved: on ADD, name can be entered directly
- A3Edit now prompts for the object group name, which is also used as application name; the old 8 character limitation was dropped long enough ago! Instead of [5 chars ... app ] the new default is the full Application Name + "App". You must either set this group name OBJ

Group "... " to match the exact name your application file had before or you must change all the references in your code to the new name

- A3Edit now prompts for the group name before saving, so you can do it at that moment
- Alt+Character in any window now first attempts to find an object with a name containing the sequence "&character"; if found, the focus will be moved to this object
- Add common data structures to Template.AP3/TX3, which may then be freely reused
- Fixed bug that could cause the parent window to be lost during editing for top-level windows
- Further enhanced handling of multiple main windows
- Generic Workflow added; Workflow is now operative with METIC, but it may be used with any other Amadeus-3 application with little effort
- HTML Help Support integrated in Help.OB2: to specify that a help file should be treated as HTML, just add ";html" after the help file name in the Application.INI file; make sure you download A3Lib\HtmlHelp.\* and A3Lib\sym\HtmlHelp.sym; add HtmlHelp.Lib to your project files

### 23.7.2003

- Resource.GetParam now falls back to [Param] entry, when no match found for environment-specific name, i.e. [Param.EnvironmentName]
- Enhanced DrawObj with Losange and Edge; now 3D look works for ellipse and losange as well
- NumToStr now supports multi-line results; NB: width field now by default either matches the refV variable dimension or is limited to LongStrLen; explicitly specify different values to change these
- A3Edit supports new parameters for Indirect String (NumToStr) variables: the Length field is now accepted as value for the output string length; another field allows the entry of the maximum number of lines for multi-line result strings
- Many new features in METIC

### 5.7.2003

- And since we are updating source code anyway, here comes another global change: WinMgr.Window.SetAttribute and WinMgr.Object.SetAttribute lose the BOOLEAN result

if you ever use

```
IF obj.SetAttribute (attrib, on ) THEN
```

replace this with:

```
obj.SetAttribute^ (attrib, on);
```

```
IF (attrib IN bt.attrib) = on THEN
```

### 4.7.2003

- New constant in Values.ob2 for use with Reset: ToNewData
- Added "to" parameter in Db.File.Reset; sorry about this, it will be a bit of work to adapt all the calls to Reset throughout your code, but it really helps; for identical results, replace all calls to Db.File.Reset with Db.File.Reset (Values.ToZero); you may now also specify one of the other options to reset file data

- When calling `Db.File.Reset (Values.ToNewData)`, the data record is left intact and only identification data is reset to zero, such as the id value associated with the file, if any; for `DbExtFile.File`, the make and time stamp values are also reset to zero; in this way, you may preserve the whole data record and then insert it as new element in a database

### 3.7.2003

- `Db.File.FileName` now applies `StrFmt.Translate` to database path and file name, so you may embed variables
- Added `SrcSelect`, which allows the use of any type of scroll window and source combination as dropdown window for item selection; still to be stabilized
- A3Edit support for `SrcSelect`
- Slightly reworked various A3Edit menus

### 28.6.2003

- A3EDIT: another bug fix for previous version and a few small improvements, such as more uniform context menus, with more access to underlying window via display objects and more

### 26.6.2003

- A3EDIT: Fixed annoying bugs that kept incorrect links to windows on screen after an application save and another one that made working with MAIN window more difficult; there are also no further problems with cross-application saving of displayed windows
- Updated A3 documentation uploaded

### 25.6.2003

- Many new features for METIC, among which a global search with full expression support
- NB: The search routine can easily be adapted to other Amadeus Applications ! You can get it on request. It's small and rather elegant. It's also a good demonstration of the power of Amadeus-3 data handling.
- `Exported Menus.Destroy` and added a parameter
- `Scroll.Window.Validate` now doesn't call `ClearLines` anymore, if the first line does not match; therefore, `w.src.SetInfo` is not called either; this is the correct interpretation, since `Validate` should not initialize a different data set, just update the current one; if your code relies on `Validate` to call `SetInfo`, then you should add the call to `w.src.SetInfo` in your code.
- Updated `DbViewer`, fixed refresh problems

### 13.6.2003

- Added support for bitmaps other than BMP (JPG, PNG, TIF etc.) directly in A3Edit
- Bitmap browse function now automatically reduces bitmap path to relative path, as per the location of the AP3 file
- Added STORE command to report syntax; same as SET, except that the string is not translated before storage; this allows the dynamic construction of strings containing meta-commands and variables

## 7.6.2003

- Changed Printing.ob2 and Reports.ob2, so that per default, the system definition is used for the paper format and orientation
- StrFmt.Interpret now ignores all unhandled formatting instructions for variables; make sure you understand the specific format for an input format string

## 22.5.2003

- New notification code WinMgr.NotifyFocusChg and invocation of hasFocus.Notify from within WinEvent.SetFocus and ReleaseFocus
- Added format string to ValueDsp.Object, to allow more flexible display options; works with ValueDsp.Object.GetString
- Support for format string editing added in A3Edit
- Added assumeDir parameter to Paths.Create; replace existing calls with Paths.Create (... , TRUE) to get identical results

## 6.5.2003

- Added searching to user rights editor, along with [documentation](#) of this Amadeus-3 tool
- User rights editor now supports multiple applications through the same user database file

## 5.5.2003

- Added Debug.Suspend and Debug.Resume, for better control of information written to debug file

## 30.4.2003

- Added Scroll.Window.handleDel and matching procedure type DelProc; is assigned HandleDel by default

## 26.4.2003

- Improved LicenceEdit: now allows opening of files via "File / Open" Command; environment parameter name may be specified; renamed directory consistently to LicenceEdit
- Environment parameter will be extracted from licence file and used if not supplied on command line
- Default licence file is now either licence\licence.dat or local file licence.dat, avoiding overwrite when copying files to existing installation; use licence.dat in root directory for test licence information

## 10.4.2003

- Added Sequence.Stepper.Command "ANYTAG" / "SETTAG" / "CLRRTAG" / "TAGGED"

## 23.3.2003

- Resource.Get and GetSys now use only the string before a "|" character as ID and append the reminder to the result string
- Dialogs now use StrFmt.Translate on title and message strings, i.e. you may include variables in message strings ("Expiration date: @[expDate]")

- Tools item User now supports rights management for multiple applications, i.e. the same user database can be used to manage rights for different applications; the name of Startup.project is used to identify the current set of rights. All basic elements - user name, password, flags etc. - will be shared, but application-specific rights will be treated separately.

### 20.3.2003

- Added Startup.GetSysParam and Startup.SetSysParam
- Streams.ReadLn now works also when Str.eol = cr,lf and the file only contains cr or lf alone as line separator
- Improved DbView, to work even with non-standard data sources (in memory etc)
- A3Edit: interface for options in setBmpWin (graphical buttons / bitmaps) improved

### 14.3.2003

- Added Db.File.ErrMsg and Btr.File.ErrMsg
- Enhanced DbExtFile.CheckResult

### 20.1.2003

- Added Licence.Demo (checks demo licence date) and Licence.RunCount (checks acceptable number of simultaneous program instances) to standardise program execution control

### 14.1.2003

- Added standard application launcher Launch.exe, which should assist in running networked applications; to use, just copy Launch.exe and an adapted version of Launch.ini to the user's local application directory. Launch.exe will read it's parameters from Launch.ini or the command line: based on the specified source directory, it will copy and/or update all application files (ignoring source and other development files) and then start the application with the specified parameters. These parameters typically include the database directory, the location of the User database and other startup parameters. If you supply these parameters via a desktop icon, you don't necessarily need Launch.ini, but if your installation is standardized (same network drive setup etc.) it is a lot easier to distribute just Launch.exe and Launch.ini with standard parameters

### 27.12.2002

- Added class change to user interface in A3Edit; you may now change the class of an object in A3Edit by selecting it with Cntrl-Left-Click, then choosing from the base class of the object and related application-defined classes that share the same base class; this may be done for window objects or objects in the main Object List or the database file list.
- Added documentation on application-defined classes and new A3Edit functions

### 17.12.2002

- Improved Text and CSV export via reports: numbers are now always exported without tick marks, to ensure compatibility with other software packages, such as EXCEL etc. which may get confused
- Fixed text export from RptTbl.Object: empty columns were not exported at all

- Convert now ignores the tickMark character, when it is set to 0X
- Added parameter "dispose" to method WinMgr.Window.Clear; if set to TRUE, items will be disposed as well as removed from the owner window (eg. large bitmaps, where you don't want to wait for the garbage collector to act)

#### 01.12.2002

- Fixed the redrawing of tabbed controls during dynamic resize operation; child windows of tabbed windows now are also properly resized when the dimensions are changed before the window is being displayed
- Added Str.CountLines to calculate number of lines in string by line break characters
- Variable-length for ValueDsp.Object text fields: if the "lines" field is negative, it specifies that the field should be of variable length and at most ABS (lines) long; this also implies that you cannot specify constant values for the number of display lines
- Added ValueDsp.Object.GetText method for retrieving the text contents of a ValueDsp object

#### 29.11.2002

- Added variable font support with standard and system fonts: Fonts.defFnt and Fonts.defMet were extended to a total of 4 default fonts, which are selected via the constants StdRegular, StdBold, SystemRegular and SystemBold
- Font selection is now the following: elements that require precise positioning, such as data entry fields and buttons, will use the System fonts; elements that are flexible in respect to font size will use Std fonts; these include scroll lists, multi-line fields, text blocks etc.
- For further information, please read the chapter on flexible fonts in the user interface

#### 27.11.2002

- Added TextEdit.Object.fnt, fg, bg for font and colour support; also extended syntax for encoding of these elements
- A3Edit now supports setting font and colour on text blocks by drag & drop of the corresponding elements
- Adding Text blocks by make menu is now also possible in A3Edit
- Startup now defines a variable "debug", which may be set via resource entry "[Param]:DebugMode = 0 | 1"
- debug mode is used in Dialogs instead of previously introduced mode variables to determine if unknown dialog messages should be added to resource file

#### 20.11.2002

- Added RealVal.Value.GetLong and PutLong, equivalent to NumVal.Value.GetLong and PutLong
- Display font may now be specified as resolution-dependent in application INI file; specify entries for Font, Font800, Font1024, Font1280 or Font1600
- Alternatively, standard fonts may be defined in the parameter section, which may adapt to various environments (cf. Ressource)
- Character size of multi-line fields is now correctly calculated for all resolutions and screen types
- Fixed global left alignment in StrFmt.Translate

## 5.11.2002

- Added StrFmt.Compile and Apply for precompilation of format strings and fast execution
- Added US printer format constants to Printing.ob2
- Added identifiers for new paper formats to report syntax
- Replaced RptTbl.reportStrLen with Reports.maxStrLen
- Added Str.PStrBase definition
- Removed RptTbl.ReportStr; use Str.PStr instead
- Report objects now all use double-indirection via @!

## 29.10.2002

- Added modal editing mode for Scroll.Window

## 21.10.2002

- Added Paths.AvailableSpace to check how much space is assigned or available to a directory

## 11.10.2002

- Enhanced Dialogs.ob2: multi/line message support, YesNoRetry selection
- DbExtFile.CrossUpdate now also works with descending key segments

## 10.10.2002

- A3Edit: Fixed 2 bugs related to copying Scroll Windows
- Fixed bug with menu editing

## 30.09.2002

- Enhanced Streams.Buffered handling with PosStreams; it now re-calculates positions if you switch from reading to writing mode or if you use SeekRel for relative positioning; it also calculates the correct position with GetPos, taking into account the position inside the buffer, not only the actual position inside the PosStream
- NumVal.Value.Encode now only encodes an attached range, if TokenStream.useTables is TRUE
- Proper icon for DbViewer added
- Added Reports.Report.SelectFont, to ensure safe font selection; incorrect font specification in a report could cause crash

## 24.09.2002

- Undo (Ctrl-Z & Ctrl-U) in Guideline editing mode now works properly again
- Fixed scroll update flickering on insert with graphical objects in list

## 23.09.2002

- Enhanced interactive definition of scroll columns; option lists may now be specified directly for ValueDsp.Object columns; these may also be displayed in the style of 3D buttons, giving the typical grid appearance
- Small improvements in the A3Edit Interface
- A few minor bugs fixed, with relation to field handling, numeric range parsing etc.
- WinMgr.Resize now enforces window minimum and maximum window size

## 14.09.2002

- Updated documentation, added Contents and Figure tables

## 10.09.2002

- Unified numeric "Range" Class to NumVal.Range, removing RealVal.Range; all range values are now stored as real number ranges, since they are only used as comparison anyway; simplifies code significantly and wastes very little storage
- Added numeric range creation and editing to A3Edit

## 09.09.2002

- Convert.ExtractNum now correctly ignores tick marks
- Improved various A3Edit window layouts
- Scroll columns may now be attached to guidelines, allowing for dynamic resizing of scroll columns; added corresponding editing functions to A3Edit
- Improved creation of objects and dictionary elements; new objects may now be created directly in A3Edit
- Revised and improved documentation

## 30.08.2002

- Added StrVal.ToUpper...ToCaps, which, when set in Value.convert, will force the compare operation to be done with the matching conversion, per default in uppercase and ASCII
- Added NumVal.Value.showZero parameter, to show or hide zero values; also added "filler" character - if specified, StdFormat and Trimmed are automatically converted to AlignRight
- Added NumVal and RealVal range encoding and decoding:Obj "name" { min "," max" } ENDOBJ
- NumVal and RangeVal values may now include a named range in their definition : RANGE "name"
- Extended A3Edit to handle the new NumVal and RealVal parameters and ranges - partially. You still need to edit the TX3 file to add a range object.

## 22.08.2002

- Dialogs now center on main window, if available, not on full screen
- Guidelines are now fully applied even if you changed the window dimensions before displaying it for the first time

- Fixed a range of minor bugs in A3Edit (Guidelines editing lose links after certain operations; Align field properly when changing label; don't keep global length; Proper field size on creation; now way too short; Keep author name when defining new class modules; store in system INI; Dropping non-display-objects leaves "phantom object"; Db editing: adding new key deletes previous ones)
- Added new alignment features to A3Edit (align right while respecting the text offset, very useful!)
- Added Scroll.NoParentSave attribute, to allow individual setting per window of the use of SaveParentData method
- Startup now loads the "Show state" parameter in the info record, i.e. you can use it to launch your program normal or maximized for example by configuring a desktop icon
- Since it is no longer used, remove the second parameter from all calls to Startup.WinStart (... , ?); if you made use of it, instead set the startup window state in the main window, e.g. in the InitInstance method

#### 16.08.2002

- Improved Guideline editing in A3Edit substantially
- GuideInfo may now also be used to define minimal and maximal extent of window; especially for complex, assembled windows, it is important to set a minimal size, which guarantees that the user always sees essential information
- WinTools was enhanced and slightly modified
- Documentation was expanded
- Guideline demo was updated
- Code generated by A3Edit was slightly changed, to avoid crashing when some objects are not present

#### 12.08.2002

- Added Guideline editing to A3Edit

#### 10.08.2002

- Fixed small bug in A3Edit
- Added Guidelines documentation

#### 07.08.2002

- WinMgr.Window.CopyParts now also needs "deep" parameter, for depth of copy; also copies guideline links for objects; added WinMgr.Window.CopyGuidelines
- When copying windows in A3Edit, the full contents of the window is copied along, with guidelines and all other attributes, making for a perfect copy you will however have to rename the copied windows, to make sure that there will be no conflicts
- Slightly changed Persist.EncodeStart and added EncodePrivate, for more encoding control; also added NoTk for "No Token", where specification of token is required by parameter
- A3Edit preserves guide line links
- Fixed several small problems with parsing; doesn't affect applications, only concerns problems with incorrect object scripts

- Still working on small bug with scroll window resizing with guide lines; only occurs in tabbed windows and with specific source problems.

05.08.2002

- Added MemOps.InclShort, ExclShort and InShort to use SHORTINT values as short SETs.
- BIG Thing: Finally added a new way of positioning objects in window: GUIDELINES; cf. [corresponding documentation](#) on web site

01.08.2002

- You may now start A3Edit with parameters containing meta-characters, such as "A\*.AP3\*.TX3" and all appropriate files will be loaded immediately.
- Scroll column adding has been changed in A3Edit; initial column width now depends on the object added and is not automatically equalized
- WinEvent.inUserChg is now a Window pointer instead of a Boolean, recording which window is actually being changed
- This change allows for a better management of scroll windows, for which updating can be delayed during window resizing
- A double-click on the stage or in the "trash can" area (in case the stage window is not accessible) brings up the menu with all open application windows; if you used the left button and then select a window name, that window will be brought to the front; if you used the right button, then the selected window will be removed from the display

31.07.2002

- Added Persist.Group.IsEmpty and Count, to check contents of group
- Changed Projects.Load to Projects.TryLoad and added PStore.InitProject, LoadProject and LoadMain as higher-level substitute with implicate error message
- Instead of: loading a series of projects like this:

```
IF Projects.Load ('Base', Base.thisProject) & Base.Init
(Base.thisProject, FALSE) & p.InitInstance^ () & App.Init (p, TRUE)
THEN
```

your main InitInstance will look like this:

```
IF PStore.LoadProject ('Base', Base.thisProject, Base.Init) &
PStore.LoadMain (p, App.Init) & p.InitInstance^ () THEN
```

- Add the following messages to your [Err] section of your INI files for proper error display:

```
ErrInAP3=Can't load the AP3/TX3 file
```

```
ErrAP3Asgn=Can't assign the application
```

With the above changes, you will automatically get full error messages when application files don't load or don't initialize properly.

- You must GENERATE all \*app.OB2 Files with A3Edit, since there is a small change in the Init procedure
- Btr.ob2 now auto-installs; you don't need to call Btr.mgr.Install anymore

28.07.2002

- Added parameter support to module Resource with GetParam, SetParam and SetParamGroup; also supports command line setting via parameter "env:"
- Improved auto-fill function, to make it work properly with objects over multiple lines and control objects (fields etc.)

21.07.2002

- Startup.CmdParam now removes "+" and "-" in front of parameters, even if not specified in call; greater flexibility in parameter specification; param name now case insensitive
- Various bug fixes and improvements in A3Edit
- Fixed hatched brushes
- Renamed project files and moved XDS Information to corresponding INI file, to avoid having Project info within the application INI file; use exename instruction to set executable name

10.07.2002

- Added ValueDsp.Object.lines to specify maximum number of lines to show for text fields. Added editing function to A3Edit.
- Added Values.AssignData, just for "emblishment" of client code
- A3Edit: Added complete information on filter and linked files to Scroll list window display; also fixed refresh bug in resized display
- Scroll lists now refresh only after a resize operation is complete; alternatively, a parameter could be added to select immediate refresh, but this should be useful in most cases; slow database access may make immediate refresh unacceptable
- Added functions Common.StrToCond and CondToStr; other utilities added: Str.AppendEol

24.06.2002

- Added NumberToString.ob2 for conversion of number to fully spelled out string in English, French and German (for now)

16.06.200

- Added Registry access functions to module Resource.ob2

12.06.2002

- Added work-around in Bitmpas.OB2, that will avoid a crash under Windows 95/98 when the garbage collector attempts finalisation of BMP bitmaps. Does not apply to W2K and XP

## 07.06.2002

- Added more functions to Debug.ob2; file may now be closed or discarded, which allows immediate access to it during execution; name may be set freely, with Debug.txt as default

## 23.05.2002

- Added Scroll.saveParentData, to control the use of Scroll.Window.SetParentData globally
- Added minimal process control through Shell.CallProcess and Shell.WaitForProcess, which allow a minimum of synchronisation between programs
- Projects.Load now properly returns TRUE, if an application file (AP3/TX3) could not be loaded for the specified ID and the INI file specifies NO application name
- Moved some constants from Sequence to Common (Less, Equal, Greater, On, Off, Invert)
- Improved DbViewer: search by name for files; DFI file or directory name accepted as command line parameter, i.e. you may associate DFI files with DbViewer

## 15.05.2002

- Bitmaps.Load now translates object name before attempting to use it as file name, allowing for embedded variables
- Reports recognize new condition: VAREXIST "variable\_name" returns TRUE, if "variable\_name" is the name of an existing variable from data dictionary

## 11.05.2002

- DbViewer prompts for password for protected files; signals errors such as attempts to alter non-modifiable keys etc.

## 10.05.2002

- DbViewer now has dynamic mask, sorting and searching by key, file printing, better display layout

## 07.05.2002

- Added module Interactions.ob2, which includes some extended functions for the user interface, such as "line sorting through dragging" as in A3Edit
- Added Buttons.SetPressed procedure, which replaces the Tools.SetActive procedure (not standard, but in use)
- DbViewer is taking on shape; many functions now actually work

## 04.05.2002

- Further improved DateTime.ob2, added support for UTC variables to A3Edit; only support for UTC objects as database keys is still required

## 03.05.2002

- Added Debug.WriteUTC
- Added Str.AbbrevTable to generate default abbreviation table

- New attributes for Date variables and fields: DayOnly,MonthOnly,YearOnly for StrToDate and DateToStr; added month and day abbreviation tables
- Abbreviation tables may now be specified in INI file, along with MonthNames and DayNames parameters
- Support for new date attributes added to module DateVal
- =>DELETE all your \*.AP3 Files and recreate them from corresponding TX3 files

#### 01.5.2002

- NEW MODULE: DateTime.ob2 combines certain functions from DateOps and TimeOps and adds support for UTC (Coordinated Universal Time)
- Enhanced DateTime.ob2 and DateOps.ob2; now accepts month names as input; allows separated input of day, month and year for separate fields
- Replace Fields.AlignRight with AlignOpposite, as numeric fields now are automatically right-aligned, this is more useful; for strings and other field types, it means "align right" and for numeric fields, it means "align left"; make sure you replace the keyword ALIGNRIGHT in your object scripts if necessary with ALIGNOPPOSITE, or remove it entirely, where no longer appropriate, i.e. for right-aligned numbers
- Added call to Printing.Printer.Mapping in Printing.Printer.Create, to ensure that page dimensions are correct from first page; required by change to Job based printing
- Added MaxPage as standard report variable; to calculate the number of pages before calling PrintAll, just set maxPage to 0 in your report variable; set it to NoCount otherwise
- Displayed page now correctly refreshed when attempting to move beyond last page; contents no longer lost on refresh
- Added Controls.Control.Text method for saving and loading GUI-specific text in control
- Added DropList.Field.IsDropped method
- Moved the following functions from template module Tools.OB2 to WinTools: Center, HideAll, HideEditWindows
- Added WinTools.FitToParent and WinTools.staticGrp; add static windows for your application to this group
- Added WinTools.Center; modified Tools.ob2 template correspondingly
- Fields.SetCursor / GetCursor are now methods of Field class
- Added Fields.FindByData, to avoid import of module SYSTEM when searching for fields by data address
- New XDS\BIN\XC.RED file, for changed directory organisation (all compiler generated files in OBJ directory: obj, sym, res, lnk), but feel free to use different organisation

#### 10.04.2002

- Enhanced Persist.Object.Command, allowing it to return string values and new commands
- Added object class Persist.Container and Command class handling special commands
- Added module DictTools for Dictionary tools; moved Reports.AddVar to this module
- Added Persist.FindProc and Projects.FindGlobal, which is assigned to Persist.find

- Added Values.Value.Command with the command codes GET and SET
- Enhanced module DbReport, to handle automatic printing of databases and their contents
- RptTbl now can do a double-translate, if you precede the string with the sequence @!
- Included new features in documentation and added several new chapters
- Added Events.UserStartChg, Events.UserStopChg and WinEvent.inUserChg, allowing the application to know when window resizing started / ended
- NB: You will need to change your main module slightly to take advantage of this new feature; please check the news section for instructions
- Since the program can now distinguish between program-generated move/resize and user actions, you may block the latter, without inhibiting program actions
- Added WinTools.StdModalCheck, which combines StdModal editing mode and field check
- Scroll.CheckAll now accepts any WinMgr.Window type as input

### 31.03.2002

- Added tracking for windows that are child windows but not fully attached, so that their parent window also displays scroll bars when they are outside the viewable area (if scroll bars are allowed according to attribute settings)
- Made Startup.CmdFlag case insensitive
- Fixed slight bug in Str.PCToAnsi with extended translation strings
- Fixed bug in A3Edit / DbEdit, where key lists were not properly reset
- Data entry fields for numeric values now automatically use right aligned / number style (which was atrociously bad in Win3.x mode)
- Added Fields.EditField.DataType method to control the above effect for derived fields
- Added new frame types to WinMgr: WindowFrame, ClientFrame and StaticFrame
- Added new window attributes in WinMgr: IsTopmost, IsToolwin and Transparent
- Added Metrics.SmallTitleHeight to retrieve the extent of toolwindow titles
- Added support for new features in A3Edit
- Fixed bug in attribute saving in A3Edit (concerns mostly DROPOK & SELECTOK) =>DELETE \*.AP3 Files and recreate them from corresponding TX3 files  
=>Before conversion, remove the attributes DROPOK SELECTOK from your TX3 files, except where you actually intended to use them, then proceed with the conversion
- Fixed bug with scroll window columns and variable deletion
- Improved DB editing in A3Edit
- Careful with the DOREORDER optimization; this generated a bug in A3Edit; had to withdraw it for safety reasons...
- Turned Reports.StdTranslator and StdVariables into methods
- Added Reports.Report.AddObj

- Added Paths.CompleteFrom to complete file names from other source than current directory
- Added boolean result to Files.Rename, so errors can be detected
- Startup.OB2 now automatically recognizes when multiple MAIN windows were loaded and assigned and re-assigns automatically the latest copy instead
- Moved ExtFile from Templates to official A3 Framework and renamed it to DbExtFile
- Dialogs now translates all strings via StrFmt.Translate, meaning you can embed variables
- Extracted module Options from module DropDown; Options.List replaces DropDown.OptionList and Options.Option
- =>Globally replace: DropList.OptionList with Options.List; DropList.optionListId with Options.listId; DropList.Option with Options.Option
- Changed Commands.BaseOffset to 1000, to allow for more simultaneous projects with independent command codes; if you used command code values over 999 in any of your TX3 files, change them so that they fit into the range 100 - 999
- =>DELETE all your \*.AP3 Files and recreate them from corresponding TX3 files (for both, command codes and option lists)
- A3Edit: Text block editing
- Fixed bug in TextEdit.Object.Dimension

28.02.2002

- ListView encoding now includes the list name and the data variable, if specified
- Added A3Edit support for ListView source parameters
- Dialogs.OB2: Changed filterSeparator from character 10X to "|"; the aim is to eliminate non-standard characters; source code managers etc. complain when these characters appear =>YOU MUST REPLACE THIS CHARACTER IN YOUR SOURCE AND INI FILES This concerns all entries for File selection prompts, as filterSeparator is used to group the different selection options
- Added WinMgr.Window.Notify and WinMgr.WinMgr.NotifyResized, NotifyMoved
- Added WinEvent.noteNewLine and changed default new line separator in info bubbles from ";" to "|" =>Change this character in bubble notes globally throughout your TX3 files and re-generate AP3 files or change the assignment of noteNewLine in your application
- Added Str.ReplaceChar to allow for replacement of only one character
- Added Graphics.CreateContext, Graphics.Palette
- Added Bitmaps.CreateInContext, SelectIntoContext
- Added ImgFiles constants, Create and Render procedures
- Added MemList.List.Clear
- Fixed a few bugs in A3Edit
- Added module SrcSelect to build drop list fields based on a generic Sequence.Source

- Added method Sequence.Reset option for SetInfo; make sure you call inherited method
- Added parameter "identical" to method Sequence.Source.Locate
- Added method Sequence.Source.Search
- Added a few accented uppercase characters to Str.ansiUpper
- Added module Common for various constants; removed Values.Less..CmpError; removed DateOps.Less..Greater; removed TimeOps.Less..Greater => Replace with Common...
- Added RealVal.Compare procedure for approximate comparison of real values

### 31.12.2001

- If WinMgr.Object.Dimension is called with other than InitialExtent, MinExtent or MaxExtent and the dimension is still undefined, then the method calls itself recursively with InitialExtent
- Extended documentation, including inside many modules
- Added extended note support to Persist, including: Note tag identifiers (NoteComment, NoteDisplay, NoteHelp) DelNoteTag, Object.SetNote, Object.HasNote and Object.GetNote and parsing support
- Added Str.ResizeTable for automatic resizing of dynamic string tables
- Added DynLoadStr for automatic resizing before load of open string tables
- MemList.List is now also a Persist.Object; you will have to replace List.Init (size) with List.Init; List.SetAtomSize (size); the list object is now fully registered;
- Modified WinEvent.DispNote, accepting note text directly; event handler now uses Persist.Object.GetNote to find display text
- Added support for new Persist.Object notes in A3Edit: menu functions for display objects and windows now give access to note definition
- Improved Help support; now uses Persist.Object.GetNote (NoteHelp...) to access correct Help information; Contents and Index now use HELP\_FINDER, according to Win32 specs
- Cleaned up module Startup; project without name now skips some steps during initialization; removed Startup.Project.Cleanup method (use Dispose, if necessary)
- Startup.WinStart now accepts project variable as parameter; Startup.project has become read-only; you will have to change main program initialisation code
- Re-arranged function distribution between Events and WinEvent: moved Message type, GetMessage, PeekMessage, CheckForMessage to Events
- Moved WinEvent.DestroyAllWindows to WinMgr
- Moved WinEvent.CheckCommand to WinMgr
- Removed procedure WinMgr.KillWindow
- Added procedure WinMgr.MoveTo (Object)
- Added Startup.Project.Handler, which replaces WinEvent.defHandler, now removed =>Change all your main module; replace DefaultHandler procedure with (p: project) Handler, remove assignment of WinEvent.defHandler and add parameter project to Startup.WinStart

- Removed WinEvent.CaptureWait and ReleaseWait; replace all calls with Pointer.CaptureWait and ReleaseWait, as WinEvent wasn't doing anything more and just added a layer (used to invoke UpdateAll, which proved to have unpleasant side effects; better manage this within your application)
- Projects.Project.Init and Load slightly changed to adjust to garbage collector and changed use of Startup.Project
- Variable WinMgr.winEventHandler allows re-direction of GUI event handler procedure
- Added Values.DictEntry.AssignedTo check, allowing a reduction of the use of SYSTEM.ADR
- Added Fields.SetAccessByData, allowing further reduction of the use of SYSTEM.ADR
- Scroll.Mask.SaveData now returns TRUE, if no scroll window is linked to the mask
- Fixed key number handling in A3Edit
- Project author now saved in System.INI when creating projects under A3Edit; also directly opens newly created project file

16.12.2001

\*\*\*\*\* Final move to WIN32 version, abandoning WIN 3.x compatibility

- The first implication is full reliance on garbage collector; no more calls to SYSTEM.DISPOSE; you may also remove procedures and methods that only serve to free up memory structures
  - =>You are advised to go through your code, updating it where necessary; remove all unnecessary imports of module SYSTEM, your code will gain in clarity; import of SYSTEM will work as intended, marking potentially UNSAFE modules
- Removed Graphics.MyInstance, which was a 'temporary' solution anyway, for compatibility purposes only
- Startup.WinStart parameters changed - requires change of all main modules: => Change Startup.WinStart call to Startup.WinStart (0); without conditions => Remove startRes variable declaration => Remove import of SYSTEM, Win:=Windows and Graphics, if not required
- Added Metrics.GetLong, which returns LONGINT
- Files.File now finalized (files automatically closed when collected)
- Files.File.Close lost attribute (discard) => you'll have to change calls
- Removed Files.pool and all that was linked to it; no longer necessary with garbage collector and finalizer =>Garbage collection makes a few changes necessary to the way buffered streams are handled in Parsing etc.
- Added Files.OpenBuffered
- Parsing.TokenStream no longer creates a buffer stream for faster input =>Wherever you call TokenStream.Init, make sure that you pass it a Buffered stream for faster reading; the easiest way is to replace the file with a Streams.Stream and call Files.OpenBuffered instead of OpenNew

- Buffered streams are now declared as sub-class of PosStream and are able to delegate position-related methods to linked stream, if it is a PosStream
- Revised Streams, Files, Paths and Parsing inline documentation
- Removed Str.Dispose and Str.DisposeTbl; replace simply with ':= NIL'
- Added Persist.Object.Finalize; add call to this method for all object classes that need to destroy external links when they are collected
- Persist.Group.RemoveLink lost parameter del - useless, as implicate with garbage collector
- Removed Parsing.TokenStream.Dispose
- Added Graphics.Pie
- Added WinEvent.FocusToWindow; forces focus to window without specific object
- Removed ListView.Source.owned and everything related to it; not required with gc
- Fixed bug in ImgLib: ImgData records need to be passed by reference, of course!
- Fixed Reports bug with SET number "string"; if string is simple number, it may be converted with errors via a real number; now expression evaluation is only used if the string is not a simple number
- Added WinEvent.ExtModal, to allow modal operation starting at specific input object
- Added Paths.CurrentDirectory, which returns both, CurrentDevice and CurrentDir
- A3Edit now locates template files correctly via the ProgEnv.ProgramName directory info, even if A3Edit was not started from it's own directory
- Added Scroll.Window.SaveParentData, which calls Scroll.Mask.SaveData for a data entry mask to which scroll list is child window
- Added WinEvent.IsMinimized, which returns TRUE if a window or any window of which it is a child window is minimized (iconized)
- Fixed bug in Note window display, that caused minimized windows to be enlarged automatically
- Moved Scroll.Window.noUpdate to WinMgr.Window.noUpdate, replacing BOOLEAN attribute adjusting; if you want to inhibit various window updating functions, just call INC (window.noUpdate), then call DEC (window.noUpdate) to allow updating again  
=>Significantly improved refreshing of scroll windows, eliminated unnecessary repeat calls to Validate and more (misfunctioning mostly due to excessive Resize messages)

26.11.2001

- Changed Values.External Encode/Decode and handling within A3Edit to accept constant values for array and item size
- Added Values.Matches and constants LessEq, GtEqual
- Added DbView.Source.vlist with all corresponding extensions to handle automatic filtering, including A3Edit support
- >> Many bug fixes and improvements in A3Edit, cf. related file in A3Edit
- Fixed bug in Resource.String
- Added window list to user rights editing in Tools\User

- Enhanced DropList.Reload a bit more, by replacing the setToFirst boolean parameter with parameter setTo, which can take any of the values ToEmpty, ToFirst and ToCurrent; also adapted behavior: the setTo parameter is only applied, when the field is currently empty (f.is.IsEmpty () = TRUE)
- Changed Graphics.Point definition to Win.POINT (attention: INTEGER / LONGINT)
- Added Graphics.PointInRect, which checks if a point is in a rectangle
- Added Graphics.RectIntersect, which checks if two rectangle coordinates overlap
- Added Graphics.Polygon
- Sequence, DateOps and TimeOps Less / Greater constants adjusted to fit Values
- Finally: Reports are now printed as a single job, which is important for use on networks and when using printer emulators, such as Adobe PDF Writer etc.
- Added prompt to DropList, before adding new element; string is "Prompt:AddOk", so you should add this to your INI files
- Added Scroll.CopyColumns method, which copies columns with attributes
- Made user-defined colour table in Dialogs accessible for outside definition and saving it from call to call; variable is Dialogs.ctbl
- Added Dialogs.InitColourTbl
- Added BOOLEAN result to DropList.OptionList.AddOption; method now checks for adding of identical codes
- Specialty fonts should now work in report by default, without specification of character set; the default set is now DEFAULT\_CHARSET (1)
- Added "New project" command to A3Edit
- Added module creation to class editing (MAKE MODULE)

## 9. INSTALLATION

As the environment for GUI development tends to be complex, please read the following carefully. Much of this information you can get from your compiler installation manual, but here is a quick overview that may save you a lot of time.

### Suggestions

It is assumed that you install everything on the same drive under DOS, OS/2 and Windows, using the default names supplied by the various systems. Under UNIX, you don't have drive letters; the compiler and other executable tools are probably installed under /usr/bin or /usr/home/yourhomedir/bin etc.

Under DOS/Windows and OS/2, it is often much better to create several smaller partitions, as this will reduce time spent searching for files, adds an additional structure layer (you may define a development partition, a test partition, a production partition etc.) and reduces wasting disk space, if you still use FAT partitions.

Under OS/2, save your source files on a FAT drive, which may be accessed from plain DOS/Windows, as well. System and Production partitions could be under HPFS.

Under Windows 95/98, use FAT32 partitions, except if you still want to access them from OS/2, in which case FAT partitions would be better. Linux now also supports read access to FAT32.

Under Windows 2000 / XP, the choice depends on various factors. It's probably preferable to keep your development files on a FAT32 partition as well, whereas the System and Production partition may well be NTFS. If so, make sure that the system partition is small. NTFS performance is very bad on large drives, especially when it's a system partition, where most of the files are clustered at the beginning of the partition.

### 9.1 Install the compiler first

Independently of your platform, first install your compiler(s), whether it is a signal Oberon-2 native compiler or an Oberon-2 to C precompiler with one or more C/C++ backend compilers.

It is very advisable to keep your development files in one directory tree. In the sample below, this directory is called DEV, for "Development", of course. Under Unix / Linux, you'll obviously have to call it something else. The name "src" is not perfect either, as there are tons of files that are not really source code.

It is preferable not to spread compiler files, libraries etc. all over the place. Think of the backup effort. If all your development-related files are in the same directory tree, it's very easy to find those that have changed since the last backup or to just back up the entire directory tree.

#### 9.1.1 Installing Amadeus-3 under Windows 95/98/NT/2000/XP

First, run the A3Setup.EXE program and follow it's instructions.

Then run the A3Tools.EXE program, to install additional libraries and tools.

If you purchased the Pervasive database, please proceed with the installation of the Pervasive SDK kit.

#### 9.1.2 Installing Amadeus-3 under DOS and OS/2

Run the A3INSTALL.EXE program and follow it's instructions

#### 9.1.3 Installing Amadeus-3 under LINUX

*To be defined*

## 9.2 Directory structure for a typical development environment

This is a suggestion on how you could arrange your directory structure. Feel free to organize your files differently, if you already have an arrangement that you are comfortable with.

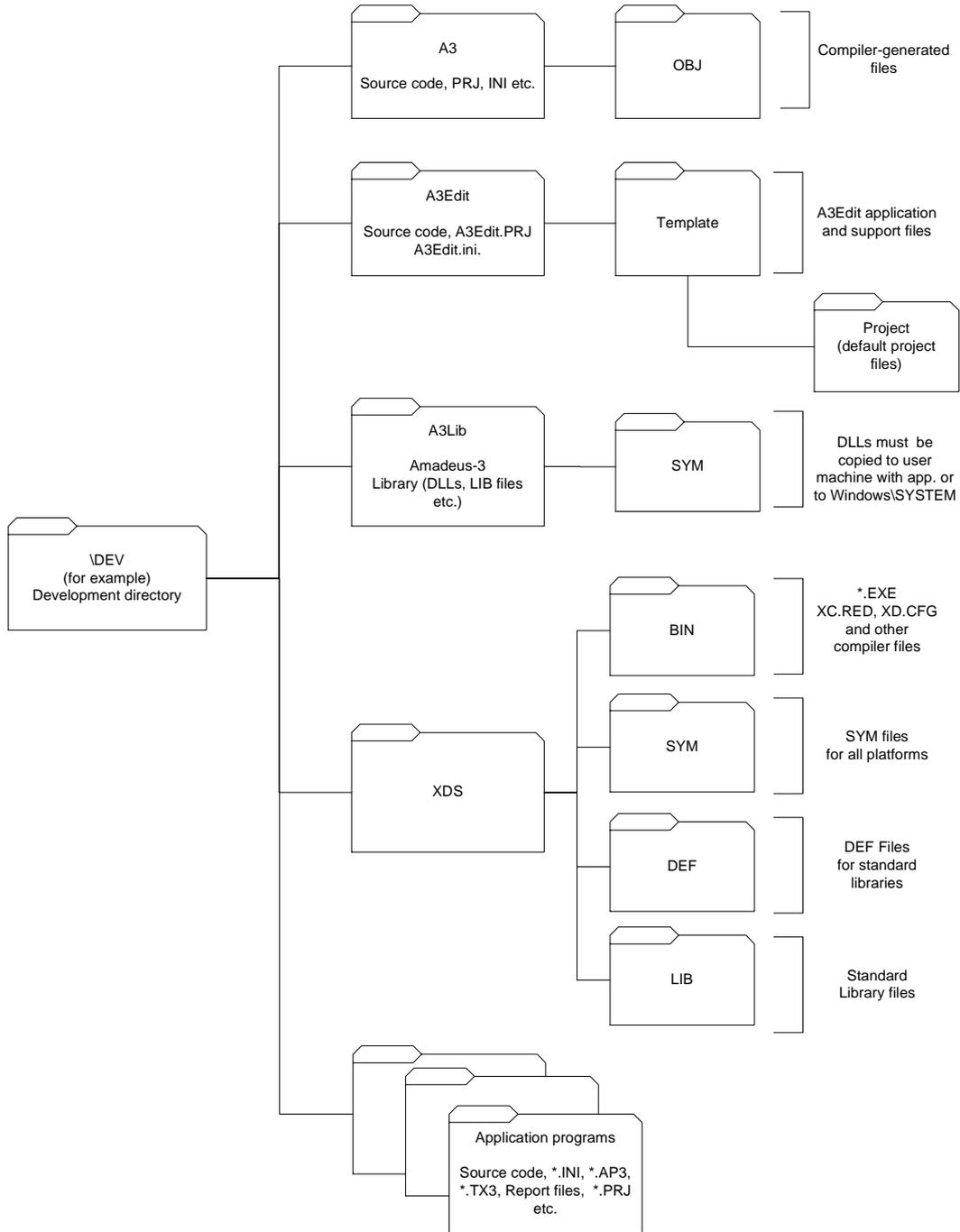


Figure 62 - Suggested directory structure with XDS

If you still use the Extacy / DOS / Win32 or DOS compiler, then your directory tree should look more like this:

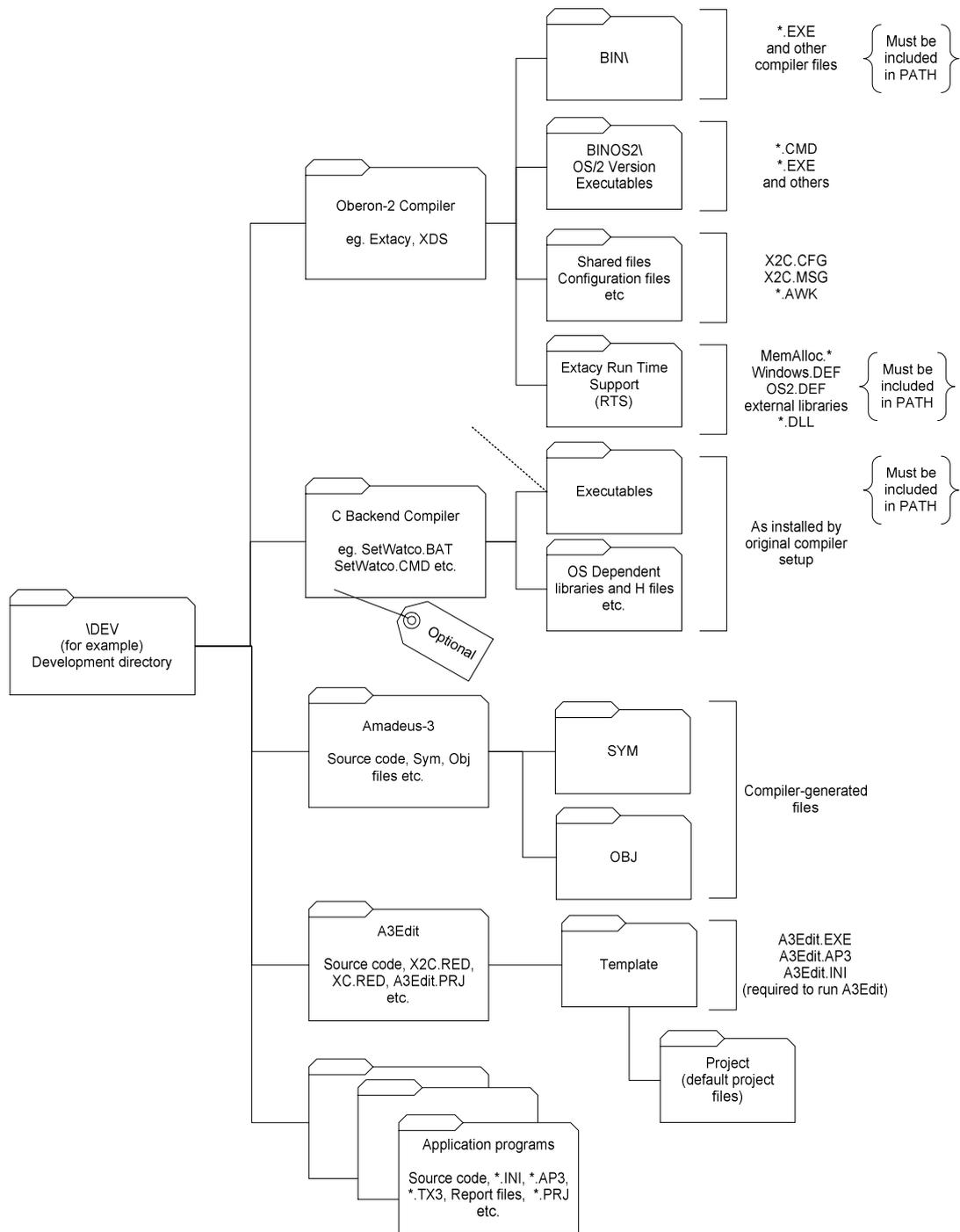


Figure 63 - Directory structure for DOS / Win32 and OS/2

### 9.3 Extacy Oberon-2 to C Translator

The requirements for Amadeus-3 development obviously depend on your platform. Extacy, on the other hand, works quite independently of the platform. The few things you have to know are the following:

#### PATH

Under UNIX as under DOS and OS/2, you must set your path so that it points to Extacy/bin and to your C Compiler executables, cf. below.

#### Redirection file

This file is named x2c.red and must always reside in your current working directory. It should point to all the other important files and directories. Please use the one supplied with the template application in a3\template.

#### Configuration files

Extacy needs a configuration file, specified with full path inside the local x2c.red redirection file. In this configuration file you specify all the defaults for compiler parameters.

##### *Recommended Configuration*

```
% Copyright (c) 1993 xTech, RTA.    ~progress
All rights reserved                :storage
% Amadeus-3 Configuration file      :fatfs
                                      :verbose

#copyright (C) 1994 by <your name :def
goes here>                          :m2addtypes
#def      .def                       :m2extensions
#mod      .mod                       :o2extensions
#code     .c                         :o2isopragma
#header   .h                         :longname
#oberon   .ob2                       :xcomments
#mkfext   .mkf                       :report
#import    .imp                       :lineno
#errlim   16
#bsdef    .o2d                       % Uncomment these lines
#xtypelim 300                        % to prevent run-time checks
:bsclosure
:bsredefine                          % :nocheck
:wofff302                            % -ADGTYRPN
```

### 9.3.1 SETUP

As Extacy needs a C Compiler, you will have to install this, too.

#### Compiler: WATCOM V.10

This compiler comes with it's own Windows and OS/2 development tools and libraries, so you don't have to install the Microsoft SDK.

#### *Environment variables*

```
PATH=?;C:\EXTACY\BIN;C:\WATCOM\BIN;C:\WATCOM\BINB;C:\WATCOM\BINW
WATCOM=C:\WATCOM
INCLUDE=.;C:\A3\H;C:\EXTACY\RTS;C:\WATCOM\H;C:\WATCOM\H\WIN;
LIB=C:\WATCOM\LIB386;C:\WATCOM\LIB386\WIN\
```

#### *Extacy redirection file*

```
x2c.msg      =..\EXTACY\BIN
x2c.cfg      =.;..\EXTACY\BIN
template.*=.;..\EXTACY\PRJ
*.sym        =.;SYM;..\A3\SYM;..\EXTACY\RTS
*.c          =.;C;..\A3\C;..\EXTACY\RTS
*.h          =.;h;..\A3\H;..\EXTACY\RTS;..\WATCOM\H;..\WATCOM\H\WIN
*.def        =.;def;..\A3;..\EXTACY\RTS
*.mod        =.;mod;..\A3;..\EXTACY\RTS
*.o2d        =.;o2d;..\A3;
*.ob2        =.;ob2;..\A3;..\EXTACY\RTS;
*.prj        =.;prj;..\EXTACY\PRJ
*.mkf        =.
```

Note that paths are specified relative to the current directory, which makes it easy to move the development environment to a different machine or disk partition, as long as the entire environment is kept on the same partition.

#### *Recommended Compiler options*

For Windows:

```
-ml -w0 -zq -3 -e25 -zW -zpl -d_WINDOWS -d_msdos -bt=windows
```

For OS/2:

```
<to be compiled>
```

For debugging, add:

```
-dDEBUG -d2
```

#### Compiler: MSC V.6 and later

To use this compiler for Windows development, you have to install the SDK with all it's utilities, which you have to purchase separately.

#### *Environment variables*

```
PATH=C:\EXTACY\BIN;C:\C600\BIN;C:\C600\BINB;C:\WINDEV\BIN
INCLUDE=C:\C600\INCLUDE;C:\WINDEV\INCLUDE
INIT=C:\C600\INIT
LIB=C:\C600\LIB\
```

#### *Extacy redirection file*

Same as for Watcom, except for this:

```
*.h =.;h;..\A3\H;..\EXTACY\RTS;..\C600\INCLUDE;..\WINDEV\INCLUDE\
```

#### *Recommended Compiler options*

For Windows:

```
-c -nologo -G2sw -Zp -W0 -AL -Os -D_msdos -D_WINDOWS
```

For debugging, add:

```
-dDEBUG -Zid -Od
```

### Batch Files

Some batch files are included with the **Amadeus-3** distribution. These are designed for use with 4DOS, a COMMAND.COM replacement that is *HIGHLY RECOMMENDED*. It also exists for OS/2 under the name of 4OS2 and is distributed as Shareware. If you don't have it, please install the shareware version included with the **Amadeus-3** distribution and do order it as soon as you're convinced !

The following batch files assume that your start-up command files (AUTOEXEC.BAT, 4START etc.) set certain variables to indicate your drive configuration. The following are required: sys, bin, dev, mem.

The file \BAT\SETENV.BAT is a good example:

```
set bin=c:
set sys=c:
set dev=d:
set aux=f:
set dos=%sys%\os2\mdos
call %bin%\bat\setvars
if not (%_4ver%) == () call %bin%\bat\setalias
```

"sys" is where the DOS or OS/2 system files and directories are located.

"bin" is the drive that contains the utility programs (\BIN\, \BAT\, editor directory etc.).

"dev" is the drive where all the development files are installed, i.e. the compiler and the application development directories, for example:

```
\A3\, \EXTACY\, \WATCOM\ etc.
```

"aux" may occasionally be used for locating other programs and directories.

"dos" indicates the system utilities directory.

\BAT\SETVARS.BAT is an example of a dynamic system configuration. It adds required directories to the path (note the use of the 4DOS Index function) and attempts to assign the RAMDISK and CD (for CDROM) variables, as well as initialising other system variables.

\BAT\SETALIAS.BTM is a 4DOS specific batch file that loads keyboard alias definitions.

\BAT\X2C.BAT simply calls \EXTACY\BIN\X2C.BAT, as EXTACY is not installed on the path when you start the system.

\EXTACY\BIN\X2C.BAT loads all EXTACY and **AMADEUS-3** specific variables and calls the compiler batch file, for example WATCOM.BAT:

If you run under DOS, your AUTOEXEC.BAT file should call SETENV.BAT, which in turn calls SETVARS.BAT and SETALIAS.BTM.

In an OS/2 DOS box, set the DOS\_AUTOEXEC option to a file that will call the SETENV.BAT file and set the DOS\_SHELL option to C:\4DOS\4DOS.COM C:\4DOS. Specify another DOS\_AUTOEXEC file for a Windows session and yet another file for programs that should run without 4DOS.

Example:

AUTOEXEC.BAT for your 4DOS tasks

AUTOWIN.BAT for Windows sessions

AUTOSTD.BAT for standard programs that work with the basic COMMAND.COM

If you work under OS/2, I would also recommend that you increase the number of lines in a DOS or OS/2 text mode session used for program development. Most text editors, including the famous BRIEF, are able to adapt to a larger number of displayed lines.

## 10. Environment

This section is only of interest to those who work with the Extacy Oberon-2 to C compiler under DOS/Windows or OS/2 for Windows 3.x. You don't have to read it, if you use the XDS programming environment or some other RAD system, which integrates all functions, such as make, compile, link etc.

Here is a description of the various environment-related topics, command files included with Amadeus-3 etc.

As a reminder: Under DOS, OS/2 and Windows NT, the command line processor replacements 4DOS, 4OS2 and 4NT are *required* if you wish to use any of the following command files. These tools are included as shareware with Amadeus-3, but we strongly encourage you to purchase complete licences, as the US\$20 or so will be very well invested money.

Under UNIX/LINUX, this is obviously not required, as on this type of operating system, the shell game is already well established, though by no means uniform.

Further useful tools that are standard on UNIX but have to be added to the other operating systems include the following:

Tool name	Description
<b>Total Commander</b>	Previously called "Windows Commander". A really phantastic File Manager; don't work without it! It supports very comfortable FTP management as if you were working locally, plus directory synchronisation, file compression in multiple formats, global search and multi-file rename. Highly recommended, it's worth every cent of the \$20 licence (although the shareware version is fully working and unrestricted, this one deserves to be bought!). Get it from <a href="http://www.ghisler.com">www.ghisler.com</a>
<b>Multi-Edit</b>	A great programming editor, which follows a long tradition of powerful editors. Under DOS, the best of it's kind was certainly Brief and of course, Multi-Edit has templates for the Brief key layout and is fully configurable, supports version management etc. It's programmable, very fast and comfortable. Highly recommended. Get it from <a href="http://www.multi-edit.com">www.multi-edit.com</a> .
<b>grep</b>	A tool for text search operations, allowing you to find any string (usually with extended pattern matching) throughout an entire set of source code files. Very helpful when you need to know where certain identifiers are referenced etc. Under 4DOS /4OS2/4NT, you may also use the FFIND command.
<b>awk</b>	A complete text processing programming language that can be operated from the command line or through scripts. Allows very advanced string searching and manipulation.
<b>data compression such as zip/unzip</b>	If you have been in the computer field for any length of time, you already have an appropriate set of compression utilities. The most common ones under DOS & OS/2 are PKZIP/PKUNZIP, ARJ, LHZ.

## 10.1 Compiling and linking

The compilation may vary according to your platform, but basically, it works like this:

- first invoke the Oberon-2 compiler's make facility
- for Oberon-2-to-C translators, you then call the backend C compiler
- on some systems, you may have to invoke a GUI resource compiler
- finally you have to link your application to produce the executable.

The last step is not required on pure Oberon-2 systems, which are not really the target of Amadeus-3. On all major platforms, linking remains necessary.

### 10.1.1 Compiling with XDS

XDS works with several configuration files you should know about:

- XC.CFG the configuration file. Contains standard parameters for various pragmas, file paths etc, cf. XC.RED, which takes precedence over the XC.CFG file. Is usually located in the XDS\bin directory, along with the executables, but you may override it with a different version in the project directory.
- XC.RED the redirection file. This file may be placed in the project directory or in the XDS\bin directory, where the compiler binaries are located. The file in the project directory takes precedence. It specifies the search path for each file type – source code, symbol, object and resource files.
- PROJECTNAME.PRJ the project file, which defines a whole range of pragmas, file paths, source files to be compiled and libraries to be included. Specifications in the project file take precedence over the definitions in the XC.RED file
- You may also specify additional parameters, pragmas etc. on the command line, which take precedence over all the previous settings.

An option can be used to specify if existing files are to be replaced (OVERWRITE+/-). If true, existing files are replaced where they are found. If the generated file does not exist yet, it is automatically placed in the local directory.

*>> That means that you have to move all sym and obj files to their intended directory manually for all new modules!*

Later on, when the sym and object files exist and are placed in the right directories, you don't have to move them anymore.

To compile an application, the best option is to create a project file and then run the compiler with the command

```
xc =pr project_name.prj
```

You may use different project files with different options, e.g. to generate a program version for debugging or release code.

When generating a new project with A3Edit, 2 standard project files with typical options for A3 applications are included. They will be named project\_name-dbg.prj and project\_name-opt.prj for Debug and Optimized.

When you want to recompile the entire project (including all libraries and imported sym files), use

```
xc =pr =all project_name.prj
```

This may take some time, so you may not want to use it often..

## 10.2 Sample XDS project file for A3Edit

Here are some sample XDS project files with the typical settings and libraries used by Amadeus applications:

### 10.2.1 Debug mode

```
% XDS project file
-DebugWrt:-
-GENDEBUG+
-GENHISTORY+
-GENPTRINIT-
-NOPTRALIAS+
-PROCINLINE-
-DOREORDER-
-NOOPTIMIZE+
-CHANGESYM+
-OVERWRITE+
-CPU = Pentium
-GUI+
-VERBOSE+
-XCOMMENTS+
-GCAUTO+
-COMPILERHEAP = 160000000
-COMPILERTHRES = 8000000
-STACKLIMIT = 60000
-HEAPLIMIT = 20000000
-GCTHRESHOLD = 3000000
-MINCPU = Pentium
-ALIGNMENT = 1
-ENUMSIZE = 2
-SETSIZE = 4
-M2EXTENSIONS+
-O2EXTENSIONS+
-O2NUMEXT-
-BSTYLE = DEF
-GENFRAME+
-WOFF+
-WOFF302-
-WOFF304-
-WOFF310-
-WOFF311-
-WOFF312-
-WOFF313-
-WOFF316-
-WOFF317-
-WOFF318-
-WOFF319-
-WOFF901-
-WOFF902-
-WOFF314-
-WOFF320-
-WOFF321-
-WOFF322-
-WOFF323-
-WOFF910-
-WOFF911-
-WOFF912-
-WOFF915-
-WOFF913-
-WOFF315-
-WOFF300-
-WOFF303-
-WOFF305-
-IOVERFLOW-
-COVERFLOW-
-FOVERFLOW-
-CHECKINDEX-
-CHECKDINDEX-
-CHECKRANGE-
-CHECKSET-
!new linker = xds
!new exename = A3Edit
!module A3edit.rc
!module A3edit.ob2
!module A3edit.res
!module TX32.lib
!module wbtrv32.lib
!module vic32.lib
!module htmlhelp.lib
!module ODBC32.lib
!module ODBCCP32.lib
```

### 10.3 Optimized code

```

% XDS project file
-DebugWrt:-
-GENDEBUG-
-GENHISTORY-
-GENPTRINIT-
-NOPTRALIAS+
-PROCINLINE+
-DOREORDER+
-NOOPTIMIZE-
-CHANGESYM+
-OVERWRITE+
-CPU = Pentium
-GUI+
-VERBOSE+
-XCOMMENTS+
-GCAUTO+
-COMPILERHEAP = 160000000
-COMPILERTHRES = 8000000
-STACKLIMIT = 60000
-HEAPLIMIT = 20000000
-GCTHRESHOLD = 3000000
-MINCPU = Pentium
-ALIGNMENT = 1
-ENUMSIZE = 2
-SETSIZE = 4
-M2EXTENSIONS+
-O2EXTENSIONS+
-O2NUMEXT-
-BSTYLE = DEF
-GENFRAME+
-WOFF+
-WOFF302-
-WOFF304-
-WOFF310-
-WOFF311-
-WOFF312-
-WOFF313-
-WOFF316-
-WOFF317-
-WOFF318-
-WOFF319-
-WOFF901-
-WOFF902-
-WOFF314-
-WOFF320-
-WOFF321-
-WOFF322-
-WOFF323-
-WOFF910-
-WOFF911-
-WOFF912-
-WOFF915-
-WOFF913-
-WOFF315-
-WOFF300-
-WOFF303-
-WOFF305-
-IOVERFLOW-
-COVERFLOW-
-FOVERFLOW-
-CHECKINDEX-
-CHECKDINDEX-
-CHECKRANGE-
-CHECKSET-
!new linker = xds
!new exename = A3Edit
!module A3edit.rc
!module A3edit.ob2
!module A3edit.res
!module TX32.lib
!module wbtrv32.lib
!module vic32.lib
!module htmlhelp.lib
!module ODBC32.lib
!module ODBCCP32.lib

```

## 10.4 Debugging

XDS comes with a very good text mode debugger for Windows, which allows you to set break points on variable access, is able to represent object structures correctly etc. Other debuggers may work perfectly well, as Oberon-2 compilers usually generate standard debugging information on each platform.

For modules that you want to debug in detail, you should add the following parameters right before the MODULE identifier (for XDS):

```
<*NOOPTIMIZE+*><*PROCINLINE-*><*DBGNESTEDPROC+*>
```

Remove this line when generating code for the final version for distribution, after testing it thoroughly (!), as there may be subtle differences introduced by the code optimizer. It's rare, but it does happen...

The command line for the XDS debugger is simply

```
xd program_name parameters
```

Beware of the following:

- The XDS debugger, working in text mode, is dependent on the available output area and will give an error message if the current window layout does not match the available area it was designed for. This layout is stored in the XD.CFG file, so you may want to keep a few copies handy for the different display types you may have to work with (XGA, SVGA, UXGA etc.), as on larger displays, you probably want a layout that allows you to see a lot more information simultaneously. The A3 supplemental files for XDS include some such specifications.
- Warning: there is a bug in the debugger (well yes, that does happen...). When you save a new layout, always store it in the current directory first, as the debugger will crash if you try to select a different one (true up to XDS version 2.5 at least). Just move the file later from the application directory to XDS\BIN to apply the new layout to all applications.
- Also note that the debugger often stops with error messages inside Windows or external library code while running the initialization part of many applications, although the code will run just fine. Apparently, that code generates exceptions which are intercepted by the debugger, although they are not fatal and handled properly by the external library code. Just continue the execution until you reach the piece of your code you are interested in.

## 10.5 Working with Multi-Edit

Multi-Edit is the recommended environment for working with Amadeus-3. You can get a pre-configured installation from Amadeus Software with the following setup:

- Oberon-2 and Modula-2 support, with syntax highlighting and templates
- Pre-configured project files, which are set up so that they will work with most applications in the A3 default environment
- Standard commands and key assignments for the most common tasks. NB: All of the following commands must be run while editing a file in the active project directory or they will not work. This is due to the generic nature of the standard macros, which refer to relative paths:
  - Ctrl-F9 = Compile current project for debugging
  - Alt+F9 = Debug current project for distribution (release)
  - Ctrl-F10 = Debug current project
  - Ctrl+Shift-R = Run application
  - Ctrl+Shift-A = Run application in admin mode
  - Ctrl+Shift-3 = Run A3Edit and load all TX3 files in project directory
  - Ctrl+Shift-L = Load the licence editor with the data from the project\licence directory
  - Ctrl+Shift-K = Load the licence editor with the data from the development licence file (i.e. the licence.dat file in the project directory)
  - To build the application (i.e. re-compile every file irrespective of time & date), use the menu entries Project / Build Release; Project / Build Debug.

NB: When you move the development environment to a different directory or another machine, make sure you recompile everything with the Build command or you won't be able to debug the application as the debugger won't find the source files.

## 10.6 Working with Notepad++

The Obide (Oberon IDE) plugin for Notepad++. Copyright (c) Alexander Iljin, 2008.

### 10.6.1 DESCRIPTION

This project was started at "Amadeus IT Solutions" to create an IDE that would support our development process, tools and project structure. The plugin is not meant to be a general purpose tool, configurable for every possible use cases. It mostly meant to provide functions not found in other plugins.

### 10.6.2 FEATURES

The following functions are currently supported:

#### 10.6.2.1 Source code navigation.

Place the caret at an identifier, then press Ctrl+9 to jump to the declaration of the identifier. If the identifier is an imported module name or alias, the corresponding file will be opened and/or activated in Notepad++. Press Ctrl+0 to jump back to the previous caret position (this also restores selection).

The following identifiers are currently supported as jump targets:

- top-level, nested and type-bound procedures;
- global and local constants;
- global and local variables, including procedure parameters;
- global and local types and record fields;
- imported module names and aliases.

### 10.6.2.2 Source code information.

To get some additional information about an identifier, place the caret at the identifier and press Ctrl+Alt+Space. If there is any information available, you will see a popup hint window. The following kinds of information are provided:

- for constants: value, as declared;
- for variables: type;
- for types: type declaration;
- for procedures: parameter list and return value.

"As declared" means that no calculations or translations are performed. e.g. if a constant X is declared as "CONST X = 5+4;", you will see "5+4" in the hint, not "9".

### 10.6.2.3 Autocompletion.

The autocompletion allows you to select an identifier from a list of identifiers, available in a given context. To make use of the feature, start typing and press Ctrl+Space. An alphabetically sorted list of identifiers is popped up for you to choose from. The following items are included:

- global and local constants;
- global and local types;
- global and local variables, including procedure parameters;
- top-level and local procedures (excluding type-bound).

If you are calling the autocompletion list for a qualified identifier you will see the list of available subidentifiers with full respect to the visibility rules and type extension. The left part of the qualified identifier in question must not contain any procedure calls.

### 10.6.2.4 Compiler integration.

Currently Obide supports very basic compiler integration. Press Ctrl+1 to compile the current project. This is accomplished by executing the "make" program in the folder of the currently active file. The output of the program is analyzed and the XDS compiler error messages are searched for. If there is an error message found, the offending module is opened and/or activated, the caret is placed at the error position and the error message is displayed in a popup hint window. You may press Ctrl+0 to jump back to the previous position.

If no XDS error messages were found, the "Compiled successfully" message is displayed to indicate that the compilation is finished. Note, this does not mean there were no other kinds of compilation problems. If something does not go as expected, you may still want to execute "make" manually in a console (or use the NppExec plugin).

Press Ctrl+- to jump to the place of the last compiler error and show the error text without recompilation. Press Ctrl+0 to jump back.

## 10.6.3 INSTALLATION

To install the plugin simply copy the Obide.dll to the "plugins" subfolder of the Notepad++ installation folder.

## 11. Annexe

Additional information regarding Amadeus-3, Oberon-2 and programming in general.

### 11.1 Applications developed with Amadeus-3

The following applications were all developed with Amadeus-3. This should demonstrate that there is little that cannot be done with this development system and that Oberon-2 code can fit all kinds of demands and environments:

- AMADEUS CONFIDENTIAL (AC): a CRM (Customer Relationship Management) written for the Royal Bank of Canada. The first version was implemented under Amadeus-2 and Modula-2 from 1990 to 1993. Since then a new, Windows-based version under Amadeus-3 and Oberon-2 has been in use, handling the highly sensitive data of numbered, private banking accounts in Switzerland.
- AMADEUS PORTFOLIO (AP): A full portfolio management system, supporting customers with multiple bank relations, multiple currencies, able to instantly calculate and display the exact status of a portfolio or a combination of portfolios for any date of the present or the past. May be combined with AC, sharing some data, but keeping personal and financial data clearly separate.
- EXPERT SPIDER®: A graphical quality control system developed for Du Pont de Nemours, to encourage internal and external customers and suppliers to use scientific methods for measuring the properties of various products. Fully configurable, the system is able to display data in the form of various charts – especially spider- or radar-charts – of a whole set of parameters. The acquisition of parameters can be documented with various tools, including videos and full descriptions.
- SKIP, ALTM, REALM: Applications for airborne laser topography, developed by TopScan GmbH, a German company, in cooperation with Optech Inc. from Canada. This is real high-tech software, that performs the data analysis and presentation. The user interface remains very useable, despite a huge number of parameters that need to be set for GPS data, Laser configuration, geographic parameters etc. The software is in worldwide use, from Europe to the USA, from South Africa to Japan.
- BMAN®: Software used by Du Pont de Nemours in its Ballistic Laboratory in Geneva, to test ballistic protection equipment made of Kevlar and other fibers. Police, Military and other exposed personnel throughout the world who need body armor very likely use products that were tested with the help of Amadeus Software since 1993.
- HP Telecom '95 Scheduling System: Telecom'95 was the 1995 edition of the biggest telecommunication related trade fair in the world, taking place every 4 years in Geneva. This application was used by Hewlett Packard to schedule the entire event: meeting agendas for their top managers, conferences, presentations, room occupation and staff organisation. During preparation, up to 80 people were entering data over the internal network and during the event, 14 terminals were permanently active on the HP stand.
- TM: Complete Intellectual Property Management for Trademarks, Models and Patents. Used to track all intellectual property for companies and individual owners, with national and international registration and a complete calendar for tracking deadlines for payments, renewals and other required administrative procedures. A smaller utility, TM AGENDA allows simplified tracking of deadlines.
- KSTOCK: Stock management and European distribution system operated for DuPont, then Invista, for their marketing and technical documentation for LYCRA® and other famous brands.

### 11.2 Common File Extensions

The following file extensions are commonly used throughout this documentation and the Amadeus-3 Object Oriented Framework.

<b>Extension</b>	<b>Usual contents</b>
<b>OB2</b>	Oberon-2 Source File
<b>RC</b>	Windows Resource File, specifying bitmaps etc. to include with the executable
<b>PRJ</b>	XDS Project File
<b>RED</b>	XDS Redirection File
<b>INI</b>	Windows INI File, used to store resource strings
<b>AP3</b>	Amadeus-3 Binary Object Script, defining mostly user interface objects etc.
<b>TX3</b>	Text version of AP3 file; may be edited; use AP3 when modification is not desired
<b>BX3</b>	TX3 Backup File; copy made by A3Edit when backup was requested
<b>TXT</b>	Simple text file, usually containing notes etc.
<b>FMT</b>	Report format file, containing a program in Amadeus-3 Report format
<b>INC</b>	Report include file; only contains definitions and procedures
<b>DAT</b>	Database file, by default generated with Pervasive.SQL
<b>DFI</b>	Object script, defining contents of DAT file of same name
<b>BAK</b>	Database backup file, generated automatically when DAT file format changed
<b>DFK</b>	Object script, defining contents of BAK file of same name
<b>OBJ</b>	Object file generated by compiler
<b>SYM</b>	Symbol file generated by compiler
<b>DLL</b>	Dynamic Link Library, as used by Windows
<b>JPEG, JPG</b>	Standard for photographic quality bitmaps with lossy compression; file extension
<b>GIF</b>	Protected standard for bitmaps with lossless compression in 256 colours; file ext.
<b>BMP</b>	Standard Windows and OS/2 bitmap format; file extension
<b>PDF</b>	Portable Document Format, as used by Adobe Acrobat®

## 11.3 Object Commands

Here is an inventory of the various object commands by object class. This list may not be exhaustive or complete, as the supported commands evolve. In case of doubt, please check the corresponding module.

### 11.3.1 Persist.Object

All persistent objects answer to the following commands:

- NAME : returns the object name as answer
- CLASS [ n ] : returns the class name of the object or it's n-th parent class

### 11.3.2 Persist.Container

- SUBNAME : returns the contained object's name as answer
- SUBSHORT : returns the contained object's short name as answer (content of name field)
- SUBCLASS [ n ] : returns the class name of the contained object or it's n-th parent class
- FIND class\_name object\_name : searches for specified object and assigns it to the container

### 11.3.3 WinMgr.Window

- RESET INIT | ZERO | DEFAULT | MIN | MAX = Call reset with the corresponding parameter
- SHOW | HIDE = show or hide the window
- PROMPT = prompt for input

### 11.3.4 Values.DictEntry

- CLEAR : clear the entire dictionary structure
- DEFAULT: set the entire dictionary structure to their respective default values.

### 11.3.5 Values.Value

- GET [ index ] : return the contents of the value with specified index in standard string format
- SET index value\_string : set the contents of the value with specified index to the string
- ARRAY value : Set the number of array elements to the specified value (dynamic!)
- ISEMPY index : return TRUE if specified array element is empty

### 11.3.6 Db.Database

- OPEN [ NEW ] : opens the database; returns name of first file that failed if unsuccessful
- CLOSE : closes the database
- COPY path : copies the database to specified path, eg. default database; does not overwrite existing files
- SCRATCH [ BAK ] [ DIR ] [ DFI ]  
erase database; BAK: erase backup files; DFI: erase DFI files; DIR: also erase directory, if empty if neither BAK nor DFI specified, erase all files, including data files
- LIST print name to debug file

### 11.3.7 Db.File

- GET FIRST|LAST|NEXT|PREV|EQUAL|GT|GTEQ|LESS|LESSEQUAL [BY key] : Access file by specified key
- GETID id\_variable\_name [index] : Access file by ID key and value found in specified variable
- PUSH | POP : Save / Restore file status
- RESET : Reset file access variable contents and position
- INSERT : Insert a new record

- UPDATE : Update the current record

### **11.3.8 Sequence.Source**

- SETINFO NEWFILTER: set filter information for new data
- SETINFO CLEAR : clear filter information

### **11.3.9 Sequence.Stepper**

- FIRST | LAST | NEXT : access data source; read either first, last or following element
- ASSIGN 'sourceName' : search for source in all open projects and assign it to stepper
- ANYTAG TRUE if any item in sequence is tagged
- TAGGED TRUE if current item in sequence is tagged
- SETTAG SET tag for current item
- CLRRTAG CLEAR tag for current item

### **11.3.10 Bitmaps.Object**

- PAGE [ number ] | [ variable ]  
Load specified page number for this image file
- FULLSZ [ ON ] | [ OFF ]  
if ON : Fully resize picture according to object extent  
if OFF: Don't resize picture to fit in object extent

### **11.3.11 GlobalDb.Object**

- CLOSE or SCRATCH  
Close all database files that are members of GlobalDb.Object

### **11.3.12 MemList.List**

- GET FIRST|LAST|NEXT|PREV variable | record  
Access list
- COUNT  
Return the number of items in list
- POS  
Return the current index in list
- APPEND variable | record  
Append the contents of the specified record to the list
- INSERT n variable | record  
insert the contents of the specified record to the list at index n (zero based)
- DELETE n  
delete the item at line n

### **11.3.13 Scroll.Mask**

- START call OnStart
- SAVE call SaveData

## 11.4 Typical errors, pending issues and work-arounds

Here are a few tips from practical experience, which are not only related to programming itself, but also to environment-related problems, which could use up a lot of your time. As problems arise which have a code solution, they are usually integrated into the Amadeus-3 library, but frequently, this just isn't possible or practical. In any case, it's worth being aware of what you could run into and how to deal with these problems.

### 11.4.1 After compilation, make sure the linker runs successfully

The linker cannot replace the previous version of an executable if the file is still being accessed, e.g. because it is still running, maybe even over the network.

### 11.4.2 Make sure the data is correct

Before you spend huge amounts of time, trying to debug the logic of your code, always make sure that the data is correct, i.e. in the format you expect. This is a fundamental rule not only for work with databases. More often than not, the heart of the problem lies not in the code, but in the data.

### 11.4.3 Database operations causing problems

One of the greatest features of Amadeus-3 is the automatic updating of database files by the application software, after you changed the data or key definition. While it is fairly safe, you are still well-advised to *check* if everything is working as intended.

The first common source of problems is an attempted conversion, while the database is actually locked. Always make sure that there are no locks on database files under development, e.g. because other users are using them.

If you are not sure if any format change is necessary, just watch out for any new \*.DFK files in the database directory after running the application with a test database, as these indicate a format change. If there is no matching \*.BAK file, then the database file could not be copied, probably because it was still locked.

There may also be problems with your key definition, such as incompatible key attributes etc. Check closely, what kind of errors the database driver signals.

If your application seems to work inconsistently, it may be because the database key structure does not really match what you expect. This can happen because of many reasons, including inconsistent copying of DAT and DFI files or incomplete format changes.

### 11.4.4 Window operations – Initial Window Placement doesn't work

When you set the attribute "make client-sized" MkClient (MKCLIENT in the object script), you cannot initialize the window size based on its frame coordinates properly, since the actual size will be computed when it is first created based on the client section of the window. Therefore, if you wish to insert a window into a global screen arrangement, it's better not to set the MkClient attribute.

### 11.4.5 Strange behavior during window updating

Object interactions may occur at many levels. In a true OO GUI, this can become quite a headache. Here is a story about a particularly puzzling scroll window update:

In the application under consideration, a scroll window is used to display a mixture of text and bitmap data. The bitmaps are represented through a class PictDsp that is derived from ValueDsp.Value. A matching Sequence.Item class stores picture data via an ImgFiles.Object.

The scroll window sometimes behaved strangely when reaching the last line, frantically updating the window. The source of the bug was in the PictDsp class: the list was not being filled completely with the last item, so Scroll attempted to fill it with another item at the top of the window. The corresponding item was retrieved and measured with the Dimensions method. The particularity of the class PictDsp is that it automatically scales all pictures, so that no list item exceeds a certain size. The line item is displayed through a static column object, which is always attached to the scroll

window, changing only its line coordinate. Unfortunately, the `Resize` method automatically invalidates the object's area if it is attached to a window. This prompted another call to the window `Paint` method, which in turn triggered another attempt at filling the list etc.

The solution was of course to set the `WinMgr.Hidden` attribute on the `ImgFiles.Object` for the `Dimension` method, then return it to its previous state, avoiding the invalidation of the corresponding window area. Hidden objects are not refreshed, `WinMgr.Object.Invalidated` does not do anything if the object is hidden.

## 11.5 Oberon-2 vs. C++

This chapter was added on "popular request". In fact, the original intent was to avoid the "Language War", but questions from various sources have clearly pointed out the need to discuss the respective features and shortcomings of these two languages, which become more and more direct competitors. First off, I want to clarify a fundamental question: is it possible to write good, functional, solid code in C or C++? Yes, absolutely. There are a lot of hackers (in the classical sense of "experts") out there, who write wonderful code in C and C++. But this is where the great motto of Prof. Wirth comes into play: "Programming Tools should lead to correct programs *naturally*, NOT through exceptional intellectual exercise!". If you are willing to sacrifice enough years of your existence and enough time and effort, you will certainly be able to create wonderful things with any tool, but is there no better way than to struggle with a difficult tool?

- Does the possibility of good C/C++ code mean that it is a good choice for the majority of programmers?
- Is it a good choice for professional applications that need to be completed within certain budget and time constraints?
- Or a good choice for applications that will have a certain life time, sometimes in excess of 10 years and that need to be maintained, with code shared among several programmers?

These are the questions we will examine in this chapter. We will do so with a particular emphasis on the Oberon-2 / C++ opposition.

For an in-depth critique of C++, please have a look at the following site:

<http://www.progsoc.uts.edu.au/~geldridg/cpp/cppcv3/sect2/index.html>

which discusses C++ and Eiffel.

Let me quote one excellent passage:

Chris Reade gives the following explanation of programming and languages.

*"One, rather narrow, view is that a program is a sequence of instructions for a machine. We hope to show that there is much to be gained from taking the much broader view that **programs** are descriptions of values, properties, methods, problems and solutions. The role of the machine is to speed up the manipulation of these descriptions to provide solutions to particular problems. A **programming language** is a convention for writing descriptions which can be evaluated."*

If you wonder whether it is relevant, just have a look at the state of software development after 50 years of practice. Despite all the theoretical progress, most code is still dangerously unstable, which is vividly pointed out by the failure of Microsoft to deliver an operating system that is even remotely as solid as the old multi-user systems from IBM and DEC after billions and billions of dollars thrown at Windows development, the numerous break-ins that use buffer overruns and other problems that are directly related to C/C++ code. A recent exploit (July 2002) against Outlook makes use of a specific weakness of C++ exception handling, associated with the usual abysmal memory management in this language.

In how far is the choice of programming language related to these problems? The usual answer by C++ programmers is "Why, not at all! There are no problems with our language, it's the best, it's perfect! How dare you doubt it? We're the majority anyway, so it doesn't matter what anyone thinks, we'll keep using C++ just as it is. Besides, programming is difficult anyway, the programming language can't possibly have an impact".

Considering the 2 to 3 years of investment to become proficient with C++, it seems obvious that no programmer wants to throw away his valuable investment. But maybe, just maybe you a smart programmer will re-consider and cut his losses, just as a smart investor won't wait until the value of losing shares drops to zero.

This documents is also for managers, who don't really have any technical basis for making decisions. It may seem dangerous for any business to invest into something that is not main-stream. On the other hand, very few people get rich by following the crowd. Maybe breaking free from some pre-conceived ideas will prove to be very profitable for your company and your career.

### 11.5.1 How important is Notation?

Fundamental !!!

Try using Roman Numerals for arithmetic calculations. Addition and subtraction are sort of ok, but when you get to multiplication and division, serious trouble sets in. You just can't do it efficiently. Which is probably why Romans were miserably bad at mathematics. Greeks also only excelled at geometry. They didn't have a proper notational system and they lacked fundamental concepts, such as the digit ZERO. Simple as it is, it opens entirely new horizons in mathematics.

$XVII * CIV / XIIX = ?$

Why should it be any different for programming languages? The way you transcribe concepts is hugely important. They also determine how you analyse problems. People who say that it doesn't matter are like people who think they can get a free lunch.

Did you ever notice how difficult it is to proof-read a document in English? Imagine that every punctuation was fundamentally important and that missing a single comma could change the meaning of the entire text. Not such a wonderful idea, right? Writing a single valid contract might be a multi-day effort, proofreading again and again.

Programming is a bit like that, but hey, that's where the computer comes in. It can proofread your code while translating it (that's what compilers do). Just make sure that any "sentence" of code is very precise. If there is no risk of changing the meaning of the code by changing a single character, then you should be doing fine. Even better, almost any random change of multiple characters should be rejected as invalid, before any harm is done. Not so for C/C++. Single-character mistakes are possible in many places and won't be detected by the compiler. (we'll get to the details later).

"No problem!" says the expert, "after all, we're not a bunch of idiots, we are serious programmers". Superhuman, quite obviously, since they're sure that they will do better than any proofreader of English - *and* understand all the semantic implications of each alternate valid spelling. And this not only while they are producing the code, but also when they read someone else's code that is 10 years old and "optimized". It's nice to know that programmers are such super-heroes.

Here is a very real example of catastrophic system failure due to a simple syntactical problem in a programming language:

The USSR Mars probe, released from the Earth's atmosphere on November 27, 1971, crash landed on the surface of Mars because its booster rockets failed. The failure was due to a single character mistake in the landing algorithm, which was written in Fortran. There was a breaking routine with a waiting loop of the style "FOR I=1,1000", which didn't perform properly, because the comma was replaced with a period, which was compiled as "FORI=1.1", variable "FORI" was assigned the value "1.1".

So yes, notation is important. The programming language is really the saw, hammer and nails of building software, with bits and bytes the raw material you shape. If your tools are crooked, it's much harder to do a good job.

In business terms, this translates into *huge losses* and *billions of wasted investments* !!! While there are many factors for the stock market crash, the inability of major companies to deliver the promised goods was certainly one of them. From there to blaming the popular use of a really bad tool, it's a short step. If you are a manager, you may want to think about it...

### 11.5.2 Objective criteria

This is a really difficult problem, since we tend to like what we know. Most people don't think about how to design a programming language, they just pick up a manual that says how it's done. Therefore, few people will say "Oh I just love this or that style of programming", they will like whatever they learned first, since that will save them a lot of intellectual effort - at least until they

are confronted with problems arising from that choice. If the first language learned was C/C++, as it still is for a large majority of programmers, then the C-style syntax, the operators *and* the traps and pitfalls of the language will seem perfectly natural, just as the words "Procedure", "Begin" and "End" will be natural to the Pascal programmer.

So I think it is necessary to give a little historic background to explain how I came to choose Modula-2 and then Oberon-2, hopefully making it very clear that my choice was based entirely on rational criteria and experience, not on habit or "religion".

I certainly didn't stick with the first thing I met, else I would still write machine-code, since I started programming of sorts in 1975, laid hands on something resembling an actual computer in 1978 and wrote my first professional applications in 1983. I passed successively through assembler, Basic (in various dialects), programmable calculators - even wrote full-fledged professional applications for the famous HP-41 - and pocket computers in HP Basic and Forth. One had to get along with very little resources, in those days. In just 17 KB of memory (that's 17 Kilo Byte, NOT Mega Byte), I managed to implement a system for field acquisition of data for high tension power pylons, which were then plotted out as terrain profile with obstacles, wires and all the associated calculations and numbers. Programs today use more memory than that just for the logo.

I've also used C for a while (on an early "multi-user" UNIX system, which ran on an 8 MHz Motorola 68000 with 1MB RAM and 40 MB hard disk - basically a Mac Classic with Unix, just a lot more expensive), UCSD Pascal (a bit like a Java Virtual Machine - Sun sure didn't invent the concept), Prolog (to play with), Lisp (wrote a Lisp interpreter in Forth) and Turbo Pascal. I also programmed many batch files on various systems, scripting, AWK and many other programming tools.

### 11.5.3 Choosing Modula-2 over C

In 1987, I had to implement a large application to run medical offices and needed a robust environment that would make team programming easier, that was efficient and stable and powerful enough to handle a large application. These were MS-DOS days, with 640K machines, but since memory wasn't used for visual gadgets, you could squeeze quite a bit of functionality into that memory.

Since I knew C and the book "Traps and Pitfalls in C and how to avoid them" and had experienced most of them first hand, it was totally out of a question to use that macro-assembler. Fortunately, Logitech (yes, the company that sells mice and joy sticks) had just started selling version 2 of its Modula-2 compiler, which turned out to be very stable and also had many nice tools, especially an excellent linker and a real debugger. Within less than a year, I wrote Amadeus version 1, , some applications for DuPont *and* the application Medisoft II, which was used for years by the two medical emergency services of Geneva, as well as by many doctors. The company I wrote it for - Gespower New Technology SA - expanded into the european market and now has over 400 customers in the medical field.

Modula-2 fully lived up to the expectations it raised. It was possible to produce good code quickly, to split the development effort through the proper use of modules, with abstract interfaces, information hiding etc. Even as of now, some Modula-2 applications written with Amadeus-2 in 1988 are still being used and can be maintained and changed. While many things have changed, including coding style - it's funny to see how narrow 80 column text seems, when you work on a 1600x1200 graphical display - the source code is as readable and understandable as the first day.

### 11.5.4 Choosing Oberon-2 over C++

From 1993 to 1995, I was involved - among other projects - with graphical presentation software. Back then, Windows 3.x was out and sort of running, but still too resource-intensive for the average portable computer of the day, so I wrote a tool for graphical presentations and a big presentation project with Modula-2 and the GX Graphics library in about 3 months. The contents was delivered by graphics experts. Given the usual hardware improvement, the following year our customers asked us to port the application to Windows, so I had to choose new tools for a new platform.

I could have stayed with Modula-2, but I had strained it to capacity with pseudo-object oriented design and really needed to move on. Programming Language technology had evolved and true

Object Orientation was a must for a GUI system. C++ was just becoming really popular, back in those days and naturally, I evaluated it fully. Don't think I rejected it off-hand, just because it resembled C. I had taken note of it's existence early on, around 1985, but for a long time, there were simply no compilers available for this complex language - just like Ada, which I also looked at in 1987. Some company had tried to sell an Ada compiler, but it was so huge and consumed so many resources, that it required a special add-on board, with a faster processor and tons of memory (6 MB, if I remember it correctly), bringing the cost of the compiler to thousands of dollars. C++ would have required a similar setup, which already raised my doubts as to it's suitability as a production tool.

Faster machines and more memory in 1993 made it possible to implement C++ compilers for PCs, though these compilers were still huge, slow and more often than not - incorrect. There was not a single standard of the language either and often, it was a matter of experimentation to find out what a particular compiler would do to a given piece of code. Worse than all of these problems, which could be overcome with time, were the deep-rooted design flaws, to which we will get shortly.

Naturally, I had also followed the progress of Oberon since it was officially released by Prof. Wirth and participated at the Oakwood industry standardisation conference for Oberon-2. In 1994 I finally took the plunge and designed Amadeus-3 for Windows from the ground up, taking along just a few things from Amadeus-2, such as easily convertible modules and the best concepts. After 3 months, it was sufficiently evolved for the first professional applications. The presentation project with similar contents as the previous one was now fully implemented in less than 1 month, a gain of about 4 weeks. Object Orientation and Oberon-2 were already showing their potential.

What I really liked above all was the small size of the language, along with it's expressiveness and incredible potential. Being experienced in programming and design, I didn't look for a large number, but for the proper set of features. What counted more than anything was the clear and precise definition of the language, the availability of correct compilers, the integration of significant advances in programming language design since 1980 and naturally the readability of the source code.

### 11.5.5 Oberon-2 is MUCH MORE THAN an advanced Pascal

Some people seem to think that Oberon-2 is to Pascal as C++ is to C. This is nonsense. While they were all created by Prof. Wirth and there is a superficial resemblance based on a few keywords, Oberon-2 is a radical departure from both, Pascal and Modula-2. There is no way an Oberon-2 compiler would recognize a Pascal program. Is that a problem? Not at all. Pascal was never intended to be a production language, although it acquired this status on it's own, but only through various dialects.

Oberon-2 was intended to be a full strength production language and was used to implement entire operating systems. Low level features are fully integrated into the language, they are clearly marked by the use of module SYSTEM. Hence it is easy to isolate potentially problematic code from easily validated modules, which is impossible in C++, where any given piece of code might wreak complete havoc with your system, even unintentionally.

Oberon-2 is actually **smaller** than Pascal or Modula-2, yet much more expressive. Most of it's strength is hidden in it's exceptional design. To name but a few:

- The object oriented features are very discreet at the syntactical level, you could almost miss them entirely, yet they are as powerful as in any other object oriented language.
- You don't *see* the Garbage Collector (GC), it simply works it's magic for you and empowers you to write code that you would never dare writing in C++, for fear of memory leaks, dangling pointers and other horrors of dynamically unsafe languages.
- Modules seem insignificant enough, to the point where some people with little knowledge of language design and the actual implications think that it's just files with headers, yet their meaning far surpasses the combination of include files, name spaces and classes from C++ put together.

As for backward compatibility, you don't use Oberon-2 to continue an application written in Pascal. You probably completely re-design the system instead, with a new perspective and the new possibilities that Oberon-2 and object oriented design offer. This is why I just don't get why some people seem to think that the compatibility of C and C++ would be so important. Why in the world would anyone want to mix old, strictly procedural, and modern, object oriented source code? I can imagine that one would want to re-use existing libraries, but that can be done through object linking, DLLs and other code sharing methods. One can indeed share C, Pascal, Modula-2, C++, Visual Basic and Oberon-2 code in this way. When you buy a DLL, you probably don't even know what language it was written in - and why should you care, if it works?

### **11.5.6 Oberon-2 is NOT in the same family as Ada or Eiffel**

This is another myth that must be dispelled once and for all. On the risk of repeating myself: the fact that two languages use "Begin" and "End" statements instead of accolades {} does *not* place them into the same family. Not anymore than C++ and Java, which are actually very dissimilar, despite appearances. Java is much closer to Oberon-2 than C++ and Ada is closer to C++ than to Oberon-2.

To evaluate a programming language, one must be able to look beyond syntax. The most important aspect is of course the semantic power. Is it possible to express various concepts and algorithms without running into major obstacles?

And it is equally important to apply Occam's Razor: remove anything unnecessary or any duplicate functionality. If there are 2 separate ways to do the same thing in a programming language, then one of them is probably dead weight.

Although Ada and Eiffel are "safe" languages (to the point of paranoia, a bit like Java, but contrary to Oberon-2, which is safe but flexible), they are in the same category as C++ for being very large and complex, which is opposite to the approach taken by Oberon-2: make it simple and efficient, a bit like RISC micro-processors. Yes, one could say that Oberon-2 is RISC applied to software. And nobody would pretend that RISC processors are less capable than CISC ones for "lacking functions".

Don't ever think that you get features of a programming language "for free". If the language is large and complex, then *everyone* who wants to use it, even just to read code, has to learn the entire extent of it. If he doesn't, then there will be pieces of code he will be unable to understand. Or there will be things he won't know how to use. He may also be tempted to use "features" because they are there, not because they make sense conceptually. Compilers become much more complex with additional features. There will be an overhead even in the executable code generated by the compiler. The code may be incorrect or different from one compiler to another.

Again the analogy with RISC / CISC holds up: CISC processors are much larger than RISC ones, they consume more energy, generate heat and waste valuable space on the die even if their extended functions are hardly ever used and don't give much of an advantage at all.

### **11.5.7 Constraints, Safe code and Programmer Freedom**

Having done my share of hacking, I'm certainly not going to say that programmers should be severely constrained in what they may or may not do with their code. If you are heading a team of programmers, you may want to impose certain rules, but each programmer individually should be perfectly free to write whatever code he thinks is doing his job. So what's the point?

For some strange reason, C/C++ programmers still cling to the bizarre myth that their language is the only one that gives the programmer total freedom. I'd say it mostly deprives the programmer of control over his own code, if anything.. This myth goes back to the original definition of Pascal. It is strange that it should still hold so much sway, since even every commercial implementation of Pascal provided mechanisms to access the underlying hardware and do what the language didn't specify.

In Oberon-2, low-level access is part and parcel of the standard language. You can do pretty much anything you want, you even get generic parameters in the form of ARRAY OF SYSTEM.BYTE. You may use SYSTEM.PUT and SYSTEM.GET to access any memory location on the machine (if the OS allows it), SYSTEM.MOVE to move blocks of memory from one location to another etc.

With this one little constraint: all such uses are "imported" from module SYSTEM. In fact, the compiler knows module SYSTEM and doesn't call a procedure for a MOVE operation, it just compiles the corresponding instruction(s). But at the level of the programming language, any such access may be detected immediately. You can browse all the modules in your application to detect any access to module SYSTEM and treat them as unsafe, scrutinize them for flaws or replace them with safe code if possible. As project manager, you can ensure that only specifically authorized modules use calls to SYSTEM, ensuring that no one is pulling any strings he shouldn't.

There are definitely legitimate uses of low-level features, they just have to be contained and well-used, to ensure that they cannot introduce subtle, hard to detect problems into your code.

To make it short, despite it's static and dynamic safety, Oberon-2 is absolutely not limited any more than C++, when it comes to system programming or programmer "freedom". You gain the benefits without sacrificing any power.

### **11.5.8 Features, Features...**

So C++ is huge and just loaded with features... but how many of these features are actually being used?

Some time ago, I did some consulting work for Deutsche Bank and had a chance to work with their development team. When I browsed through the code of their main trading application, it took me a while to figure out that it was actually C++. It looked more like Java or in fact Modula-2, apart from the accolades. Talking to the project leader, I learned that they had very strict guidelines for programming:

- one instruction per line
- every instruction with comment
- NO Multiple inheritance
- NO Macros
- NO Generic modules
- NO Overloading
- NO fancy C-style expressions, only basic operators
- Naming conventions strictly enforced

Thanks to these guidelines, the team was able to function, even if there were some staff changes, which did occur frequently enough to be given serious consideration. Of course it meant that any newcomer had to undergo a complete training, to ensure the proper use of these guidelines. By what I saw, the effort to conform to local C++ standards was certainly no greater than the effort to learn Oberon-2 from scratch, assuming that we are dealing with well-trained programmers.

In fact, anyone who was able to actually master C++ will be able to learn Oberon-2 in no time flat, since it's a much easier language. Anyone who knows about object oriented programming and basic algorithmic will be able to become productive in Oberon-2 after only a few days. What does he gain? Certainly more features than the reduced set of C++ and many other benefits... including a great stress reduction, due to the fact that the language itself enforces all the above "guidelines". And if Deutsche Bank could get away without all of these nice "features", most other applications probably can as well.

## **11.6 Description of an exploit against Microsoft Outlook**

The following text describes an "exploit", which uses extensive knowledge of the internal defects of C++ programs, their exception mechanism and especially their miserable memory management, to corrupt Microsoft Outlook to the point of compromising the PGP Plugin:

"By creating a malformed email we can overwrite a section of heap memory that contains various data. By overwriting this section of heap with valid addresses of an unused section in the PEB, which is the same across all NT systems, we can walk the email parsing and eventually get to something easily exploitable:

```
CALL DWORD PTR [ecx]
```

This pointer address references a function pointer list. At the time of exploitation, an attacker controlled buffer address is the first item on the stack. By overwriting the function pointer list pointer address with the address of an Import table, we can call any imported function. Our current stack will be passed into the function for parameter use. The first item on our stack is an address that points to attacker-controlled data.

By overwriting the address with the address of the `SetUnhandledExceptionFilter()` IAT entry, execution will redirect into this address when the default exception handler is called.

After returning from `SetUnhandledExceptionFilter()` PGP, Outlook will fail as it crawls back down the call stack. After cycling through the exception list it will call the `DefaultExceptionHandler`, which now contains the address of our code. This can also be exploited silently using frame reconstruction.

### 11.6.1 Major flaws of C++

Contrary to Oberon-2, which is the result of true research by highly recognized experts of the domain, C++ was just an attempt to add features to C, while staying fully compatible with that old language. The author, Stroustrup, never pretended to have done extensive research to justify the various features he added. He included almost anything that was still under investigation through the menu. Multiple inheritance, Genericity, Operator overloading... all those nice features, that C++ programmers seem to cherish so much.

Simultaneously, he ignored major advances in program language design, even those that are recognized as fundamental to modern programming technology, most likely because of the "compatibility" issue with C. Given the popularity of C++, one wonders if old errors from 1969 and later from C++ will ever be removed... the fact that it's in the hands of a committee is not very encouraging. Committees tend to include the kitchen sink - after removing the water tab, to make it "safe".

Anyway, the role of the committee is pretty much limited to adding even more stuff, since it will be almost impossible to remove anything at all without a war breaking out. So C++ will forever stay very similar to C, dragging along all the famous pitfalls from that old language. Therefore, we have to assume that the following list of flaws will be valid for a long time to come.

Please also note that not all mistakes are unintentional. Any disgruntled employee who is looking for a good way to "make his boss pay" could introduce subtle and very difficult to find errors into an application's source code.

Problem (Type)	Description
No garbage collector (Lack of important feature)	<p>The absence of a GC is a significant defect, especially for a language which claims to be object-oriented. As code becomes sufficiently complex and dynamic, it is almost impossible to keep track of all the objects that are being passed around. An application that wanted to do this would almost have to write its own local GC. A true GC <i>cannot</i> be added to C++, except by reducing the language to a safe sub-set, something similar to Java. But in that case, it wouldn't be C++ anymore and most definitely not in the least C compatible.</p> <p>Since Amadeus-3 started life on top of a cross-compiler that produced intermediate C code and lacked a garbage collector, it became quickly evident that even with the most careful memory management, some problems got out of hand and were only corrected after moving on to a native Oberon-2 compiler with garbage collector. A3Edit, the GUI design editor is a typical example. It's impossible to predict how a user will move and copy objects, what links will be created. Drag &amp; Drop allows completely unpredictable re-arrangements of data structures. In the absence of a garbage collector, the only option for a C++ programmer would be to either spend huge amounts of time to control each data object with special code or to simply reduce the user interface to manageable proportions.</p> <p>Not having a GC limits the possibilities open to the programmer in very serious ways.</p>

Absence of modules (Lack of important feature)	<p>Modules are recognized as fundamental building blocks of programs since the late eighties. Modula-2 is an important - but certainly not the only - member of the family of modular languages. Ada has packages, so do Java and Eiffel. C++ has include files. So where's the problem?</p> <p>C++ has no concept of locality. Everything is global. When you declare a global variable, a procedure or a class in a CPP file, it's potentially in conflict with any similarly named variable in any other file that will be linked into the same executable. Modules provide a true name space "for free".</p> <p>The module is an important structural element, which is clearly represented as such in UML diagrams. A module is not just a file and a Class is most certainly not a substitute for a module. Relationships between classes can be established inside a module, voiding the need for "friend" relations. Dependent classes may be extended externally, if they are exported, which is not possible for child classes in C++.</p> <p>Abusing classes for encapsulation and as a replacement for true modules is a "high crime" in programming. It completely mixes up concepts, which leads to syntactic and semantic problems. It becomes impossible to properly separate entities in any analysis. Yet it is the only way to overcome this serious flaw in C++, leading directly to entirely new problems.</p> <p>Modules are encapsulation units, which may alternately hide or expose constants, types, classes, variables and procedures.</p> <p>Modules are fully self-contained units, with automatic initialisation. Termination can be added programmatically, since a termination procedure may be added during module initialisation.</p> <p>The fact of importing a module does not automatically make all imported objects visible to client modules.</p> <p>Moving an entity from one module to another actually has implications, since a module is more than just a container. Since it is a structural unit, the fact that an entity stems from module X or module Y has logical consequences. In C/C++, you may well use naming conventions, but when you move a class DB_Table from file DB.C to file SQL.C, this has no implications whatsoever. The lack of coherence won't generate any errors, nor force an adjustment of client modules, which may actually be affected by a semantic change as well.</p> <p>Each Oberon-2 module, when compiled, generates a symbol file, which is derived directly from the module code, not from an external header file. Therefore, no additional effort is needed to maintain the header or - as in Modula-2 - the definition file. The symbol file can be locked, rejecting the compilation if there is any change to the module interface, as it was previously defined. If dependent modules rely on a specific interface, then any change might break them and hence it is a very good thing for large projects if such unchecked changes are rejected.</p> <p>The symbol file also greatly increases the speed of the compiler. Any compilation that imports the Windows header files in C++ will be slowed down to a crawl, whereas Oberon-2 compilers can read in 20 header files, including the full Windows definition at breathtaking speed.</p>
---	--

<p>No multi-dimensional dynamic arrays (Lack of important feature)</p>	<p>Modern programming is all about <i>dynamic</i> programming. You don't want to allocate fixed-size tables and data structures, you want them to be dynamic.</p> <p>Tables are not one-dimensional all the time, they often have 2 dimensions (as for example the table you are presently reading), some times 3 and occasionally more.</p> <p>If each dimension of the table is potentially dynamic as well, which is frequently the case, then you have a major problem. You can either do everything with some other data structure, which is cumbersome or you have to define some specific access mechanism, such as overloaded bracket operators.</p> <p>We'll get to overloading later, for now just imagine all the code you have to drag around for managing elementary data structures...</p> <p>Considering the size of C++, it is a matter of disbelief that even such simple features are not properly supported.</p>
<p>Unqualified identifiers (Lack of important feature)</p>	<p>With the general, unqualified #include mechanism, it is impossible to know from which section of code - leave alone from which file - any given identifier comes from. It might be defined inside the same file, in the header file of the same C file, in an included header file or in a header file included by an included file... which is why C/C++ programs tend to use huge prefixes and very ungainly names for any identifier that appears in a header file. These prefixes are of course not enforced in any way, they may be used based on conventions. Misspelled or omitted prefix strings will not cause any kind of complaint from the compiler.</p> <p>Name spaces only alleviate the problem of conflicting names, they don't help you recognize every identifier in any given piece of code. The usual workaround is of course through utilities, such as databases of identifiers, help files and plain and 'grep' - searching all source files of the project for a given identifier.</p> <p>When you program in Oberon-2, this problem seems really alien. Whenever you look at a piece of code, you immediately know which identifiers are local to the module and which are imported. You also know from which module and don't have to go on a wild goose chase. The code immediately makes sense. Identical identifiers may be used in different modules. In that way, the local code can be very compact, with short identifiers, which are perfectly coherent within their specific context. Only when leaving the local module you have to use the fully qualified identifiers.</p> <p>This is especially invaluable when developing large, shared libraires and when mixing components from different sources. It is also absolutely necessary, when you read code that you have not been working with recently. You won't remember the source and meaning of every imported identifier.</p>

Multiple inheritance (Flawed feature)	<p>There are so many problems with multiple inheritance, we'll not get started on them here. You can find plenty of article for and against, presenting the different semantic obstacles, on the web. For now, I will concentrate on the conceptual problems with multiple inheritance and why it is totally unnecessary.</p> <p>The idea behind Object Oriented Programming is that you create logical entities, which map directly to real entities and emulate the attributes and behavior of this object, along with it's interaction with other objects. In an application for the management of a hotel, you may have an object "guest", an object "room", an object "reception" and an object "check-out desk". You can then create methods where the guest interacts with the reception, occupies the room and finally interacts with the check-out desk. The association of procedures and data turned out to be indeed a powerful one.</p> <p>But now we get a weird one: based not on observation of the real world, but on the "technically possible", multiple inheritance is introduced. Some lazy person thought: well, I have a class "Queue" and a class "Guest". I'd really like to use my existing "Queue" class with guests, so why shouldn't I create a combined class, that inherits from Queue as well as from Guest? That would be very elegant... and hence was born GestQueue and multiple inheritance.</p> <p>In fact, this is a very strange idea. There is no such thing in the real world as a GuestQueue. There are queues and there are guests. You can line up guests in a queue, that's it. You wouldn't describe something as a EngineBodyWheels, when you actually mean a car. And a car is <i>assembled</i> from various parts. You take an engine, some wheels and a body and you assemble them into a car. You can pick different wheels, a different body and a different engine and still assemble a working car.</p> <p>Inheritance makes sense when you have a basic concept, such as "wheel" and you want to make it more specific. A cart wheel and a sports car wheel have the same basic property of being able to roll, but they may have very different attributes, such as tires, suspension, disk breaks etc. Yet when assembling a car, it will still perform the one basic function that you need (in software... reality may be different).</p> <p>Introducing multiple inheritance was a very bad idea, since it makes people think in wrong categories. With good design, there is nothing you cannot achieve with proper assembly - and there are many ways to assemble object classes. MI introduces additional overhead for every call to a virtual method, which seems quite contrary to the C/C++ crowd's insistence on performance. It may cause serious semantic problems, when inheriting from multiple related classes etc. It adds sever semantic constructs to deal with those complexities. What's the point, when you don't even gain anything?</p> <p>Whenever you think you need multiple inheritance, you probably didn't analyse the problem properly. Go through your concepts again and see how to replace multiple inheritance with proper assembly and you will see that your design actually improves.</p>
--	--

<p style="text-align: center;">Overloading (Flawed feature)</p>	<p>Overloading seems like a really nifty idea. It gives the programmer the ability to declare his own operators and C-style programmers like operators, right? As most other "nifty" features, this is a deeply flawed one: there are only very few instances where operators actually make sense, such as for matrix operations or other mathematical functions. While nice for mathematicians, it certainly makes no sense from a programming point of view. In any other circumstance, program created operators are totally counter-productive.</p> <p>Why should "x * y" be any more readable than "x.Multiply (y)"? It is a method call - in the case of overloading, even a static one which cannot be overridden - but with overloaded operators not instantly recognizable as such, creating obfuscation.</p> <p>Using an identical name or operator for different functions is absurd. You could use the same operator, within the same program, for totally unrelated operations. In one case, you could define "x ^ y" for "attach x to y" and in another class it could mean "combine y and x". There is no way of telling. Assuming that people will make reasonable choices is not a good idea: what is "logical" is different fore everyone. A named method or procedure call is much better: "Queue.Attach (y, x);" or "x.Combine (y)" will be a hundred times more readable.</p> <p>Even worse, there is yet another possibility for introducing very subtle and hard to find bugs into an application. The selection of the proper overloaded operator occurs during compile time and it is a non-obvious process. In fact, several operators may match the same set of operands, for example because the overloading concerns different child classes of a same base class. Now let's suppose that at some point, a programmer decides to remove an overloaded operator. If the is another possible selection, it will be used instead of the previous one. This switch will happen without any warning or indication. Worse: it could happen because of an accidental or provisional change in the other overloading procedure's interface.</p> <p>And think about how you will find clients of the overloaded operator. Suppose the operator is "+". Is anybody really going to search the entire source code of a large project for all occurrences of the character "+"? In Oberon-2, you would just have to search for all occurrences of the method name, i.e. anything starting with an identifier, a period and the name of the method. Yes, you could choose bad names for your methods or attributes. The difference is that overloading implies that there will be more than one identically named operator.</p>
<p style="text-align: center;">Genericity (Flawed feature)</p>	<p>Object Orientation is inherently generic. Generic code in the C++ sense preceded OO. Ada used it already, so the two concepts are not linked. In fact, an OO language needs genericity like a fish needs a bicycle. In the best case, it is only a syntactic trick, a pre-processor "cheating" the compiler, in the worst case, it's a wonderful way to multiply identical code. It doesn't add any semantics and it doesn't perform any useful function.</p> <p>The usual argument is that it allows the re-use of standard functions, such as data structure management. This is only true for the most trivial cases, but as complexity of the task grows - such as when you need to sort or search the structure - genericity becomes useles, since it cannot know how to compare items of complex types, such as objects.</p> <p>True generic programming is Object Oriented, implemented via abstract classes. To insert any kind of entity into a data structure, you simply create a link object class, which knows about the entity you are inserting and all the appropriate methods to use for different functions. Standard link classes can be provided for basic types, if useful. This approach is much more flexible and powerful.</p> <p>Obviously, for basic types only, not objects, you can also write data structures that accept ARRAY OF SYSTEM.BYTE, but - to ensure type safety - use a typed interface for access and internal controls to ensure proper use. This will be strictly equivalent to the genericity provided by C++, but with greater control.</p>

Dynamic type recognition (Flawed feature)	<p>The only way to determine the run-time type of an object in C++ (and in Eiffel, for that matter) is to perform a type cast. You have to try to cast the object and you'll know if your guess was correct, if the result is not NULL (NIL).</p> <p>Considering the fact that most people seem to think that this is acceptable, one dares not think about how static their code must be... any serious object oriented application with a decent GUI interface will behave "erratically", so that it is impossible to know in advance , say, what sub-class of the global class for display objects will have to be handled. Oberon-2 allows for proper testing, followed by a type cast, if required. A typical situation is one where your procedure receives an object and you want to decide what to do with it:</p> <p>Example: In A3Edit, when you drop a variable onto a window, A3Edit decides which is the best default representation for this variable: String variables will be represented as text fields, boolean variables as check boxes etc. This is possible through a statement such as:</p> <pre>WITH variable : StrVal.Value DO ...   Toggle.Value: ... ... ELSE (* default action for unspecified or unknown class *) END;</pre> <p>You can also use a simple "IF variable IS StrVal.Value THEN" statement. You could then follow up with a type cast: "v := variable (StrVal.Value)". They are obviously checked at run-time, to ensure type safety.</p>
No local procedures (Lack of important feature)	<p>The absence of local procedures is quite significant. It leads to excessively long parameter lists, where local variables need to be shared or the use of global variables, where none should be used. C programmers also declare variables "just before use" inside the code, which is a very bad idea.</p> <p>With local procedures, the best way to declare a FOR loop variable, for example, is to place the loop and its variable into a local procedure, which is still able to access the entire environment. The compiler will in-line any local procedures that are short or only called once (even long ones), so there is no overhead, but readability is enormously enhanced.</p> <p>The basic idea is: one coherent process = one procedure and thanks to local procedures, this is easy to apply.</p>
Single character errors (Syntax general C problem)	<p>A single character added or removed may completely change the semantics of the code. The common problem is that all of them are almost invisible and could happen accidentally, moving blocks of code or just through a typo. To name a few:</p> <p>adding a colon after a conditional statement will cause the following statement or block NOT to be affected by the condition</p> <p>a conditional statement with "=" or "==" is equally valid</p> <p>any of the combined assignment statements "+=", "-=" etc. are perfectly interchangeable with a simple assignment; adding or removing the operation will totally change the meaning of the statement, but leave the code syntactically correct and no warning can be produced.</p>

Readability (Syntax, general C problem)	Yes, I know, many people will tell you that they like accolades. Well, if you manage to always properly write and read this kind of code:	
	Standard formatting:	Compact formatting
	<pre> void doSomething {   if (...)   {     while (...)     {       if (...)       {       }     }   } } </pre>	<pre> void doSomething {...   if (...) {     while (...) {       if (...) {       }     }   } } </pre>
	<p>Lining up all those accolades quickly becomes tedious. Adding comments at the end of each block, as is advisable for long sequences of code and as is usually done in Oberon-2 or Modula-2, may actually make the block end difficult to see, so it's usually omitted.</p> <pre> PROCEDURE doSomething (); BEGIN   IF ... THEN   WHILE ... DO   IF THEN   END;   END;   END; END doSomething; </pre> <p>Only one spelling here, since there is no symbol for the beginning of a block. This is just one of many samples. A major improvement in Oberon-2 is - as you may have noticed, that each IF, WHILE etc. automatically require an END statement. You cannot have an open block with no END. And you cannot possibly forget a block BEGIN statement. This is a significant improvement over Pascal, introduced with Modula-2.</p> <p>It might be worth doing an actual statistical test on this, but the Oberon-2 syntax seems much more in line with human style pattern recognition. Let's not forget that the main reason for choosing the terse characters in C was the lack of memory on computers in 1970, not the result of extensive research.</p>	

Parameter passing (Syntax, general C problem)	<p>Although C++ improves on the definition of procedural parameters, the very mechanism remains tainted. When you pass a parameter by value or by address, you actually have to add a symbol to conform to what is expected by the procedure.</p> <p>In Oberon-2, if you want to get a parameter by address, you specify it with PROCEDURE something (VAR parameterName: type);</p> <p>whereas if you want to pass it by value, you simply omit the VAR attribute. The decision to use either one naturally rests with the designer of the procedure. In C/C++, the caller must verify the interface every time he wishes to make the call, to ensure that he added the appropriate "something (*parametr)" or "something (&amp;parameter)".</p> <p>Although many of the related errors may be caught, it is sufficient for the parameter to be compatible with the derived class to cause major problems. This type of error is very, very hard to find and it seems that every C++ programmer has a story to tell about it.</p>
Virtual procedures (Not best OO practice)	<p>C++ classes must define right away which methods will be static and which will be virtual.</p> <p>Only virtual ones can be overridden by a derived class. This is a major limitation, since it requires the designer of the class to make a firm decision to allow inheritance, limiting severely the possible uses of the class..</p> <p>For "performance reasons", C++ programmers will prefer static methods, which will lead to a very static, non-object oriented design.</p> <p>In Oberon, a static procedure is just that - a procedure. In the same module as the class, maybe, but not a method of the class. If it is in the same module, it has full access to all fields and methods of the class, but it is clearly recognizable as a static procedure.</p>

The list could go on and on and on... When reflecting on these shortcomings of C++, one can't help but think of a famous line from "Jurassic Park, The Lost World".

- Hammond: "See, we're not making the old mistakes again."
- Malcom: "Yes indeed, you are making entirely new ones!!!"

Except that C++ manages to combine the old mistake from C with a whole new set of serious problems.

## **11.7 Using Amadeus Code with Terminal Services / Citrix**

Terminal Services and – in a more powerful version – Citrix, are a way to execute applications remotely. Basically, all display commands from an application are compressed and sent to another computer, which executes these commands to show the same image as you would see if you ran the application locally and transfers back all the user interactions. Both support a level of file sharing, allowing to open local files from your remote application.

Amadeus applications work perfectly well with both, Terminal Services and Citrix. They are even ideally suited for such an environment.

## 11.8 Table of Figures

Figure 1 - Amadeus-3 Context.....	20
Figure 2 - The Layering of OS, GUI, Amadeus-3 and Application .....	21
Figure 3 – Overall Amadeus-3 Program Structure .....	25
Figure 4 – Execution flow in an Amadeus-3 application.....	26
Figure 5 - Object encoding .....	32
Figure 6 - Data dictionary concept.....	43
Figure 7 - Basic window attributes .....	44
Figure 8 - Important window fields .....	47
Figure 9 - Adding a display object to a window .....	48
Figure 10 – Sample Bitmaps.....	52
Figure 11 – Tab Control.....	53
Figure 12 - Use of ValueDsp . Object in scroll lists .....	54
Figure 13 - New display object "Projection" .....	55
Figure 14 - Module/Object concept for KS Demo.....	58
Figure 15 - Application-defined display objects.....	59
Figure 16 - Guideline concept.....	60
Figure 17 - Guideline data structure .....	61
Figure 18 - Guideline record structure .....	62
Figure 19 - Guideline options .....	64
Figure 20 - Conceptual view of layers .....	65
Figure 21 - Window layout concept.....	68
Figure 22 - Message handling.....	70
Figure 23 – Sequential Source and Scrolling explained .....	77
Figure 24 – Scroll window update mechanism .....	78
Figure 25 - Drag and Drop object structure .....	85
Figure 26 – Database internal structure .....	91
Figure 27 – Sorting strings.....	92
Figure 28 – Sorting composite indexes (segmented keys).....	93
Figure 29 - Sample database design.....	94
Figure 30 - Index structure table.....	95
Figure 31 - Select database in DbViewer.....	104
Figure 32 -Viewing selected file.....	104
Figure 33 - Editing database contents and definition.....	105
Figure 34 - Interaction of application and report object .....	106
Figure 35 - Simple report viewer .....	109
Figure 36 - Sample report with graphical objects .....	110
Figure 37 - Sample of Report Table.....	116
Figure 38 - Complex form using object insertion to fill blank area .....	122
Figure 39 - The code generator .....	125
Figure 40 - The application editor.....	127
Figure 41 - Ordering objects .....	129
Figure 42 - Object note settings .....	131
Figure 43 - Creating new objects .....	134
Figure 44 - Creating a new dictionary entry .....	135
Figure 45 - Creating dictionary object from tempalte.....	135

Figure 46 - New decorative object.....	136
Figure 47 - Create Module from Application Class.....	137
Figure 48 - Define Object Class.....	138
Figure 49 - Layer and object lists.....	139
Figure 50 - Guideline editing .....	141
Figure 51 - Menu editing windows .....	142
Figure 52 - Database editing .....	143
Figure 53 - Window template selection .....	144
Figure 54 - New project .....	145
Figure 55 - Parsing error display .....	147
Figure 56 - "Hello World!" Demo program .....	153
Figure 57 - A3Edit with initial Tutorial data .....	154
Figure 58 - Sample window "Button Test" .....	156
Figure 59 - "Data entry Mask" .....	159
Figure 60 - Window Drag & Drop .....	160
Figure 61 - Module Overview Diagram.....	191
Figure 62 - Suggested directory structure with XDS .....	271
Figure 63 - Directory structure for DOS / Win32 and OS/2 .....	272