
Aglet Modula-2 PPC

Beta release (15.2.2012)
compiler v3.2 (13.2.2012)
Mar 4, 2012

Overview
Requirements
Installation

[Overview](#)
[Requirements](#)
[Installation](#)

Modula-2 Language

[M2_Language](#)

Command Line Tutorial
Aglet Implementation
Example Programs

[CLI_Tutorial](#)
[Aglet_Implementation](#)
[M2_Examples](#)

IDE Tutorial
M2IDE

[IDE_Tutorial](#)
[M2_IDE](#)

Release History

[Version_History](#)

Modula-2 Syntax

[ISO Modula-2 Syntax](#)

Tom Breeden
tmb@virginia.edu

AmigaGuide(R)

Next section: [Requirements](#)

Aglet Modula-2 PPC Overview

Introduction	Introduction
Features	Features
Limits	Limits
Known Bugs	Bugs
Future	Future

This is a Beta release of a native PPC Modula-2 compiler for Amiga OS4. *Aglet M2 PPC* v3.2 Beta (15.2.2012) implements much of the ISO Modula-2 base standard.

I make no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

This is copyrighted freeware being distributed "as-is". I hope it can be useful for anyone interested in developing new generation Amiga software with a Wirthian language.

Even though this is a beta release, I believe the package is in a usable condition. I have successfully built a number of non-trivial programs with it:

- > *Mod2* compiles itself.
- > *Mod2Lnk* , the pre-Linker used for building programs
- > The included *M2IDE* development environment
- > *IDLtm2* , an IDLTool analogue for producing Interface DEFINITION modules
- > A test generator program, *tgM2* , for Modula-2
- > A TestManager program for the above to handle the creation, compiling, linking, and analysis of results for lists of tgM2 test modules.

- > The *GuideMaker* program on OS4Depot
- > The *LoggerWindow* program on OS4Depot
- > The *Capture* Programmer's Challenge Game.

The compiler is, of course, not competitive with GCC for PPC code optimization, but it does a good job of creating correct machine code for a correct Modula-2 program.

Modula-2 is certainly a relatively "obscure" (at least in the U.S) language, but far from a dead one. A number of compilers are available without cost for different platforms. There is an ISO standard and most newer compilers, including *Aglet M2 PPC* , cleave closely enough to the standard to achieve good portability.

M2 does offer some things you don't get with C:

A better approach to building modular software - You don't have to spend 50% of

your development time figuring out why your "make" file does not work. :)

A cleaner, simpler language than C, offering a better type system, more rational array handling, much better design for modular programming supporting Abstract Data Types and much greater opportunity to change module implementations without propagating complexity and uncertainty.

Included Amiga-oriented support modules designed to get you effectively using Intuition, Reaction, etc, without having to become an expert in all the details - Direct calls to almost all Amiga Libraries are available, but intermediate modules from Aglet like "SimpleGUI", "SimpleRequesters", "SimpleImageHander", "SimpleRexx", and "AmigaTimer" expose a straightforward interface to common needs.

Section head: [Overview](#)
Next subsection: [Features](#)

Aglet Modula-2 PPC Introduction

Aglet M2 PPC is a complete rewrite of my previous M68K compiler. The compiler has been redesigned as a front-end/back-end compiler with intermediate code production in order to open the potential for increased optimization and portability. Code is generated compatible to the SYSV ABI for PPC.

It implements much of the *ISO 10514 Modula-2 Base Standard*, and (most of) the specified ISO standard libraries have been provided. The AOS4 SDK is represented by modules containing Amiga System Library definitions and interfaces.

PPC native code is written out as an assembler language file, which is processed by *as*, the GNU assembler delivered with the AOS4 SDK. A M2 pre-linker pulls together all the modules used by your program and implements inter-module version checking, then invokes *ld* to create the executable.

The compiler has bootstrapped itself, so most modules compile very quickly. It does not yet do much in the way of optimization, but I think it offers simpler way to get into producing sophisticated native Amiga OS4 programs than *gcc*.

It has roots in the *Benchmark Modula-2* compiler (which in turn has its roots in the ETH one-pass compiler for the M68000). Many thanks to Jim Olinger of Armadillo Computing, with whom I worked for a number of years on the AmigaOS library support for *Benchmark Modula-2*. Though Jim is no longer active in Amiga matters, he expresses his support for this new M2 project building on what he provided the Amiga community in *Benchmark Modula-2*.

Section head: [Overview](#)Next subsection: [Limits](#)

Aglet Modula-2 Features

The most important goal for *Aglet M2 PPC* has been to provide a modern Amiga M2 compiler consistent with the ISO Modula-2 Standard. There is still a good deal to be done before it can be called ISO conforming, but much has been implemented.

For an overview of the Modula-2 language, see the section [M2_Language](#). Notable language ISO additions which were not in Niklaus Wirth's *Programming in Modula-2* document include:

- > module termination (**FINALLY** section)
- > an exception handling facility (**EXCEPT** section)
- > the ISO standard library modules.

The compiler and pre-linker can be used from the CLI, but *Aglet M2 PPC* also comes with a basic IDE: *M2IDE* organizes compile order dependencies for you, presents a GUI for one button compiling and linking, integrates with a text editor, and has more features useful for larger projects.

Aglet M2 PPC comes with Definition modules for direct calls to Amiga Libraries in AOS4, with parameters just as described in the SDK's AutoDocs (though in many case with improved item typing).

The ISO standard library provides platform independent modules supporting IO, Strings and number/string conversions, Math, multiprocessing/threading, and reading the system clock.

I've also included, as part of *Aglet M2 PPC*, a fairly extensive set of Aglet support modules, both Amiga specific and of more generic interest. eg,

- A number of Amiga-specific modules supporting a higher level usage of things like the Timer Device, Font access, ARexx, Amiga CLI argument processing, separate Amiga process startup, etc.
- A mixed bag of generally useful programming support modules for things such as
 - > Dynamic Strings
 - > Binary Trees, Hash Tables
 - > QuickSort, Binary Search
 - > Regular Expressions
 - > Matrix Operations, Simultaneous Equations
 - > Huge Integers
- A module package, "SimpleGui", intended to simplify the use of Reaction without sacrificing capabilities.
- A module package, "SimpleGraphics", providing 2D drawing and graphing calls into Intuition windows and regions within windows.

Section head: [Overview](#)Next subsection: [Bugs](#)

Limitations

See also: [Bugs](#) [LowLevel](#) [ISO](#)

Compiler

In general, limits are imposed only by the amount of memory available and the 32 bit size of the computer word used by the compiler. Some other details:

size of procedure/Module	-> Current limiting factor is how many live vars a proc creates (1024).
length of string literals	-> One line of text
number of dimensions of open arrays	-> 1
number of enumeration literals	-> 256
number of coroutines	-> 0 (Coroutines NYI)
size of FOR loop	-> 32K bytes (16 bit offset branches)

ISO features not yet implemented:
64 bit INTEGER/CARDINAL
tagged NEW/DISPOSE/TSIZE
structured type constructors
literal string append operations
multidimensional open arrays
dynamic modules
type COMPLEX

Linker

No facilities yet to support Amiga Library creation.
Linking of modules into object file archives not yet implemented.

IDE

AOS File Notification not yet used for source change status.

Section head: [Overview](#)
Next subsection: [Future](#)

Known Problems

Open arrays of SYSTEM types

Parameters which are open arrays of SYSTEM types (e.g., ARRAY OF WORD, ARRAY OF LOC) are not yet correctly implemented, eg. HIGH() may not be correct on these.

Grim Reaper during compile

Illegal memory accesses with incorrect source code may still occur. In most cases these are benign and you can continue with the compile after the GR by simply choosing "Continue Program". If not, see the notes in [VerboseCompile](#).

Wrong source position reported for some errors

The error position reported for parameter syntax errors always points to the closing parenthesis, rather than to the parameter that caused the error.

Most Aglet support modules are not thread-safe

The Storage module is an exception, so threads (eg started from the ISO Processes module) can allocate and deallocate memory safely.

Plans

AmiUpdate

After Release 3.2, I plan to provide more frequent, smaller fixes and updates via AmiUpdate (www.amiupdate.net).

Fix Known Bugs and Awkwardnesses

[Bugs](#)

Compiler Robustness

The compiler does a good job of parsing correct M2 source code, but unfortunately there are currently still some circumstances in which a syntax error will evoke a Grim Reaper. Most of these are innocuous NIL dereferences so that the compiler can be continued from the GR, but all need to be fixed.

Code Generation

The compiler currently does only a little work attempting to improve the baseline correct (hopefully) intermediate and machine code. There is lots of work to be done improving this.

Modules as Amiga Libraries

To be done.

Modules as Static Libraries

This could decrease executable size from 0% to 20%, depending on the modules used.

ISO compatibility

- Structured Constants
- SYSTEM type parameters ISO-compliant
- Dynamic Modules
- Tagged Allocate/Deallocate
- ISO-compliant CAST
- Multi-Dimensional Open Arrays

Overflow and Range Checking

Currently, the compiler does implement run-time overflow and range checking, but not consistently, ie, there remain many places where such checking could occur but, at this time, does not get inserted.

Object Oriented ISO Extensions

The ISO group also specified a standard for adding classes with inheritance to the base standard. I am quite interested in implementing this.

Aglet M2 Requirements

Machine Resources

- > 26MB disk space is required for the entire contents (7MB less if the module source archive is not selected).
- > My development has been done on an 800MHz AmigaOne XE and a 666MHz Sam440ep with 512MB memory.

For Compiler and Linker

- > Amiga OS4.x up-to-date Installation. Development work is being done on AOS 4.1 update4 on a Sam440-flex and an AmigaOne.
- > Hyperion SDK Installed
 - SDK should be v53.8 or later. Development work is being done on SDK 53.20.
 - "as" and "ld" are called by the M2 compiler and linker, and must be accessible by the path "SDK:gcc/bin/" (as they will be after the usual SDK install).
 - Transcendental and sqrt functions are implemented via the C library SDK:clib2/lib/libm.a (or by the NewLib library SOBJS:libc.so).

The SDK is freely available at the Hyperion web site, <http://www.hyperion-entertainment.biz/>, in the "Downloads" section.

- > A directory for the Aglet M2 package (~20MB)
 - e.g. "Work:AgletV3". I suggest that an Assign for this directory be created as "AGV3:".
- > multi ASSIGN of "M2Lv3:" for supplied library modules
- > "T:" assigned to a nice location for temporary files.
- > "PIPE:" device accessible.

The install will create an assign script you can insert or call from s:user-startup:

```

;-----
; M2-Assigns.s   Assigns for Aglet Modula-2
;
assign agv3: ; <directory in which you installed Aglet M2 PPC>
;
assign m2lv3: agv3:system agv3:amiga agv3:iso agv3:reaction
agv3:sysmod agv3:experimental
;

```

```
path agv3: ADD  
LOADWB NEWPATH ; <remove if the script is run in user-startup>  
;-----
```

For IDE

> Arexx up and running

> An Arexx capable Editor

A "plugin" is currently provided for these editors

GoldEd (tested on v7.23)

TurboText (tested on v2.0)

CygnusEd (tested on v4.20)

MicroGoldEd (probably works, is untested)

Annotate (tested on v2.7.5 and v2.7.7)

Others may be made available by implementing the ARexx commands in M2IDEEdtCmds.def following the patterns of the sources of the above plugins. Send me a message and I may be able to do it fairly quickly.

Prev section: [Requirements](#)

Next section: [M2_Language](#)

Aglet M2 Installation

To Install:

Unpack the AgletM2PPC.lha file into a temporary directory. This unpacks to three files, AgletM2PPCBin.lha, AgletM2PPCModSrc.lha, and an AmigaDOS script, DoInstallAgletM2PPC.s.

EXECUTE the script and it will allow you to choose a parent directory in which it will create a directory named "AgletV3" and distribute the LHA contents into a directory structure underneath.

It will also create a file, "m2-Assigns.s", which will do the required ASSIGN, "M2Lv3:" (see [Requirements](#)). This assign is used by the compiler to find the standard support modules' symbol and object files. You may want to add a call to this script from S:user-startup.

Other things you will have to do are:

- > Put the directory where the compiler executables were placed into your shell path.
e.g., Use the PATH ADD shell command. This directory was called "AGV3:" in [Requirements](#) so it would be, "PATH AGV3: ADD". You may want to add this command into S:shell-startup.
- > If you don't have GoldEd as your editor, rename one of the other supported editor "front" programs to "EdtFront".
As of v0.3 (27.2.2010) you can instead explicitly give the name of your editor interface program. Use the CLI -EDITOR switch, eg. "-Editor PROGDIR:cedfront"
- > If you do have a the GoldEd (and the Cubic package), do visit OS4 Depot and download Frank Ruthe's materials for a very nice extended integration of M2 into GoldEd 8. Filed as "development/ide/agletm2cubic.lha" It includes a very detailed description of how to setup the various features of GoldEd for this purpose.
URL: http://os4depot.net/share/development/ide/agletm2cubic_lha.readme
- > Frank Ruthe has also designed some very nice icons for AgletM2 and given permission for me to include these with this distribution. You may want to set these as your system default icons for M2 source files and M2IDE project files.

----- Notes -----

The executables are **Mod2**, **Mod2Lnk**, **M2Err**, **IDLTm2** for the Compiler/Linker, and **M2IDE**, **WindowLogger**, **EdtFront** for the IDE.

The IDE communicates with your selected editor via an intermediate program, named "EdtFront" (unless the -EDITOR switch was used). One of the supplied executables (GedFront, AnnFront, CedFront, or TtxFront) should be copied as file "EdtFront". Sources for these programs are supplied in case you want to write one for a different editor.

The "Name List" file, "RegMods.nl", is used by *M2IDE* to exclude processing of the standard support modules from your project. This leaves only the modules you are working on for a specific project in the *M2IDE* modules window, and avoids touching the standard support modules.

A GoldEd syntax file for Modula-2 syntax coloring is also provided. It goes into GoldEd/add-ons/Modula2/Syntax/Dictionaries. A similar syntax file suitable for Annotate is also provided.

A few test and example program sources are unpacked into the "/Examples/" subdirectory.

.

Prev section: [Installation](#)
Next section: [CLI_Tutorial](#)

The Modula-2 Language

M2 Description	Modula-2
M2 Syntax	M2_Syntax
Module Consistency	ModuleKeys
Module Init and Term	InitTerm
Exceptions	Exceptions
ISO M2 Standard	ISO
ISO Standard Libs	ISOMods
ISO Examples	ISOExamples
Obj Oriented M2	OO_Extensions
M2 Links	M2_Links

Modula-2 Description

Modula-2 was designed and released in the '80s by Niklaus Wirth as the "serious" language for large application and system software to follow up on the "teaching" language, Pascal, that he had worked on in the '60s and '70s.

The major design idea is to foster reduction in complexity via "information hiding" while at the same time providing the programmer with all necessary tools for low level programming within the framework of a strongly typed language.

The public interface of a software module (which is the **DEFINITION MODULE**) is published separately from the code itself (which is the **IMPLEMENTATION MODULE**). If changes, or even major revisions of the code, are limited to the Implementation module, they need not propagate to other code that uses your module. Client code can see only what is in the Definition module.

Modula-2 thus provides an ideal platform for "Abstract Data Types": an ADT is a programmer-defined object type along with a collection of functions providing the complete set of allowable operations on the objects of that type. The ADT approach fosters programming with objects, though some important features of true Object Oriented Programming are missing: ADT objects themselves have no initialization and termination routines (though their Modules do), and there are no inheritance relationships.

In the 90's, an [ISO](#) was developed and published (ISO/IEC 10154-1) which defined the language in detail, and added a few things without materially changing Modula-2's basic design and elegance. Notably, this included a module *termination* routine, a well defined mechanism for *exception* trapping and recovery, and a [ISOMods](#) that all implementations are expected to provide.

In addition, two standard extensions defining how a language implementer should, if desired, provide an extended language in two areas: One defined language additions to provide true **Object Oriented Modula-2** (with *classes*, *inheritance*, and *garbage collection*) . The other was a **Generic Programming Extension**, to provide template-like modules that still maintain type checking safety.

On the web, check out [M2_Links](#).

Modula-2 Syntax

The syntax of Modula-2 will be fairly familiar to anyone who knows a bit about Pascal. Like C and Pascal, M2 is a block structured declarative programming language. Unlike them, it was designed from the beginning to support modular programming and information hiding.

The best resource for learning to use Modula-2 is an on-line textbook, "Modula-2: Abstractions for Data and Programming Structures - Modula-2 shareware textbook" by Rick Sutcliffe. This can be found at <http://www.csc.twu.ca/rsbook>.

An explicit specification of the static syntax of M2 is included here: [ISO Modula-2 Syntax](#).

The ISO/IEC 10514-1 document explicitly defining the ISO standard semantics is available, for purchase, from ANSI in the United States, and other ISO organizations elsewhere.

The GNU language backend is currently receiving a mostly ISO compliant Modula-2 front end. Though some differences exist based on its integration with the GNU languages, documentation for this compiler is freely available. (<http://www.nongnu.org/gm2/homepage.html>)

On the web, check out [M2_Links](#).

Module Consistency

Definition modules are the public interfaces of the units that are put together to form a M2 program, and Modula-2 will enforce that they be compiled in the order of dependency and modification:

All Definition modules that a Definition module **IMPORTS** (ie, that it depends upon) must be compiled before the Definition module itself. Thus, if a Definition module is changed and re-compiled, all compiled modules that depend on (ie, **IMPORT**) it need to be re-compiled.

Any change and re-compile of a Definition module results in a new unique key being stored in the symbol file (`#.SBM`) created for this Definition module compilation. Its symbol file also contains the keys of each module that it imports, directly or indirectly.

So, the compiler can enforce this consistency in Definition modules when they are combined into a program or Implementation module by using the keys to check that all versions for each used Definition module in the `#.SBM` files are the same.

For Example

Given that Definition module **A** imports Definition modules **B** and **C**, and that Definition module **B** imports **C**. In the last compile of the following sequence the compiler will detect an consistency error:

```
Compile C.def
Compile B.def
Modify C.def
Re-compile C.def
Compile A.def <- error emitted
```

The compiler will give a "keys of imported symbol files do not match" error. As the public interface of **C** changed but **B** was compiled with the old version, you will need to recompile **B** before the compile of **A** succeeds.

M2IDE

The included Integrated Development Environment for *Aglet M2 PPC* automatically determines the correct order of compilation of Definition modules within a project.

Implementation Modules

Because the implementation of the module is hidden, in general the order of compilation of Program and Implementation modules does not matter so long as all the Definition modules they use have been compiled appropriately.

If not, however, similar consistency errors to that described above can be revealed during compilation of Implementation modules.

It is also possible that some may not be detectable at compile time, but only when two fairly independent modules are combined at link time. *Mod2Lnk* will report any of these [ModKeysLink](#).

One more potential problem that may show up at link-time stems from [ModKeysCircle](#) between two implementation modules during module initialization.

Link-time Inconsistency

For Implementation modules, some import version inconsistencies are not detectable by the compiler, but will be discoverable at link-time.

This may occur when the attempt is made to link together two independent Implementation modules which were compiled with different versions of a third Definition module.

The *Aglet M2 PPC* pre-linker, *Mod2Lnk*, will detect these and will output information to help you determine which Implementation modules were compiled with the wrong version.

For example, the following output from a development link of M2IDE, makes it clear that the M2IDE#? and LoggerDefs Implementation modules have not been recompiled with the up-to-date version of AmigaDOS2.def.

Module Key Mismatches

AmigaDOS2

matching clients:

**AmigaDosInterfaces DebugIO SystemRTS
 Assertions Debugging OSFile StdChans
 LocaleInterfaces CharClass
 ProcessesSupportSimpleScreens SysClock
 VirtualTerminal MyTerminal
 DirUtil2SimpleRexx DirUtil3 PipeIO
 ProgramArgs ArgsSupport TextDisplays**

mismatching clients:

M2IDERun M2IDEutils M2IDEapi LoggerDefs

Prev section: [M2_Language](#)

Circular Import Inconsistency

Mod2Lnk will attempt to arrange the order of initialization of a program's modules such that when a module's init code is called, it can assume that the init code for all modules that it **IMPORTs** will already have been called.

However, if two Implementation modules each import from the other's Definition module, this is not possible, and there may be a problem.

The problem occurs when something in the initialization code of Module A requires that the initialization code of Module B already be run, or vice versa, or both.

Usually, this requires a direct call from init code in one of the modules to a initialization-sensitive procedure in the other module, however, indirect circularity can exist and can be hard to spot just by perusing the code.

The compiler cannot detect this situation, but the *Mod2Lnk* can at least find all Implementation modules that mutually import each other and emit a warning that the error is possible (but makes no attempt to analyze module semantics to determine if it occurs!).

For example, *Mod2Lnk* could output:

Warning MutualImport: IOLink IOChan

Warning MutualImport: SimpleGUIHidden SimpleGUI

Warning MutualImport: SimpleGUISupport SimpleGUI

Note that this is only a *warning*. The use of circular imports which *aren't* dependent on module initialization order are not a problem. Further, if the initialization dependency is only one-way between the two modules, the programmer can determine which one initializes first by importing it textually before the other one in the Program module.

If you get many of these warnings on a run of *Mod2Lnk*, it is probably a hint that the design of your partition of the program into modules may not be ideal.

NOTE: Circular imports among Definition modules *are* a language *error* and will be detected as such by the compiler.

Module Initializations and Terminations

As specified in the ISO standard for Modula-2, any module can have a code section to be executed on program termination, in addition to the one for initialization.

The module body code can be divided into two sections, the first containing the initialization code and the second the termination code. A new language symbol, **FINALLY**, was introduced to mark the beginning of the termination code.

Termination code will be called at program exit no matter what the reason for ending the program: end of main module body, a **HALT**, or a trapped exception.

Modules' termination codes will be called in the opposite order from which their initialization codes were called.

A module's termination code is called only once. A **HALT** or exception within its termination code will end that one and continue with the next module's termination code.

If program exit occurs during initialization before the main module is reached, modules whose initialization code has not been entered will not be terminated. This implementation will always call termination code, if any, for modules that have no initialization code at all.

The standard system module **TERMINATION** is supported as a "built-in" standard module. It has two procedures:

PROCEDURE IsTerminating():BOOLEAN;
PROCEDURE HasHalted():BOOLEAN;

Exceptions Support

The general exception model allows an **EXCEPT** section at the end of module init sections, module term sections, and procedure bodies.

Upon an exception, control transfers to the start of the last activated **EXCEPT** block. An exception block becomes activated when its associated code block is entered, and deactivated on the code block exit.

A **RETRY** statement within the exception block will clear the exception status and restart the procedure (or module init/term code) from the beginning.

A **RETURN** statement within the exception block will clear the exception status and exit from the procedure or module body.

If neither statement is executed within the exception block (ie. the processing "runs off the end"), the exception state remains raised and the previously active exception block is entered. If there is none, the program ends via the runtime system default exception handler.

Some exceptions are pre-defined since they are specified in one of the Standard ISO Modules. User written modules can also define their own exceptions via the standard module **EXCEPTIONS**. User defined exceptions are sent to the exception handler cascade by a call to the `EXCEPTIONS.Raise()` procedure.

In addition, there are a set of "language exceptions" raised by the operating system or by run-time checks inserted by the compiler. Standard module **M2EXCEPTION** defines an enumeration of these so that they can be handled, if desired, in your module's **EXCEPT** section:

**(indexException, rangeException, caseSelectException,
invalidLocation, functionException, wholeValueException,
wholeDivException, realValueException, realDivException,
complexValueException, complexDivException, protException,
sysException, coException, exException)**

The compiler switches "-rngchk" and "-ovflchk" may determine, in many instances, where or whether most of the language exception situations will be detected at run time.

ISO Modula-2 Standard

The best resource for learning to use ISO Modula-2 is an on-line textbook, "Modula-2: Abstractions for Data and Programming Structures - Modula-2 shareware textbook" by Rick Sutcliffe. This can be found at <http://www.csc.twu.ca/rsbook>.

ISO Modula-2 Syntax

Implemented Features in *Aglet M2 PPC*

- [InitTerm](#)
- [Exceptions](#)
- **FOR** loop semantics
- **LENGTH** function
- **CAST()** and **VAL()** differences
- **SYSTEM.ADDADR()**.
- Storage module semantics
- [ISOMods](#)
- the **FINALLY** block
- the **EXCEPT** block
- index variable must be local & unthreatened
- for null delimited strings
- Type transfer "functions" removed
- ALLOCATE failure returns NIL, not an exception

Not (yet) Implemented in *Aglet M2 PPC*

- Structured type constructors
- Multidimensional open arrays
- Dynamic modules
- **COROUTINES** module
- **COMPLEX** Type

Section head: [M2_Language](#)
 Section head: [Aglet_Implementation](#)

ISO Standard Libraries

The *Aglet M2 PPC* implementation of these ISO standard modules was based on a set of sources provided (and programmed) by Richard Sutcliffe, Trinity Western University, BC Canada (Portions coded by G. Tischer). I am extremely grateful for his contribution.

Thanks to the WG13 and ISO/CS in Geneva, for making the text of all the definition modules in the standard available.

Built-In Language Modules

SYSTEM
 TERMINATION
 EXCEPTIONS
 M2EXCEPTION

StdIO Library - Logical Device (Channel) Access

SeqFile	Rewindable sequential files
StreamFile	Independent sequential data streams
RndFile	Random access files
TermFile	The console device
ProgramArgs	Program arguments as an IO channel

StdIO Library - Channel Input/Output

TextIO	Character and line IO
WholeIO	Integer and cardinal IO
RealIO	Real number IO
LongIO	Long real number IO
RawIO	Byte buffer IO

StdIO Library - Errors and Result Codes

ChanConsts	Open modes and open results on channels
IOConsts	Read operation result codes
IOResult	ReadResult() procedure

StdIO Library - Standard Channel Input/Output

STextIO	As above, but the channel is implied.
SWholeIO	"
SRealIO	"
SLongIO	"
SRawIO	"
SIOResult	"
StdChans	Set and get the actual channel used for Standard IO

StdIO Library - Device/Channel Association

IOChan	Lower level channel operations, channel exceptions
IOLink	Channel architecture for device association

Strings Library

Strings	Comparison, extraction, insertion, deletion
CharClass	Numeric, letter, control, whitespace

High Level Conversions

WholeStr	Integer/Cardinal to and from string
RealStr	" for Real, with float, fixed, engineering formatting
ConvTypes	Conversion error codes

Low Level Conversions

WholeConv	Useful for constructing conversion routines on Int/Card
RealConv	" for Reals
LongConv	" for Long Reals

Math Library

RealMath	Trig, log, square root, rounding functions
LongMath	" for LONGREAL
LowReal	Access to underlying properties of the REAL type
LowLong	" for LONGREAL

NOTE: Aglet's RealMath and LongMath use C libraries to implement their procedures. You must use either "-libname SDK:clib2/lib/libm.a", to use Clib2 or "-libname SOBJS:libc.so" to use C Newlib for Mod2Lnk to find the implementation.

Multiprogramming/Threading

Processes	Implemented via AmigaDOS Processes
Semaphores	Provides mutual exclusion facilities for processes
Coroutines	<not yet implemented>

Etc

SysClock	Access to current date and time
----------	---------------------------------

ISO Examples

- [ISO_Exam1](#) Open an existing text file for reading
Read a text file
- [ISO_Exam2](#) Create a new file "safely"
Create a new file "destructively"
- [ISO_Exam3](#) Open an existing binary file to read it
Read a binary sequential file
- [ISO_Exam4](#) Open an existing random-access file for binary reading
Open or Create a random-access file for binary read/write

ISO Examples-1

Open an existing text file to read it

```

FROM IOChan IMPORT ChanId;
FROM ChanConsts IMPORT ChanFlags, FlagSet, OpenResults;
FROM SeqFile IMPORT OpenRead;

VAR f :ChanId;
oRes :OpenResults;
...
OpenRead(f, "MyDev:MyPath/MyFile.ext", FlagSet{ }, oRes);
IF oRes <> opened THEN
  <error processing>
...

```

Read a text file

See Aglet support module [TextIOHelper](#) as well.

```

FROM IOConsts IMPORT ReadResults;
FROM IOResult IMPORT ReadResult;
FROM TextIO IMPORT ReadString, SkipLine;

VAR OneLine :Str0.String132;
rRes :ReadResults;
...
ReadString(f, OneLine);
rRes := ReadResult(f);
WHILE rRes # endOfInput DO
  IF rRes = endOfLine THEN
    SkipLine(f);
  ELSE
    Assert(rRes = allRight, "Error encountered reading text file");
    <process the OneLine>
    ReadString(f, OneLine);
  END;
  rRes := ReadResult(f);
END;

```

ISO Examples-2

Create a text file, erroring if there is an existing file of that name

```
FROM IOChan IMPORT ChanId;  
FROM ChanConsts IMPORT ChanFlags, FlagSet, OpenResults;  
FROM SeqFile IMPORT OpenRead;  
  
VAR f :ChanId;  
oRes :OpenResults;  
...  
OpenWrite(f, "MyDev:MyPath/MyFile.ext", FlagSet{}, oRes);  
IF oRes <> opened THEN  
  <error processing>  
...  

```

Create a text file, deleting any existing file of that name

```
FROM IOChan IMPORT ChanId;  
FROM ChanConsts IMPORT ChanFlags, FlagSet, OpenResults;  
FROM SeqFile IMPORT OpenRead;  
  
VAR f :ChanId;  
oRes :OpenResults;  
...  
OpenWrite(f, "MyDev:MyPath/MyFile.ext", FlagSet{oldFlag}, oRes);  
IF oRes <> opened THEN  
  <error processing>  
...  

```

ISO Examples-3

Open an existing binary sequential file to read it

```
FROM IOChan IMPORT ChanId;
FROM ChanConsts IMPORT ChanFlags, FlagSet, OpenResults;
FROM SeqFile IMPORT OpenRead;

VAR f :ChanId;
oRes :OpenResults;
...
OpenRead(f, "MyDev:MyPath/MyFile.ext", FlagSet{rawFlag}, oRes);
IF oRes <> opened THEN
  <error processing>
...

```

Read a binary sequential file

```
FROM IOConsts IMPORT ReadResults;
FROM IOResult IMPORT ReadResult;
FROM RawIO IMPORT Read;

TYPE MyRecords = RECORD Code, Month, Day, Year:CARDINAL END;
VAR OneRec :MyRecords;
rRes :ReadResults;
...
Read(f, OneRec);
rRes := ReadResult(f);
WHILE rRes # endOfInput DO
  IF rRes # allRight THEN
    <process the error>
  ELSE
    <process the OneRec>
    Read(f, OneRec);
  END;
  rRes := ReadResult(f);
END;

```

ISO Examples-4

Open an existing random-access file for binary reading

```
FROM IOChan IMPORT ChanId;
FROM ChanConsts IMPORT ChanFlags, FlagSet, OpenResults;
FROM RndFile IMPORT OpenOld;

VAR f :ChanId;
oRes :OpenResults;
...
OpenOld(f, "MyDev:MyPath/MyFile.ext", FlagSet{}, oRes);
IF oRes <> opened THEN
  <error processing>
...

```

Open or Create a random-access file for binary read/write

```
FROM IOChan IMPORT ChanId;
FROM ChanConsts IMPORT ChanFlags, FlagSet, OpenResults;
FROM RndFile IMPORT OpenOld;

VAR f :ChanId;
oRes :OpenResults;
...
OpenOld(f, "MyDev:MyPath/MyFile.ext", FlagSet{writeFlag+oldFlag},
oRes);
IF oRes <> opened THEN
  <error processing>
...

```

Object-Oriented Extensions

ISO also standardized an set of Object Oriented extensions for the base Modula-2 standard.

Hopefully, I will be able to implement these into the compiler classes in the foreseeable future.

In this distribution of *Aglet M2 PPC* , most of the class implementation mechanics are available via use of an accessory standard M2 module, [Module_Obj](#). It does implement true object-oriented classes (similar to BOOPSI). Unfortunately, the programmer-required bookkeeping is a bit onerous, and I have used it only in one module, [SimpleGraphics](#).

Some Web Links for Modula-2 Info

<http://www.csc.twu.ca/rsbook/index.html>

Modula-2: Abstractions for Data and Programming Structures - Modula-2 shareware textbook by Rick Sutcliffe

<http://www.modula2.org/>

A good source for info and source code. Modula-2 Org

<http://www.arjay.bc.ca/Modula-2/m2faq.html>

Modula-2 FAQ maintained by Rick Sutcliffe

<http://freepages.modula2.org/>

Modula-2 News Site

<http://www.nongnu.org/gm2/homepage.html>

GNU Modula-2 Home Page

<http://www.modula2.org/adwm2/>

A very nice freeware Windows M2 compiler system.

<http://sc22wg13.twi.tudelft.nl/>

ISO/IEC JTC1/ SC22 / WG13 Modula-2

* ISO/IEC 10514-1:1996 - Modula-2 (Base Language) (published on 1996-06-01)

* ISO/IEC 10514-2:1998 - Modula-2 (OO extension) (published on 1998-12-19)

* ISO/IEC 10514-3:1998 - Modula-2 (Generic extension) (published on 1998-12-19)

The international standardization working group for the programming language Modula-2.

<http://www.ohloh.net/p/m2r10>

Home site for a "next generation" Modula-2 language project.

Also, for discussions with other Modula-2 users you can subscribe to the Google group, *comp.lang.modula2* .

[Use M2IDE instead of the CLI for "real" projects.]

Command Line Tutorial

The general procedure for building a program via the CLI interface is to iterate through the following process:

- > Design your program, using existing support Modules and/or constructing new Definition Modules for the major program components.
- > Write the main program Module.
- > Write the Implementation modules for the new Definition modules you have designed and written.
- > Compile these Definition modules.
- > Compile your main Program module and all your new Implementation modules.
- > Link the program, eg, "mod2lnk MyProgram"

Note: Definition modules must be compiled in the logical order: all Definition modules that a Definition module **IMPORTs** (ie, that it depends upon) must be compiled before the Definition module itself. (Using *M2IDE* instead of the CLI to build your package will insure that this requirement is met.)

Any change and re-compile of a Definition module results in a new unique key being assigned to the symbol file (`#.SBM`) created during compilation.

By reading these files, the compiler will check the Definition module dependencies, direct and indirect, of your program and all its **IMPORTed** modules. It will declare an error if it finds the dependency tree contains more than one version of the same Definition module (ie, Definition modules were not compiled in the correct order).

In general, the order of compilation of Program and Implementation modules does not matter, but see [ModuleKeys](#) for details on exceptions to this.

[Use M2IDE instead of the CLI for "real" projects.]

A Simple CLI-Built Example

Write your module source

Let's start with the time-honored example, but leave out a parenthesis in order to make the compile more interesting. [HelloWorld_err](#)

Compile the module

Open a Shell window and invoke the compiler on your source program. You should get compilation messages that look similar to the below.

```
> mod2 HelloWorld.mod
Aglet PPC Mod2 v3.1 Compiler Beta1 (13.2.2011)
Copyright (c) 2004 by Thomas Breeden

HelloWorld.mod
<- M2Lv3:STextIO.SBM

  >>Errors in Source File!
```

Run the M2Err program to see the error messages.

```
> m2err
M2Err Copyright © 2004 Tom Breeden
LIST OF ERRORS FOUND IN FILE: HelloWorld.mod
WriteString("Hello world!";    (* missing parenthesis *)
                             ^
line: 10  err: 015 ) right parenthesis expected
```

[Simple_CLI2](#) (Fix and re-Compile the module)

[Use M2IDE instead of the CLI for "real" projects.]

A Simple CLI-Built Example, continued

Fix and re-Compile the module

Edit the parenthesis into the file at the appropriate point and try again. [HelloWorld](#)

```
> mod2 HelloWorld
Aglet PPC Mod2 v3.1 Compiler Beta1 (13.2.2011)
Copyright (c) 2004 by Thomas Breeden
HelloWorld.mod
<- M2Lv3:STextIO.SBM

-> T:HelloWorld.asm

Optimize Setting: DeadCode
SDK:gcc/bin/as -o HelloWorld.o T:HelloWorld.asm 0
```

This time the compiler successfully creates the PPC assembly language file and invokes the SDK program *as* to produce the Elf object file.

[Simple_CLI3](#) (Link the Program Module)

[Use M2IDE instead of the CLI for "real" projects.]

A Simple CLI-Built Example, continued

Link the Program Module

```
> mod2lnk HelloWorld
Mod2Lnk Amiga 0.4 (16.12.2009) OS 4.0 Beta2
Copyright (c) 2004 Tom Breeden

HelloWorld
SDK:gcc/bin/as -o HelloWorld_start.o HelloWorld_start.asm
SDK:gcc/bin/ld -o HelloWorld T:HelloWorld.lnk -q -nostdlib -x
Mod2Lnk done ErrStatus: 0000
```

The M2 pre-linker has inspected the HelloWorld.o file, which contains a summary of the program Module's imports, found the standard IO module STextIO and continued recursively to determine all the modules that must be linked for this program.

It created and assembled the startup object module for this program.

And finally it put together the a command file for the SDK linker and invoked *ld* to produce the executable, HelloWorld.

Run the Program

```
> HelloWorld
Hello world!
Goodbye world
```

```
(*#####*)
MODULE HelloWorld;          (* $VER: HelloWorld.mod 0.0 (13.3.2008) *)
(*#####*)

FROM STextIO IMPORT WriteLn, WriteString;

BEGIN

WriteString("Hello world!");  (* missing parenthesis *)
WriteLn;

FINALLY

WriteString("Goodbye world"); WriteLn;

END HelloWorld.
```

```
(*#####*)
MODULE HelloWorld;          (* $VER: HelloWorld.mod 0.0 (13.3.2008) *)
(*#####*)

FROM STextIO IMPORT WriteLn, WriteString;

BEGIN

WriteString("Hello world!");      (* missing parenthesis fixed here *)
WriteLn;

FINALLY

WriteString("Goodbye world"); WriteLn;

END HelloWorld.
```

Prev section: [CLI_Tutorial](#)
Next section: [M2_Examples](#)

Aglet Implementation

M2 Compiler
Obj File Linking
Tools

[Compiler](#)
[Linker](#)
[Tools](#)

Aglet Support Modules

[Support](#)

Amiga Specific Info
Tips
Debugging Hints

[Amiga_Specific](#)
[Tips](#)
[Debugging](#)

.

M2 Compiler

The compiler version documented here is:
Aglet M2 PPC Compiler v3.2 Beta Compiler (13.2.2012)
Copyright (c) 2004 by Tom Breeden

Compiler-Switches

All switches start with a dash.

Compiler-Pragmas

All Pragmas (embedded switches) are delimited by the "<*" and "*>" characters, similar to a "special" comment.

ALIB

LibAutoOpen

After the startup code executes, a number of libraries will already be open: exec, dos, graphics, intuition, layers, utility.

For nearly all other Amiga libraries, modules that provide access to the procedures will automatically open that shared library, and its main Interface, in the module initialization code (and close it in the termination code), so you will rarely if ever need to call OpenLibrary in your code.

CLIB

CNewLib

Other compiler info

1. A null char is placed at end of every literal string.
2. An embedded PPC assembler is provided, but it is very "bare-bones", and probably of little use except for very specialized requirements (eg, non-ABI calls).
3. On exit, the compiler provides a CLI ReturnCode of 20 for errors that prevent compilation, such as invalid arguments or inability to open a file. Runs that complete with compilation errors return 5, otherwise a 0 is returned for a successful compile.

SYSTEM.def

LowLevel

Compiler Switches

Runtime Checking Control

- rngchk** Range Chk = checks of indexing, subrange, implied type conversions. <- DEFAULT
- rngchkoff** Compile without generating any range checks.
- ovflchk** Overflow Chk = checks of arithmetic operations.
- ovflchkoff** Compile without generating any overflow checks. <- DEFAULT

Code Generation Control

- nocode** Parse and produce intermediate code, but skip generating PPC code.
- optlev <num>** Optimize Level (currently not needed as there is only one level)
- debugopt** Does a compile with very little optimization.

Debugging and Information

- list** Output a source + code listing
- listfil <str>** Write the listing to this file
- verbose** Verbose compile, reporting on each phase of the compile.
- verbose+** More Verbose compile

Import and Output Control

- infil <str>** Input file specification
- outdir <str>** Out Files Dir (for the sym and obj files) <- DEFAULT is current dir
- symdir <str>** Sym Files Dir - "M2Lv3:" will always be searched as well.

Settings of compiler resource sizes

- instrubuf <num>** Instruction Buffer size. <- DEFAULT is 5000

The Amiga program template looks like this:

```
-infil,-symdir/K,-outdir/K,-nocode/S,-optlev/K/N,-rngchk/S,-rngchkoff/S,
-ovflchk/S,-ovflchkoff/S,-stkchk/S,-instrubuf/K/N,
-list/S,-listfil/K,-pragma/K,-verbose/S,-verbose+/S,
-debugopt/S,-debug/S,-wdebug/S,-dbgverbose/S,
-dbgverbose+/S,-todo/S,-clrkey/S,-version/S,-help/S
```

Compiler Pragmas

<code><*ISOMode(param)*></code>	param = Strict, Semi, or Loose - eg " <code><*ISOMode(Semi)*></code> "
<code><*AlignMode(param)*></code>	param = SysV, Amiga, or Default
<code><*NoParamCopy(param)*></code>	param = ON or OFF
<code><*ForeignProc(param)*></code>	param = ON or OFF
<code><*Asm(param)*></code>	param = ON or OFF
<code><*PrologOff(param)*></code>	param = ON or OFF
<code><*PostlogOff(param)*></code>	param = ON or OFF
<code><*RangeCheck(param)*></code>	param = ON or OFF
<code><*OverflowCheck(param)*></code>	param = ON or OFF

NOTES:

1. ISOMode

Strict - Any non-ISO compliant usage is flagged as an error.

Semi - Allows ADR() on string literals

Loose - Allows ADR() on a lev 0 procedures.

- HIGH() on non-open arrays
- VAL() on pointers, sets, opaques
- CAST() on a numeric literal.
- SIZE() on a non-entire variable, eg, Ar[b], Pt[^], R.c
- Interprets quoted n, b, r, t, f, v, xNN NNN

2. AlignMode

Needed for most existing AmigaLib structure definitions, which are mapped to memory differently from the PPC SysV alignment rules. New code, if it does not require M68K compatibility, should use the default of SysV alignment.

3. NoParamCopy

Forces off any copying of non-**VAR** dynamic arrays. The compiler only makes a copy of these pass-by-value parameters if the procedure actually changes or "threatens" the parameter. In programming for a C based API like Amiga's this usually is due to having to use the ADR() function on a non-VAR parameter in order to pass a string pointer. Documentation should be consulted to determine if the called routine actually changes the string.

Careful, do not use this unless you are sure the array is not changed.

4. Asm

Bracket embedded PPC assembly code with `<*Asm(ON)*>` / `<*Asm(OFF)*>`. Use only in simple situations; may give problems if used within conditional statements.

5. ForeignProc

Procedures declared in the Definition module while **<*ForeignProc(ON)*>** are not present in the Implementation module. e.g, the procs in libm.def, which are linked in from the SDK's C library "clib2/lib/libm.a".

6. PrologOff/PostlogOff

Special usage, perhaps with **<*Asm(ON)*>**.

7. RangeCheck

For turning code generation of range checking ON or OFF between statements.

8. OverflowCheck

For turning code generation of overflow checking ON or OFF between statements.

Amiga Library Calls

Amiga library **Interfaces** are represented in *Aglet M2 PPC* as pointers to a record each field of which represents, and is named for, one of the functions defined in the interface's XML specification.

The first parameter, always the interface pointer itself, is not hidden. Varargs versions of the functions are not supported (and are represented in the Interface table structure simply by dummy **PROC** entries).

These records are defined in Definition modules distributed with *Aglet M2 PPC*, generally named by appending "Interfaces" to the name of the Amiga library. The Interface pointer, which is auto-opened in the module initialization code, is also declared in this Definition module, ready-to-use.

For example, to call the AmigaDos library Open routine, use:

```
f := IDOS^.Open(IDOS, "MyFile", ModeOldFile);
```

From the AmigaDosInterfaces.def file:

```
(*#####*)
DEFINITION MODULE AmigaDosInterfaces;
(*#####*)
FROM SYSTEM           IMPORT ADDRESS, BYTE;
FROM Types            IMPORT INTEGER64, STRPTR;
FROM Interfaces       IMPORT InterfaceData, InterfacePtr;
FROM AmigaDOS         IMPORT BPTR, BSTR, DateStampPtr, DateStampRecord,
                        FileHandle, FileInfoBlockPtr, <etc> ...

<*AlignMode (SysV) *>

TYPE DOSIFacePtr = POINTER TO DOSIFace;
CONST dosLibName = "dos.library";
VAR   DOSBase    :LibraryPtr;
        IDOS      :DOSIFacePtr;

TYPE
DOSIFace = RECORD    (* v1.0 *)
  Data :InterfaceData;
  Obtain      : PROCEDURE (DOSIFacePtr) : CARDINAL;
  Release     : PROCEDURE (DOSIFacePtr) : CARDINAL;
  Expunge     : PROCEDURE (DOSIFacePtr);
  Clone       : PROCEDURE (DOSIFacePtr) : InterfacePtr;
  Open        : PROCEDURE (DOSIFacePtr, (*name*) ARRAY OF CHAR,
                          (*accessMode*) INTEGER) : FileHandle;
  Close       : PROCEDURE (DOSIFacePtr, FileHandle) : INTEGER;
  Read        : PROCEDURE (DOSIFacePtr, FileHandle, (*buffer*) ADDRESS,
                          (*length*) INTEGER) : INTEGER;

  <etc> ...
```

See the description of the tool [IDLm2](#) for more details on *Aglet M2 PPC* 's support for creating interface Definition module files from the AmigaLibrary's XML Interface specification.

Library Auto-Open

The initialization code of a library interface Implementation module opens the library and its Main Interface, which is used to make calls to the library. The module's termination code closes both as well.

For example, for DataTypes we have

DEFINITION MODULE DataTypesInterfaces;

which exports these items:

VAR DataTypesBase :LibraryPtr;

TYPE DataTypesIFacePtr = **POINTER TO** DataTypesIFace;

VAR IDataTypes :DataTypesIFacePtr;

"DataTypesIFace" is a **RECORD** defining the interfaces's procedures, each as a named field in the record.

"IDataTypes" is the interface handle pointer which is used in calling the library's procedures and functions.

"DataTypesBase" is the library base pointer, not normally needed except within the module's init and term code.

eg,

NewDTObjectA :**PROCEDURE**(DataTypesIFacePtr, (*name*)**ARRAY OF CHAR**,
ARRAY OF TagItem):Object;

which is called as:

obj := IDataTypes^.NewDTObjectA(IDataTypes, 'Myfile.guide', TagItems);

Static C Library Calls

The ELF object file format used by the compiler is compatible with that of gcc. You can use C procedures from a static library (that follows the SYSV ABI) as follows:

- > Create a Definition module using the **<*ForeignProc(ON)*>** compiler pragma.

This defines the parameters and return value for M2 use and tells the compiler not to look for any implementation of these procedures in the corresponding Implementation module.

- > Use the **-libname <path/file>** switch when using *Mod2Lnk* so that the linker can retrieve the object code for these C functions.

(It should also be possible to prepare a static library from a Modula-2 module that can be linked into a C program, but I have done no work with this.)

Actually, the only static C library I've used so far is SDK:clib2/lib/libm, which is currently used to implement most of the ISO Standard RealMath library modules.

eg, _____

```
(*#####*)
DEFINITION MODULE libm;           (* $VER: libm.def 0.0 (3.7.2007) *)
(*#####*)
(* For calling functions in SDK:clib2/lib/libm.a
   You will need to use the "-libname" switch in Mod2Lnk as well. *)
```

```
<*ForeignProc(ON)*>
```

```
PROCEDURE acos(x:LONGREAL):LONGREAL;
PROCEDURE asin(x:LONGREAL):LONGREAL;
PROCEDURE atan(x:LONGREAL):LONGREAL;
PROCEDURE atan2(y, x:LONGREAL):LONGREAL;
PROCEDURE ceil(x:LONGREAL):LONGREAL;
PROCEDURE cos(x:LONGREAL):LONGREAL;
<etc> ...
```

Dynamic ".so" C Library Calls

Unix style ".so" C libraries can be linked similarly to static C archive libraries.

Mod2Lnk is currently configured only for the C Newlib .so library. ie, if *Mod2Lnk* is started with the CLI switch "-libname SOBJS:libc.so", it will search Newlib for unresolved symbols.

This has only be used and tested for the procedures in libm.DEF, which implement the ISO RealMath and LongMath modules. In fact, those modules are only implemented via the `<*ForeignProc()*>` pragma.

ie, either "-libname SDK:clib2/lib/libm.a", to use the static CLib2, or "-libname SOBJS:libc.so", to use C Newlib, must be used if any of RealMath or LongMath procedures are called in your program.

(I am not aware of any advantage or disadvantage of one or the other).

What Mod2Lnk actually does to use this library is:

- link system module CNewLib to open the newlib.library Amiga library at startup.
- add libc.so to the *.lnk control file for the *ld* linker.
- invoke *ld* with a CLI parameter "--defsym INewlib=CNewLib\$_data"

Section head: [Compiler](#)

The pseudo-module SYSTEM is embedded in the compiler, but still must be IMPORTED if you use its low-level facilities:

```
(*#####*)
DEFINITION MODULE SYSTEM;
(*#####*)

(* Gives access to system programming facilities that are probably non
portable. *)
(* The constants and types define underlying properties of storage *)

CONST

    BITSPERLOC      = 8;
    LOCSPERWORD     = 4;

    LOCSPERBYTE     = 1;
    LOCSPERHALFWORD = 2;

TYPE LOC; (* A system basic type. Values are the uninterpreted contents of the
smallest addressable unit of storage *)
    ADDRESS = POINTER TO LOC;
    WORD    = ARRAY [0 .. LOCSPERWORD-1] OF LOC;

(* BYTE and LOCSPERBYTE are provided if appropriate for machine *)
TYPE BYTE      = ARRAY [0 .. LOCSPERBYTE-1] OF LOC;
TYPE HALFWORD  = ARRAY [0 .. LOCSPERWORD-1] OF LOC;
TYPE WORDWORD  = ARRAY [0 .. LOCSPERWORD-1] OF LOC;

PROCEDURE ADDADR(addr:ADDRESS; offset:CARDINAL):ADDRESS;
(* Returns address given by (addr + offset), or may raise an exception if
this address is not valid. *)

PROCEDURE SUBADR(addr:ADDRESS; offset:CARDINAL):ADDRESS;
(* Returns address given by (addr - offset), or may raise an exception if
this address is not valid. *)

PROCEDURE DIFADR(addr1, addr2:ADDRESS):INTEGER;
(* Returns the difference between addresses (addr1 - addr2), or may raise
an exception if the arguments are invalid or if the address space is
non-contiguous. *)

PROCEDURE ADR (VAR v: <anytype>): ADDRESS;
(* Returns the address of variable v. *)

PROCEDURE ROTATE (val: <a packedset type>; num: INTEGER): <type of first
parameter>;
(* Returns a bit sequence obtained from val by rotating up or down (left or
right) by the absolute value of num. The direction is down if the sign
of num is negative, otherwise the direction is up. *)
```



```
PROCEDURE SHIFT (val: <a packedset type>; num: INTEGER): <type of first
    parameter>;
    (* Returns a bit sequence obtained from val by shifting up or down (left or
    right) by the absolute value of num, introducing zeros as necessary.
    The direction is down if the sign of num is negative, otherwise the
    direction is up. *)

PROCEDURE CAST(<targettype>; val:<anytype>):<targettype>;
    (* CAST is a type transfer function. Given the expression denoted by val,
    it returns a value of the type <targettype>. An invalid value for the
    target value or a physical address alignment problem may raise an
    exception. *)

PROCEDURE TSIZE(<type>; . . . ):CARDINAL;
    (* Returns the number of LOCS used to store a value of the specified
    <type>. The extra parameters, if present, are used to distinguish
    variants in a variant record. *)

(* ----- *)

PROCEDURE REG(r:[0..31]):CARDINAL;
PROCEDURE FREG(fr:[0..31]):LONGREAL;
PROCEDURE SREG([0..4]):CARDINAL;      (* 0=CR, 1=LR, 2=CTR, 3=XER, 4=FPSCR *)
(* FPSCR NYI *)

PROCEDURE SETREG(r:[0..31]; val:CARDINAL);
PROCEDURE SETFREG(r:[0..31]; val:CARDINAL);
PROCEDURE SETSREG(r:[0..4]; val:CARDINAL);
(* FPSCR NYI *)

END SYSTEM.
```

.

Low Level Information

Module SystemRTS

The Aglet system uses a "hidden" module, **SystemRTS**, to provide some common run-time code support for every program.

The compiler embeds calls to some of the SystemRTS routines in generated code in order to implement actions related to procedure prologue and epilogue, most of the EXCEPTION processing support, and other things. *Mod2Lnk* uses it for program startup and exit code.

SystemRTS is a normal module (residing in M2Lv3:system/), and must be accessible during every compile and link. You do not have to explicitly import it for the compiler's usage.

Currently, however, there are item(s) in its Definition module that you may want to access in your code (eg, a variable containing the CLIReturnCode). In which case, you must explicitly import SystemRTS.

Module CNewLib

Another "hidden" module, **CNewLib**, will be included automatically by *Mod2Lnk* if it is given the switch "-libname SOBJS:libc.so".

Module SymbolsRTS

The module **SymbolsRTS** can be used to include code to display the source name of procedures when an EXCEPTION occurs at runtime (or an Assert() fails).

SymbolsRTS is not hidden; the programmer must add "IMPORT SymbolsRTS;" to his program.

Type Sizes

BOOLEAN	1 byte
CHAR	1 byte
CARDINAL	4 bytes
INTEGER	4 bytes
BITSET8	1 bytes
BITSET	2 bytes
BITSET32	4 bytes
REAL	4 bytes
LONGREAL	8 bytes
PROC	4 bytes
CARDINAL16	2 bytes
CARDINAL8	1 byte

INTEGER16	2 bytes
INTEGER8	1 byte
CARDINAL32 = CARDINAL	
INTEGER32 = INTEGER	
Array indices	1, 2, or 4 bytes, depending on the index type
Enumerations	1 byte
Sets	1, 2, or 4 bytes, depending of the set size

SYSV

Module Keys

The module version key is a 32 bit "Fletcher" type checksum. The extra bits are currently zeroed.

The checksum is only calculated on significant characters in the DEF file, ie it does not include spaces or tabs (unless in a literal) or comments, so a DEF file can be reformatted graphically and comments added/changed and it will still compile to the same key value.

SysV ABI

Calling protocol

PPC SysV ABI is used for parameters.

HIGH() values are passed "hidden" from the ABI (available to Modula-2 programs only)

The Module's global data pointer is maintained in r13.

The Procedure's mark (frame) data pointer is maintained in r14.

PPC SysV ABI Summary

Register r1 is used as the stack pointer.

Most parameters are passed in registers; the first eight non-floating point parameters are assigned to r3..r10 in order left to right. The first eight FP parameters are assigned to f1..f8.

Floating point return values are placed in f1. Other return values into r3, and, if 64 bit, r4.

Only for registers r13..r31 and f14..f31 will the contents be preserved across procedure calls.

Any parameters after the eight GP registers (or seven FP registers) are passed in a special area of the caller's stack frame. Each of these parameters is stored extended to 32 bits in that area, except that all Floating Points parameters are converted to Double Precision (64 bits).

In general, scaler types are allocated at an address aligned to their size (eg, CHAR is aligned to 1 byte, INTEGER to 4 bytes, REAL to 4 bytes).

A RECORD is placed in memory at an alignment equal to the largest alignment value of any of its members. An ARRAY is always aligned to the alignment of its elements.

Details can be found in the document, *SystemV Application Binary Interface, PowerPC Processor Supplement*, Steve Zucker, Sunsoft and Kari Karhi, IBM - available on-line.

[StackFrame](#)

Stack Frame

	FPR Save Area	<varying size>	max is 32*8 = 256	
	GPR Save Area	<varying size>	max is 32*4 = 128	
	CR Save Area		4 bytes	
	Frame Padding to 16 bytes	<varying size>	-----,	
	Dyn array pass-by-val copy	<varying size>	Modula-2 ---,	
	pass-by-value array copy	<varying size>	Modula-2 ---'	ABI "Local Variable Area"
	Local Variable Space	<varying size>	-----,	
	for Reg Params and HIGHS	<varying size>	Modula-2	
	gp save word		4 bytes	Modula-2
mp ->	mp Back Chain		4 bytes	Modula-2 ---' ----'
	Static Link for Callee		4/0 bytes	Modula-2 ---,
	...			ABI Parameter Passing Area
	HIGHS for Callee		n*4 bytes	Modula-2
	ABI Overflow params	<varying size>	-----'	
	LR Save word		4 bytes	
sp ->	Frame Back Chain		4 bytes	

The stack frame has 16 byte(!) alignment.

Section Head: [Aglet_Implementation](#)Next subsection: [Tools](#)

Amiga Object File Linking

Mod2Lnk performs some Modula-2 specific pre-processing and then invokes the *ld* linker from the Amiga OS4 SDK to produce a standard AOS4 Elf object file. I often refer to *Mod2Lnk* as the "pre-linker".

What Mod2Lnk Does

Finds, using the current directory, any command line switches, and the "M2Lv3:" assign, the object files for all directly and indirectly imported modules used by your program.

Checks for module key [ModKeysLink](#).

Warns about [ModKeysCircle](#) between modules, which may introduce problems due to initialization order.

Generates an "xxx_start" object file which controls the proper order of module initialization and termination calls.

Linker Switches

-infil/A	Name of program module
-indir/K	Dir to search for object files ("M2Lv3:" will always be searched)
-outdir/K	Where to put the output file
-stack/K/N	Stack size to embed in executable via the "\$STACK:xxxxxx" string.
-g=-incldbg/S	Currently, specifies that local proc symbols be included in the executable.
-verbose/S	Display a list of all modules linked and the order of module inits/terms
-verbose+/S	As above, but also include info on the clients each module has.
-listfil/K	Output info to a file rather than stdout
-libname/K	Non-M2 obj library to include in the link (eg. "SDK:clib2/lib/libm.a")
-listmap/K	Ask ld to produce a link map into the specified file.
-version/S	
-help/S	

Example

A link of the program *GuideMaker* :

mod2lnk GuideMaker -libname Work:SDK/clib2/lib/libm.a

**Mod2Lnk Amiga 0.4 (11.7.2009)
OS 4.0 Copyright (c) 2004 Tom Breeden**

GuideMaker**Warning MutualImport: SimpleGUIHidden SimpleGUI****Warning MutualImport: SimpleGUISupport SimpleGUI****Warning MutualImport: GuideMakerNodeProcess GuideMakerGUI****SDK:gcc/bin/as -o GuideMaker_start.o GuideMaker_start.asm****SDK:gcc/bin/ld -o GuideMaker T:GuideMaker.lnk -q -nostdlib -x****Mod2Lnk done ErrStatus: 0000**

Section Head: [Aglet_Implementation](#)

Next subsection: [Support](#)

Tools

[M2Err](#)

Error lister

[IDLm2](#)

Interface XML to Definition module

[TGm2](#)

Test Generator

M2Err

The compiler writes out error information for each module compilation into the directory "T:" in binary format and named as <module>".err".

The small program "M2Err" will interpret and list the errors in the .err file. If the source file is accessible to M2Err, it will list the appropriate source line along with each error message.

M2Err template: -f,-w/S,-p/S

- f** should be followed by the name of the error file.
- p** causes M2Err to pause between the output of each error.
- w** causes it to open a window and write the error results into it.

If no "-f" parameter is given, it will process the most recently written .err file in T:.

Note: If you use the integrated development environment, *M2IDE*, you will not need to run *M2Err* yourself.

IDLm2

IDLm2 is a program to take the AOS4 library interface description XML file and produce a Modula-2 Definition module file that can (with some additional edits) be used to call the library directly from M2 programs and implementation modules. ie., something of a Modula-2 analog of the SDK program *IDLTool* .

PgmTemplate

-infil/A,-impl=implement/S,-version/S,-help/S

-infil The XML file describing the Amiga library interface(s).
 -impl Produce a skeleton Implementation module as well as the Definition module.

The version with this release is v1.1 (11/23/2011).

The XML files for the standard Amiga Libraries are distributed with the SDK and named appropriately for their library, eg, for diskfont.library this is "diskfont.xml".

Output

The output file will be will be a draft version of the DEFINITION MODULE for the interface(s), named by adding "Interfaces" to the library name, eg, "diskfontInterfaces.def", and, optionally, a draft version of the IMPLEMENTATION MODULE with autoopen code included.

These will require a little editing before use.

IDLm2 will automatically create the declarations for a constant containing the library name, and variables to hold the library base and the interface (pointer) itself. eg,

```
CONST diskfontLibName = "diskfont.library";
VAR  DiskfontBase :LibraryPtr;
TYPE DiskfontIFacePtr = POINTER TO DiskfontIFace;
VAR  IDiskfont  :DiskfontIFacePtr;
```

And, of course, the RECORD containing the definitions of the interface's procedures. *IDLm2* translates the parameters to each procedure and function into compatible, but often not ideal, M2 data types. A little knowledge of the SDK AutoDocs and the Aglet implementation will often allow you to edit the initially produced definition into something equivalent, but better suited to Modula-2. For example you can often avoid C's lack of arrays as parameters knowing that *Aglet M2 PPC* passes all arrays (and records) by reference, whether VAR or non-VAR. Thus, the original output of *IDLm2* :

```
OpenOutlineFont :PROCEDURE(DiskfontIFacePtr, (*name*)ADDRESS, ListPtr,  
                             (*flags*)CARDINAL):OutlineFontPtr;
```

can be manually edited to a better declaration:

```
OpenOutlineFont :PROCEDURE(DiskfontIFacePtr, (*name*)ARRAY OF CHAR,  
                             ListPtr, (*flags*)BITSET32):OutlineFontPtr;
```

IDLm2 knows about a few of these optimizations (eg, it will always translate "const struct TagItem *" as "ARRAY OF TagItem", but most you will need to do yourself (if you want the best DEFINITION file).

See [ALIB](#) for more information.

In most cases, for Amiga libraries, I have kept the constants, types, and variables used by the library in separate DEFINITION module(s) from the interface's one. For the Reaction classes, however, I have added the class-associated types, consts, and variables to the *IDLm2* produced DEFINITION file.

TGm2 Test Generator Program

tgM2 is a program to aide in unit testing of Modula-2 software. It takes as input a user-prepared test script (*.ts file) which specifies one or more tests of a procedure or code block to be executed and reported upon. *tgM2* writes out a Modula-2 program for this test suite. The output program itself writes its results to standard output.

tgM2 owes its existence and most of its architecture to the Ada 95 test driver generator program *tg* for Ada written by Andre Spiegel, along with the version of *tg* for Modula-2 (still an Ada 95 program) adapted to generate Modula-2 scripts done by Ralf Reissing.

(See more documentation at "<http://www.modula2.org/projects/tgM2.php>")

A simple program, *TestManager*, has been written for automated testing that uses *tgM2* and its *.ts files to generate, run, and report the results on large batches of test cases.

The beginnings of a compiler regression testing suite is located in the "Validate/" subdirectory of the *Aglet M2 PPC* distribution.

Section Head: [Aglet_Implementation](#)Next subsection: [Amiga_Specific](#)

AgletM2 Support Modules

General Notes

These [LibAutoOpen](#) any Amiga libraries they use (in the module initialization code).

Only a few have some "thread safety" build into them. e.g, the modules Storage and SystemStorage, do wrap their internal critical sections in semaphores.

[AmigaMods](#)

[ReactionMods](#)

[ISOMods](#)

[SysMods](#)

[ExperimentalMods](#)

Section Head: [Support](#)
Next subsection: [ReactionMods](#)

Amiga Library Definition Modules

The SDK include files defining the types and constants used by the Amiga libraries are the model for the set of Modula-2 DEFINITION modules serving the same purpose. Most of the current definitions files were derived from SDK v53.20.

In many cases, multiple related *.h files have been combined into one *.def file. Also, in some cases the C definitions have been converted to utilize the more expressive type system of M2, often by introducing appropriate SET types where int or unsigned have been used in C. The IMPLEMENTATION module for these *.h file conversions is usually empty.

The AOS4 library interfaces are represented by one or more RECORD type definitions representing the interface jump table. These have usually been placed into a Modula-2 DEFINITION module named after the Amiga library, eg, "DEFINITION MODULE IntuitionInterfaces".

The IMPLEMENTATION module for these interface definitions auto-opens the library and interface on module initialization, and closes them on module termination. See more details in [ALIB](#) and [LibAutoOpen](#)

[AmigaModsList](#)

.

Amiga Library Modules List

Alerts	-
AmigaDOS	-
AmigaDOS2	-
AmigaDOSExt	-
AmigaDosInterfaces	-
AmigaDOSNotify	-
AmigaDOSProcess	-
AmigaDOSTags	-
AmigaGuide	-
AmigaGuideInterfaces	-
AmigaInput	-
AmigaInputInterfaces	-
AmigaSemaphores	-
AnimationClass	-
Application	-
ApplicationInterfaces	-
Areas	-
Asl	-
aslInterfaces	-
AVL	-
BitMapShare	-
Blit	-
BoopsiAlib	-
BoopsiClasses	-
BoopsiImplementor	-
BoopsiUser	-
bulletInterfaces	-
ClipBoardDevice	-
Clipping	-
ColorWheel	-
ColorWheelInterfaces	-
Composite	-
ConsoleDevice	-
consoleInterfaces	-
Copper	-
CustomHardware	-
DataTypes	-
DataTypesClass	-
DataTypesInterfaces	-
DiskFont	-
diskfontInterfaces	-
DiskFontTag	-
DisplayInfo	-
Docky	-
DockyInterfaces	-

Drawing	-
elf	-
elfInterfaces	-
Emulation	-
ExecBase	-
ExecInterfaces	-
ExecTags	-
FileHandler	-
FontPrefs	-
GadTools	-
GadToolsInterfaces	-
Gels	-
Glyph	-
Graphics	-
GraphicsBase	-
GraphicsInterfaces	-
HunkInterfaces	-
Hunks	-
Icon	-
iconInterfaces	-
IFFParse	-
IFFParseInterfaces	-
Initializers	-
InputEvents	-
Interfaces	-
Interrupts	-
Intuition	-
Intuition2	-
IntuitionBase	-
IntuitionGUI	-
IntuitionInterfaces	-
IntuitionNotify	-
IntuitionPlugins	-
IODevices	-
KeyMap	-
KeyMapInterfaces	-
LayersInterfaces	-
Libraries	-
LibrariesExtended	-
Lists	-
Locale	-
LocaleInterfaces	-
Memory	-
ModeId	-
Monitor	-
NewMenus	-
NewStyleDevice	-
Nodes	-
OTErrors	-

PictureClass	-
Ports	-
Preferences	-
PrefHdr	-
PrinterDevice	-
PrinterGraphics	-
Rasters	-
ReactionPrefs	-
Regions	-
Resident	-
RexxIO	-
RexxStorage	-
rexsyslibInterfaces	-
RxsLib	-
ScreensPrefs	-
SoundClass	-
Sprites	-
Tasks	-
Text	-
TextClass	-
TextClipInterfaces	-
TimerDevice	-
timerInterfaces	-
TimeSync	-
TimeSyncInterfaces	-
TimeZoneInterfaces	-
UserInterfPrefs	-
Utility	-
UtilityInterfaces	-
UtilityTags	-
Views	-
wbInterfaces	-
WorkBench	-

Reaction Definition Modules

These are derived from the AmigaOS 4.1 C includes and XML files for Reaction.

See also the Aglet experimental module [SimpleGUI](#). This is the beginnings of a higher level interface layered on Reaction, but insulating the programmer from many of the details of programming for Reaction.

Similar Aglet "second level" modules for Amiga programming are SimpleRequesters, SimpleMenus, SimpleImageHandler, SimpleScreens, [SimpleGraphics](#).

[ReactionModsList](#)

.

Amiga Reaction Modules List

Reaction	-
ReactionARexx	-
ReactionBevel	-
ReactionBitMap	-
ReactionButton	-
ReactionCheckBox	-
ReactionChooser	-
ReactionClickTab	-
ReactionDateBrowser	-
ReactionDrawList	-
ReactionFiller	-
ReactionFuelGauge	-
ReactionGetColor	-
ReactionGetFile	-
ReactionGetFont	-
ReactionGetScreenMode	-
ReactionGlyph	-
ReactionInteger	-
ReactionLabel	-
ReactionLayout	-
ReactionLib	-
ReactionListBrowser	-
ReactionPalette	-
ReactionPenmap	-
ReactionPrefs	-
ReactionRadioButton	-
ReactionRequester	-
ReactionScroller	-
ReactionSketchBoard	-
ReactionSlider	-
ReactionSpace	-
ReactionSpeedBar	-
ReactionString	-
ReactionTextEditor	-
ReactionVirtual	-
ReactionWindow	-
UserInterfPrefs	-

Section Head: [Support](#)
 Next subsection: [ExperimentalMods](#)

Aglet Sysmod Library

Amiga Specific

AccessFontPrefs	Returns info on the five Prefs-specified fonts.
AccessScreenPrefs	Returns list of the Prefs-specified Public Screens.
AmigaTimer	Simple Interface for using the Amiga Timer Device
AnsiTerminal	CLI window as an ANSI standard video terminal
ArgsSupport	AmigaOS style arguments and templates
Break	Test SigBreakCtrlC (^C) for an Amiga process
FontSupport	Connect to Amiga common fonts via "generic" font names
MachineEnv	Access to screen, OS, and CPU config details
PipeIO	IO implementation suitable for PIPES
SimpleRexx	patterned on Michael Sinz' original C version
SimpRexx1	support for creating ARexx Command Messages
TagsUtils	For setting up Tag arrays used by AOS procedure calls
TagsUtilsDyn	As above, but the Tag array size is not fixed at compile time.
IntuiSupport	Amiga window and screen support layer [Superceded by SimpleGUI]
IDCMPraw	IDCMP message handling process for IntuiSupport [Superceded by SimpleGUI]

GUI

FileBrowsing	Use the ASL file browser
(see also SimpleGUI , et.al. under ExperimentalMods)	

Data Structures

BinaryTree	Binary Trees, pointer based genericity
BSEARCH	Binary Search
HashT	Hash Table
ListV2Processor	Generic List Processor, with element ordering.
Lists0	Simpler list processing layered directly on Exec Lists.
QSORT	QuickSort
Queue	Generic queues
RandomFiles	Random access to fixed size record file.
Set0	bitsets larger than 32 elements
STACK	Generic stacks

Math

MatrixOperations
RandomNumbers

RandInts	Processing groups of random integers
SimultaneousEquations	
Arith64	Current substitute for the lack of LONGINTEGER / LONGCARDINAL
BigInt	Arithmetic on extremely large integers, up to about 48 decimal digits.

String Handling

Str0, Str1, Str1NC Str2, Str3	Null delimited static string handling procedures
StrValue MyConversions	String <-> Number conversions see also ISO mods Strings, WholeConv, RealConv)
DynStr0, DynStr1 DynStr2, DynStr2A	Dynamically allocated, null delimited string procedures
DynStr3, DynStr3A DynStr3Policies	Dynamically reallocated, null delimited string procedures
StrSubstitutes StrMacros	String token substitution in blocks of text String expansion macros, with parameters
TextBlocks TbChans	Processing large blocks of text in memory ISO Channel IO interface to TextBlocks
RegularExpressions	as in AmigaOS

DateTime

DateSupport	Julian date processing from 1/1/1900
UnixTime	Unix type times, seconds from 1/1/1970

Directory Processing

DirUtils	Procs for getting File Directory contents and attributes.
DirUtil0, DirUtil1	
DirUtil2, DirUtil3	

Etc

Assertions	Programmer inserted assertions (or warnings).
ModDebug	Programmer inserted debugging statements, supporting enable/disable
MaxMin	for basic types
CriticalSections	mutual exclusion semaphores w/ producer/consumer counts
RTFWrite	Write support for RTF files
TextIOHelper	for using ISO TextIO to read text files line by line correctly
FontSupport	very simple generic font selection.
FileNameParser	Separate file specifications into device, path, name, ext.

TextIOHelper Module

I found myself repeatedly making the same logic errors in simply reading plain text files using the procedures in the ISO TextIO module directory.

TextIOHelper was written to make it easier to get it right. And it adds a couple of other fillips: skipping blank lines and using expandable strings to handle long lines.

```
|
| PROCEDURE ReadNextLine(cid:ChanId; VAR OneLine:ARRAY OF CHAR):ReadResults;
| PROCEDURE ReadNextNonBlankLine(cid:ChanId; VAR OneLine:ARRAY OF CHAR; VAR
|           NumSkipped:CARDINAL):ReadResults;
|
| PROCEDURE dReadNextLine(cid:ChanId; VAR dOneLine:DynStr):ReadResults;
|   (* dOneLine must exist or .str be NIL *)
| PROCEDURE dReadNextNonBlankLine(cid:ChanId; VAR dOneLine:DynStr; VAR
|           NumSkipped:CARDINAL):ReadResults;
|
| (*
|   ReadNextLine - silently truncates any line that won't fit in the OneLine
|                 parameter
|                 - never returns endOfLine. For empty line returns allRight with
|                 empty OneLine
|                 - endOfInput is returned only on the call AFTER the one that
|                 returned the last line allRight.
|
|   dReadNextLine - reallocates dOneLine to be larger if necessary.
| *)
|
```

Aglet Experimental Mods

Amiga Specific

IconSupport

For creating and updating Workbench icons.

GUI

[SimpleGUI](#)

Implemented as a layer above Reaction

SimpleMenus

System standard menus - use with SimpleGUI

SimpleRequesters

Procedures to invoke various requesters with a call

CustomGlyphs

Provides a number of Boopsi drawlist figures

SimpleImageHandler

Provides standard and custom Boopsi image and glyph objects

SimpleScreens

For using public screens with your app.

Graphics

[SimpleGraphics](#)

Directory Processing

DirUtilsDyn

As DirUtils, but using unlimited length strings for file specifications.

Process Control

OsRun

Module for programatically starting independent CLI processes

Etc

[Module_Obj](#)

OO programming in unextended Modula-2

SimpleGUI Module

SimpleGUI is a module that provides easy access to Amiga Reaction window and gadget facilities.

Its approach is essentially that of "simple things should be simple" and that "complex things should be possible" with development from a simple starting point.

Reaction Details 'Abstracted out'

Though its implementation is Reaction based, it does abstract out the presented GUI ideas to an extent. (It actually derives from an Amiga-inspired Windows M2 implementation of SimpleGUI that has not yet been fully re-implemented).

Multi-Window Input Thread

Besides the procedures for creating, customizing, and positioning windows and gadgets, SimpleGUI handles all the details of starting a higher-priority thread to monitor and buffer the incoming messages from multiple windows/gadgets.

Types, Attributes, and Procedures

A set of available gadgets, not dissimilar to Reaction's, is presented for use:

TYPE Gadgs = (LabelSG, StringSG, EditSG, IntegerSG, RealSG, ButtonSG, CheckBoxSG, RadioButtonsSG, ScrollerSG, ChooseSG, MenuSG, ListSG, FuelGaugeSG, SpaceSG, BoxSG, ContainerSG);

One or more attributes, or "modifiers", can be applied to customize the gadgets' features and behavior:

Modifiers = (Boxed, Static, Transparent, Emphasized, Sorted(*NYI*), HighCapacity, ReadOnly, Disabled, SizeToFitH, SizeToFitV, EqualSize, FixedH, FixedV, GreedyH, GreedyV, Hscroller, Vscroller, UserData);

A number of different events may be retrieved in a simple procedure call by your program from the SimpleGUI event queue. Each event comes with associated information attached, such as its originating gadget/window, the mouse position, the menu select id:

WndoEvnts = (NullEv, CloseEv, IconifyEv, UniconifyEv, SizeEv, PositionEv, FocusInEv, FocusOutEv, SelectEv, DblClickEv, MenuEv, KbdEv, HelpEv, LButtonDnEv, LButtonUpEv, RButtonDnEv, RButtonUpEv, MButtonDnEv, MButtonUpEv, MMoveEv, PacerEv);

Procedures are used for creating, customizing, positioning, altering, etc, the gadgets. Some of the procedures apply to any of the Gadgs types, and other

procedures are specific for one or two types.

A Work-In-Progress

SimpleGUI is definitely a work-in-progress, though complete enough to produce non-trivial GUIs (see the M2IDE, and program GuideMaker). New modifiers, increased orthogonality, and new Gadg types will be added to better take advantage more of Reaction's capabilities and to make using SimpleGUI easier.

Please see the documentation residing in the comment section of [SimpleGUIDef](#) for more details.

```
(*#####*)
DEFINITION MODULE SimpleGUI;      (* $VER: SimpleGUI.def 0.3 (6.12.2011) *)
(*#####*)
(*)
Copyright (C) 2002 Thomas Breeden.
```

Permission to use, copy, modify and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both the copyright notice and this permission notice appear in supporting documentation. Thomas Breeden makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.
*)

```
FROM SYSTEM      IMPORT ADDRESS;
FROM DynStr0     IMPORT DynStr;

FROM BoopsiUser   IMPORT Object;

FROM SimpleImageHandler IMPORT Images;
FROM SimpleMenus IMPORT MenuItemId, sMenu;
```

```
(*
Gadgs
```

```
-----
  ContainerSG      Contains other Gadgs (eg, a Reaction Layout)

  LabelSG          Static modifier labels are more efficient.

  StringSG         Typing ENTER into this Gadg will always generate a SelectEv "event"

  EditSG
  IntegerSG
  ButtonSG
  CheckBoxSG

  ChooseSG         a "one-is-always-selected" gadget of a number of text choices,
                  displayed as a single line of text, with a cycle-through button on
                  the left, and clicking on the text itself will pop-up all choices.
                  eg, program modes, or source of window contents

  MenuSG           a "do-this-action" or "use-this-datum" gadget, expected to be
                  displayed as a small button that results in a drop-down list for
                  picking from. eg, previous files opened, or a compact means of
                  presenting multiple next actions.

  ListSG           Box with vertical scroll bar containing multiple possible selections.
                  May be none selected or more than one (NYI). Selected item remains
                  highlighted.

  SpaceSG          Space filler
                  Space gadgets are generally given last priority in determining their
                  actual size, but they always have a minimum H/V size below which they
                  will not go.

  ScrollerSG       Sliding block within a vertical or horizontal background, with arrow
                  icons for small steps.

  RealSG           <Not Yet Implemented>
```

RadioButtonsSG <Not Yet Implemented>
 FuelGaugeSG <Not Yet Implemented>
 BoxSG <Not Yet Implemented>

Modifiers

Boxed -Surround the gadg with a box
 -works with ContainerSG, LabelSG (non-static only), ListSG (multicolumn only), CheckBoxSG, IntegerSG, SpaceSG, CheckBoxSG
 -for multi column list browsers means: put in vertical separators between columns

Static -Promises that the contents of a Label Gadg will never change once created.

Transparent -As Reaction's Button_Transparent tag. With a hook background can be transparent for ButtonSG or LabelSG.

Emphasized -Label will be drawn with the HighLightPen.
 HighCapacity -for StringSG, means it can handle very long strings.
 -for ButtonSG, ChooserSG, MenuSG, ListSG, any images are hi-res and/or multicolor

ReadOnly -Won't respond to clicks, but does not have the disabled appearance.
 Disabled

SizeToFit -generally, sets the gadg size to just big enuf for its contents
 SizeToFitH -AsgGadgSize/Dimensions() will override these flags (in the selected dimension)

SizeToFitV

EqualSize -for Containers, all included Gadgs with have equal width.
 -works best with a set of similar gadgets, eg, of buttons or labels.

FixedH -size of gadget horizontally cannot vary from the minsiz.x
 FixedV -size of gadget horizontally cannot vary from the minsiz.y

GreedyH -IF GreedyH/V, the gadget is given priority in satisfying its requested size.
 GreedyV

Hscroller -Include a scroller along with this Gadg.
 Vscroller

UserData -for ListSG, indicates that one ADDRESS per column can be attached as user data on each row

Sorted <not yet implemented>

(* currently, Amiga: SizeToFit works only with Buttons, Labels, and Containers -many gadgs are implicitly SizeToFitH so they won't expand vertically *)

SubModifiers

Toggle <not yet implemented>
 PushButton <not yet implemented>

NotSelectable -for multicolumn ListSG, used to mark one or more columns passive.

Img -Use a Boopsi image object instead of text for this Button, List Column Item, or MenuSG item.
-Vector drawn images will resize with the button, other images may not.

MonoText -Use a monospaced font for this Label or Edit Gadg.

Rjust -Specifies column justification for column in multicolumn List Box.

WndoModifiers

Busy -The HourGlass pointer is displayed.

NoCloseGad -The window does not get a close gadget

NoSizeGad -The window does not get a size gadget.

IconifyGad -The window gets an iconify gadget.

NoActivateWndo -The window is not activated when opened.

HelpW -Window will return a HelpEv event if the Help key is pressed. Also GadgetHelp bubbles are enabled.

PubScreen -The window may appear on an existing, public, screen.

WndoEvnts

NullEv

CloseEv

IconifyEv

UniconifyEv

SizeEv

PositionEv

FocusInEv

FocusOutEv

SelectEv

DblClickEv

MenuEv

KbdEv

HelpEv -Help key was pressed on keyboard.

LButtonDnEv

LButtonUpEv

RButtonDnEv

RButtonUpEv

MButtonDnEv

MButtonUpEv

MMoveEv

PacerEv -Tenth of a second events (eg, IntuiTicks)

GadgStatus

On

Off

Hit

DoubleClicked

Disabld

Changd <Not Yet Implemented>

*)

(*

"NYI" = <not yet implemented>

*)

TYPE Gadg;
Wndo;

TYPE Gadgs = (LabelSG, StringSG, EditSG, IntegerSG, RealSG(*NYI*), ButtonSG, CheckBoxSG, RadioButtonsSG(*NYI*), ScrollerSG, ChooseSG, MenuSG, ListSG, FuelGaugeSG(*NYI*), SpaceSG, BoxSG(*NYI*), ContainerSG);

CONST TwoD = **RECORD** x, y:INTEGER; **END**;
Ignore = MIN(INTEGER); (* value for TwoD, and some other specs, meaning ignore this field in the call *)

TYPE Orientations = (Horizontal, Vertical);
OrientationSet = **SET OF** Orientations;
CONST BothDimensions = OrientationSet{Horizontal, Vertical};

TYPE Directions = (Left, Right, Up, Down, CenterH, CenterV);
DirectionSet = **SET OF** Directions;

Zpos = [Up..Down];

Sizes = (Huge, Big, NormalSize, Small, Tiny, CustomSize, UnsetSize);
DimensionSpec = **RECORD**
 minsize,
 maxsize :TwoD;
END;

TYPE Modifiers = (Boxed, Static, Transparent, Emphasized, Sorted(*NYI*), HighCapacity, ReadOnly, Disabled, SizeToFitH, SizeToFitV, EqualSize, FixedH, FixedV, GreedyH, GreedyV, Hscroller, Vscroller, UserData);

CONST ModifierSet = **SET OF** Modifiers;
SizeToFit = ModifierSet{SizeToFitH, SizeToFitV};
FixedSize = ModifierSet{FixedH, FixedV};
Greedy = ModifierSet{GreedyH, GreedyV};

TYPE SubModifiers = (Toggle(*NYI*), PushButton, NotSelectable, Img, MonoText, RJust);
SubModifierSet = **SET OF** SubModifiers;

TYPE WndoModifiers = (Busy, NoCloseGad, NoSizeGad, IconifyGad, NoActivateWndo, HelpW, PubScreen);
WndoModifierSet = **SET OF** WndoModifiers;

WndoEvnts = (NullEv, CloseEv, IconifyEv, UniconifyEv, SizeEv, PositionEv, FocusInEv, FocusOutEv, SelectEv, DblClickEv, MenuEv, KbdEv, HelpEv, LButtonDnEv, LButtonUpEv, RbuttonDnEv, RButtonUpEv, MButtonDnEv, MButtonUpEv, MMoveEv, PacerEv);

WndoEvtSet = **SET OF** WndoEvnts;

EvtInfo = **RECORD**
 CASE ev:WndoEvnts **OF**
 HelpEv,
 SelectEv,
 DblClickEv: g :Gadg;
 col :CARDINAL; |
 KbdEv: ch :CHAR; |
 LButtonDnEv..MButtonUpEv, MMoveEv:

```

                                gb   :Gadg;
                                x, y :INTEGER;  (* relative to gb Gadg *) |
MenuEv:                          mi   :MenuItemId; |
                                ELSE
                                END;
                                END;
EvtInfoPtr      = POINTER TO EvtInfo;

GadgStatus      = (On, Off, Hit, DoubleClicked, Disabld, Changd);
GadgStatusSet   = SET OF GadgStatus;

(* ----- Wndos ----- *)

PROCEDURE NewWndo(siz:TwoD; orient:Orientations; just:DirectionSet;
                 modif:WndoModifierSet):Wndo;
  (* also creates the outermost container gadg, using "orient" and "just" *)
  (* a siz of ignore, ignore will let Reaction determine the best size *)

PROCEDURE WndoContainer(win:Wndo):Gadg;

PROCEDURE OpenWndo(win:Wndo; position:TwoD; Title:ARRAY OF CHAR);
PROCEDURE CloseWndo(win:Wndo);
PROCEDURE IsWndoOpen(win:Wndo):BOOLEAN;
  (* ok to call on NullWndo() *)
PROCEDURE IconifyWndo(win:Wndo);
PROCEDURE UniconifyWndo(win:Wndo);

PROCEDURE InclWndoModifier(wndo:Wndo; wm:WndoModifiers);
PROCEDURE ExclWndoModifier(wndo:Wndo; wm:WndoModifiers);
PROCEDURE GetWndoModifiers(wndo:Wndo; VAR wms:WndoModifierSet);

PROCEDURE DisposeWndo(VAR win:Wndo);
  (* also disposes any gadgs in the wndo; ok to call if win open or closed *)
  (* ok to call on NullWndo() *)

PROCEDURE AttachWndoMenu(win:Wndo; m:sMenu);
PROCEDURE DetachWndoMenu(win:Wndo; VAR m:sMenu);
PROCEDURE QueryWndoMenu(win:Wndo; VAR m:sMenu);
  (* menus are created, constructed, or destroyed outside of SimpleGUI *)

PROCEDURE GetWndoEvt(win:Wndo; VAR ev:EvtInfo):BOOLEAN;
  (* safe to call on NullWndo(), BUT NullWndo DOES NOT mean "any window" *)

PROCEDURE WaitWndoEvt(win:Wndo);
PROCEDURE WaitWndoEvtTimed(win:Wndo; TimeoutSecs:CARDINAL);  (* DOES NOT SEEM TO WORK
                                                             IN A THREAD *)
  (* NullWndo means "any window" -> CURRENTLY ONLY ONE PROCESS CAN WAIT ON NullWndo()
*)

PROCEDURE SetWndoPacer(win:Wndo; On:BOOLEAN);
  (* opens in Off mode *)
PROCEDURE TriggerWndoEvt(win:Wndo; evt:EvtInfo);
  (* TriggerWndoEvt(NullWndo()) means a WaitWndoEvt(NullWndo()) will end the wait,
  but no event is currently queued *)

PROCEDURE SetSpecificScreen(wndo:Wndo; ScreenName:ARRAY OF CHAR):BOOLEAN;
  (* call this just after NewWndo() (or after OpenWndo()) but jumping screens is NOT
  TESTED *)
PROCEDURE GetSpecificScreen(wndo:Wndo; VAR ScreenName:ARRAY OF CHAR);

```

```

PROCEDURE AsgWndoTitle(win:Wndo; Title:ARRAY OF CHAR);
PROCEDURE AsgWndoPos(win:Wndo; Position, Size:TwoD);
PROCEDURE AsgWndoAltPos(win:Wndo; AltPosition, AltSize:TwoD);
  (* use Ignore const for unwanted options *)
  (* will change wndo immediately if open and in the corresponding Pos
  (regular/alternate) *)

PROCEDURE AsgWndoZPos(win:Wndo; UpDown:Zpos);
  (* will change wndo immediately *)

PROCEDURE GetWndoPos(win:Wndo; VAR Position, Size:TwoD);
PROCEDURE GetWndoNormPos(win:Wndo; VAR Position, Size:TwoD);
PROCEDURE GetWndoAltPos(win:Wndo; VAR Position, Size:TwoD);
PROCEDURE IsWndoAltPos(win:Wndo):BOOLEAN;

(* ----- Gadgs ----- *)

PROCEDURE NewGadg(typ:Gadgs; modif:ModifierSet):Gadg;
PROCEDURE NewContainer(orient:Orientations; modif:ModifierSet):Gadg;
  (* {CenterH, CenterV} is the initial justification. Use AsgGadgJustification() to
  change that *)
PROCEDURE NewButton(text:ARRAY OF CHAR; modif:ModifierSet):Gadg;
PROCEDURE NewCheckBox(text:ARRAY OF CHAR; modif:ModifierSet):Gadg;
PROCEDURE NewEditBox(text:ARRAY OF CHAR; modif:ModifierSet):Gadg;
PROCEDURE NewStringBox(text:ARRAY OF CHAR; modif:ModifierSet):Gadg;
PROCEDURE NewIntegerBox(initial, min, max:INTEGER; modif:ModifierSet):Gadg;

PROCEDURE NewChoose(modif:ModifierSet):Gadg;
PROCEDURE NewMenuGadg(text:ARRAY OF CHAR; modif:ModifierSet):Gadg;
PROCEDURE NewListBox(modif:ModifierSet):Gadg;
PROCEDURE NewListBoxCols(modif:ModifierSet; NumCols:CARDINAL; ColModif:ARRAY OF
  SubModifierSet):Gadg;

PROCEDURE NewSpace(siz:Sizes; modif:ModifierSet):Gadg;
PROCEDURE NewLabel(text:ARRAY OF CHAR; modif:ModifierSet):Gadg;

PROCEDURE NewScroller(lowKnob, highKnob, min, max:INTEGER; orient:Orientations;
  modif:ModifierSet):Gadg;

PROCEDURE DisposeGadg(VAR gad:Gadg);

PROCEDURE ActivateGadg(gad:Gadg):BOOLEAN;
  (* usually, leave this to the user *)

PROCEDURE AsgGadgDimensions(gad:Gadg; orients:OrientationSet; siz:DimensionSpec);
PROCEDURE AsgGadgSize(gad:Gadg; orients:OrientationSet; siz:Sizes);
PROCEDURE AsgGadgOrientation(gad:Gadg; orient:Orientations);
PROCEDURE AsgGadgJustification(gad:Gadg; just:DirectionSet);
  (* for a container, the justification value applies to the gadgets within it *)
PROCEDURE AsgGadgInteger(gad:Gadg; ival:INTEGER);
PROCEDURE AsgGadgText(gad:Gadg; text:ARRAY OF CHAR; ResizeFit:BOOLEAN);
  (* ResizeFit ignored on Amiga currently *)
PROCEDURE AsgGadgdText(gad:Gadg; dtext:DynStr);
PROCEDURE AsgGadgImg(gad:Gadg; image:Images (*,ResizeFit:BOOLEAN*));
PROCEDURE AsgGadgHelpText(gad:Gadg; dtext:DynStr);

PROCEDURE RefreshGadg(gad:Gadg);

```

```
PROCEDURE ClrGadgGraphics(gad:Gadg; IncludeBorder:BOOLEAN); (* DOES NOT WORK 7/27/03 *)
```

```
PROCEDURE GetGadgType(gad:Gadg):Gadgs;
```

```
PROCEDURE GetGadgDimensions(gad:Gadg; VAR siz:DimensionSpec);
```

```
PROCEDURE GetGadgSize(gad:Gadg; orient:OrientationSet; VAR siz:Sizes);
```

```
PROCEDURE GetGadgOrientation(gad:Gadg; VAR orient:Orientations);
```

```
PROCEDURE GetGadgJustification(gad:Gadg; VAR just:DirectionSet);
```

```
PROCEDURE GetGadgText(gad:Gadg; VAR text:ARRAY OF CHAR);
```

```
PROCEDURE GetGadgdText(gad:Gadg; VAR dtext:DynStr);
```

```
  (* if dtext.str = NIL, this proc will create it *)
```

```
PROCEDURE GetGadgHelpText(gad:Gadg; VAR dtext:DynStr);
```

```
PROCEDURE GetGadgInteger(gad:Gadg; VAR ival:INTEGER);
```

```
  (* returns -1 on a list is none is selected *)
```

```
PROCEDURE InclGadgModifiers(gad:Gadg; gms:ModifierSet);
```

```
PROCEDURE ExclGadgModifiers(gad:Gadg; gms:ModifierSet);
```

```
  (* Incl/ExclGadgModifier() implementation started, not implemented for all  
gadgs/modifiers *)
```

```
PROCEDURE GetGadgModifiers(gad:Gadg; VAR gms:ModifierSet);
```

```
PROCEDURE InclGadgSubModifiers(gad:Gadg; gsms:SubModifierSet);
```

```
PROCEDURE ExclGadgSubModifiers(gad:Gadg; gsms:SubModifierSet);
```

```
PROCEDURE GetGadgSubModifiers(gad:Gadg; VAR gsms:SubModifierSet);
```

```
PROCEDURE InclGadgStatus(gad:Gadg; gs:GadgStatus);
```

```
PROCEDURE ExclGadgStatus(gad:Gadg; gs:GadgStatus);
```

```
PROCEDURE GetGadgStatus(gad:Gadg; VAR gss:GadgStatusSet):GadgStatus;
```

```
  (* reading a "Hit" clears it *)
```

```
  (* func result will always be "Hit" for a button, "On" or "Off" for a checkbox *)
```

```
    (* set/reset Disabld via Incl/ExclGadgModifier(), not here *)
```

```
PROCEDURE AddGadg(container:Gadg; gad:Gadg);
```

```
PROCEDURE GetContainer(gad:Gadg):Gadg;
```

```
PROCEDURE GetFirstContainedGadg(cont:Gadg):Gadg;
```

```
PROCEDURE GetNextContainedGadg(cont:Gadg):Gadg;
```

```
PROCEDURE NullGadg():Gadg;
```

```
PROCEDURE NullWndo():Wndo;
```

```
(* --- Edit Gadg Procedures --- *)
```

```
PROCEDURE GetGadgCursor(gad:Gadg; VAR x, y:INTEGER); (* from 0 *)
```

```
PROCEDURE SetGadgCursor(gad:Gadg; x, y:INTEGER);
```

```
PROCEDURE SetGadgCursorX(gad:Gadg; x:INTEGER);
```

```
PROCEDURE SetGadgCursorY(gad:Gadg; y:INTEGER);
```

```
PROCEDURE ExistsGadgSelectText(gad:Gadg):BOOLEAN;
```

```
PROCEDURE GetGadgSelectBlock(gad:Gadg; VAR row0, col0, row1, col1:CARDINAL):BOOLEAN;
```

```
PROCEDURE SetGadgSelectBlock(gad:Gadg; row0, col0, row1, col1:CARDINAL);
```

```
PROCEDURE GetGadgSelectText(gad:Gadg; VAR dtext:DynStr);
```

```
  (* creates dtext if .str is NIL. Clears the selection *)
```

```
PROCEDURE DelGadgSelectText(gad:Gadg):BOOLEAN;
```

```
PROCEDURE ReplGadgSelectText(gad:Gadg; dtext:DynStr):BOOLEAN;
```


(* --- Scroller Gadg Procedures --- *)

```

PROCEDURE GetMinMax(gad:Gadg; VAR min, max:INTEGER);
PROCEDURE AsgMinMax(gad:Gadg; min, max:INTEGER);
PROCEDURE GetKnob(gad:Gadg; VAR low, high:INTEGER);
PROCEDURE AsgKnob(gad:Gadg; low, high:INTEGER);

```

(* --- Choice/Menu Gadg Procedures --- *)

```

PROCEDURE AddGadgText(gad:Gadg; text:ARRAY OF CHAR);
  (* also works for EditGadg, LabelGadg *)
PROCEDURE RemoveGadgText(gad:Gadg; text:ARRAY OF CHAR);
  (* PROCEDURE AsgGadgInteger() sets selected choice *)

```

(* --- List Box Gadg Procedures --- *)

(* for single col gadgs only *)

```

PROCEDURE AddGadgListImg(gad:Gadg; Img:Images);
PROCEDURE AddGadgListName(gad:Gadg; Name:DynStr);
  (* at end of list, Name is "deep" copied *)
PROCEDURE InsertGadgListName(gad:Gadg; EntryIndex:INTEGER; Name:DynStr);
  (* New name ends up at this entry in the list, -1 means at end of list, 0 is
  beginning of list *)
PROCEDURE AddGadgListNames(gad:Gadg; Names:ARRAY OF DynStr; NumNames:INTEGER);
PROCEDURE InsertGadgListNames(gad:Gadg; EntryIndex:INTEGER; Names:ARRAY OF DynStr;
  NumNames:INTEGER);
PROCEDURE RemoveGadgListName(gad:Gadg; ColNum:CARDINAL; Name:DynStr);

```

(* for multi-col gadgs *)

```

PROCEDURE AddGadgListNameCols(gad:Gadg; NumRows:INTEGER; Names:ARRAY OF DynStr);
  (* across the row, must be numcols x NumRows names in the array *)
  (* the input Names are copied, not just referenced *)
PROCEDURE AddGadgListMixedCols(gad:Gadg; NumRows:INTEGER; Names:ARRAY OF DynStr;
  Imgs:ARRAY OF Images);
  (* names and images filled in order across row *)

```

```

PROCEDURE InsertGadgListNameCols(gad:Gadg; EntryIndex:INTEGER; Names:ARRAY OF DynStr);
  (* NYI *)

```

```

PROCEDURE ClearGadgListNames(gad:Gadg; RefreshDisplay:BOOLEAN);
  (* also clears any selection *)

```

```

PROCEDURE RemoveGadgListEntry(gad:Gadg; EntryIndex:INTEGER);
  (* removes entire row *)

```

```

PROCEDURE GetGadgListEntry(gad:Gadg; EntryIndex:INTEGER; ColNum:CARDINAL; VAR
  Name:DynStr);
  (* the DynStr must be already allocated or its .str field NIL *) (* deallocated
  string if too few *)
  (* Name is "deep" copied *) (* disposes Name so .str is NIL if no such entry *)

```

```

PROCEDURE GetGadgListIndex(gad:Gadg; Name:DynStr; ColNum:CARDINAL):INTEGER;
  (* -1 means it was not found *)

```

```

PROCEDURE GetGadgListSelect(gad:Gadg; VAR index:INTEGER; VAR ColNum:CARDINAL);
  (* -1 means none selected *)

```

```

PROCEDURE SetGadgListSelect(gad:Gadg; EntryIndex:INTEGER; ColNum:CARDINAL);

```

```

PROCEDURE SetListEntryUser(gad:Gadg; EntryIndex:INTEGER; ColNum:CARDINAL;
  userdata:ADDRESS);

```

```

PROCEDURE GetListEntryUser(gad:Gadg; EntryIndex:INTEGER; ColNum:CARDINAL):ADDRESS;

```

(* NOTE: Currently a double click on a list item will return two gadget hits, the first a SelectEv w/o DoubleClick and the second a DblClickEv with DoubleClick in the status *)

(*
VAR DebugMsg :IntuiMessage;
 DebugCntr :CARDINAL;
*)

(* Specialized usage *)

PROCEDURE DebugMode(On:BOOLEAN);
 (* always draws boxes for SpaceSG and ContainerSG *)

PROCEDURE RevealIDField(gag:Gadg):INTEGER;
PROCEDURE RevealWndoField(gad:Gadg):Wndo;
 (* only the top level container will return non NIL. *)

PROCEDURE RevealNativeScreen(wndo:Wndo; VAR sp:ADDRESS);
PROCEDURE RevealNativeWindow(wndo:Wndo; VAR wp:ADDRESS);
PROCEDURE RevealNativeGadget(gag:Gadg; VAR gp:ADDRESS);

PROCEDURE RevealWndoObj(wndo:Wndo; VAR winObj:Object);

PROCEDURE FindWndo(Nativewp:ADDRESS):Wndo;

PROCEDURE GetGadgCurrentPlacement(gad:Gadg; VAR LeftTopOffset, WidthHeight:TwoD);

(* NOTES

1. For now, need to add gadgets "from the top down". ie, the container it is added to must have a path up to a WindowContainer.
2. Pacer events, when enabled, are only put out if a second or more passes with no other window event. They will come out at the rate of one a second.
3. A RETURN or TAB in a String or Integer box results in a SelectEv for it. All String and Integer Gadgs have TabCycle attribute.

*)

END SimpleGUI.

SimpleGraphics Module

SimpleGraphics is an object-oriented 2D graphics package for AOS4.

It derives from an M2 Windows API implementation that has been (partially so far) transported as a layer above Intuition and Graphics library.

When fully implemented it will still be mostly a plotting package, but should be useful and extendable via inheritance.

Classes in the basic package are:

Screen

A drawing surface, currently implemented by an Amiga Window.

BufferedScreen

An extension of class Screen, useful for animation.

GraphRegion

A rectangular regions within a screen.

GraphText

An extension of class GraphRegion that encloses text.

GraphPlot

An extension of class GraphRegion that includes a scaling REAL number coordinate system and drawing operations.

GraphPlotWithAxes

An extension of class GraphPlot that supports placement of one or more horizontal or vertical axes onto the plot.

GraphPlotIterator

A virtual class that can be used to draw data sets onto a GraphPlot.

Section Head: [Support](#)

Module Obj

This module can be used to implement a true Object Oriented package in base Modula-2, including inheritance, method replacement by the child, with static and dynamic class membership for an object.

In the future, if the ISO Modula-2 "Object Oriented Extensions" are implemented in this compiler, it will take care of all the "manual" tasks required here to do true O-O, and replace the need for this module.

Nevertheless, in case you want to experiment, some instructions for using module Obj are in Obj.def.

(Also see module SimpleGraphics, currently the only module I've written using Obj.)

Section Head: [Aglet_Implementation](#)Next subsection: [Tips](#)

Amiga Specific Info

Program Return Code

The shell return value, \$RC, will be automatically set to 20 or above if the program terminates due to an EXCEPTION (including Assertions.Assert() failure).

Otherwise, you can explicitly write to the variable SystemRTS.CLIReturnCode to set this value.

Program Arguments - Shell and Workbench

ISO module ProgramArgs now has code to process arguments passed from the Workbench startup message as well as from the CLI. The code attempts to pretty much allow the ToolTypes in the WB icon to be the same as the CLI switches. This means that the Aglet module ArgsSupport usually will work transparently whether the program started via the CLI or the Workbench.

There are some rules for how you should set up your program argument templates and ToolTypes for this to work well.

- > the "Project" file(s) on which a "Tool" (program) starts up on, if any, must be the beginning arguments and must not require the CLI arg key (ie, no "/K")
- > ToolTypes are turned into following key or key keyvalue strings as if they had come from the CLI that way. ToolType lines beginning with a "(" are ignored, so that this (informal?) way of documenting what ToolTypes are available without issuing them will still work.
- > Multiple ToolType values separated by "|" may work for /F or /M CLI templates, but only if things are kept simple.
- > all the ToolTypes in the (first) Project icon will be applied, AND all the ToolTypes which are in the Tool (program) file icon but not in the (first) Project icon are also applied.

Hooks

Amiga "Hooks" are used in *Aglet M2 PPC* essentially as described in the Autodocs for C hooks. You should, however, always use the procedure specially provided in the Utility module as the .hEntry field of the Hook RECORD order to ensure that any M2 required extra prolog and postlog processing is done.

For example, if you have written a String Gadget hook procedure:

```
PROCEDURE sgHookProcedure(hook:HookPtr; obj:ADDRESS;
                          msg:ADDRESS):CARDINAL;
```

and allocated a HookPtr "hk", you can set up the Hook as:

```
WITH hk^ DO
  hEntry      := Utility.M2HookStub;
  hSubEntry   := sgHookProcedure;
  hData       := <whatever, usually ptr>;
END;
```

Note that the definition of a type HookFunc in Utility requires you to use ADDRESS as the types of the two parameters. Within sgHookProcedure, you will need to assign "obj" to a typed pointer, "SGWorkPtr", to use it to access the required fields, and, in this case the "msg" parameter is simply to be CAST to the command, a CARDINAL.

Docs for each OS Hook need be consulted to determine what is actually being passed.

Section Head: [Aglet_Implementation](#)Next subsection: [Debugging](#)

Language and Compiler Tips

Report Problems

Tom Breeden
Aglet Software
tmb@virginia.edu

Use M2IDE

If at all possible, for non-trivial projects use the *M2IDE* since it will track module import dependencies and compile your modules in the correct order (use "Update" on the program module after a "Refresh" if necessary).

GoldEd is a very nice commercial editor configurable for syntax coloring for Modula-2. It is still available for sale at Dietmar Eilert's pages on the web.

Turbotext was a commercial editor that has now been made freely available. Turbotext V2 runs well on the latest release of AOS4. For download, see the URL <http://www.monkeyhouse.eclipse.co.uk/amiga/turbotext/>.

An editor interface is also provided for *CygnusEd* and for the nice freeware editor, *Annotate*, available on OS4Depot.

Use the ISO Standard Library

There are a number of ISO compatible or ISO-like M2 compilers for other platforms, eg. GNU Modula-2.

Starting your program via a WB Icon

Programs can be started from a shell with Amiga style arguments, or from the WB with equivalent Tooltypes.

M2IDE: Placement of program source

Currently upon new project creation *M2IDE* expects that the program source file will be in the current directory. You can get around this by adding your desired directory to the Source Files name list immediately after starting up *M2IDE*, saving the project, and then restarting *M2IDE* on the project, upon which the existing program (or program skeleton) file will be found. This will be stored in the .PRJ file so needs be done only at the start of a new project.

M2IDE: Watch out for source file errors in the IMPORT section

M2IDE parses the import section on all project files it can find, and uses that information to generate the dependency tree. If it does not seem to have identified all the necessary modules, or does not seem to be able to open a dependent module even though that module's path is in the Sources NameList, it is likely due to syntax errors in the IMPORT section. (*M2IDE* now gives a pop-up warning in these cases).

Asm Files

The compiler writes out the temporary `#?.asm` files (as well as `#?.err` files) to "T:". It does not delete these files, expecting that T: is assigned to RAM: or RAD:. If that is not the case on your system, you may want to clear out T: from time to time.

Object files (`#?.o` files) for implementation modules and Symbol files (`#?.sbm`) for definition modules are written to the output directory (current directory) and should be kept around to avoid the necessity of re-compiling support modules.

Debugging Tips

Currently there is no source level debugger for *Aglet M2 PPC*, but there are a number of ways to get debugging information (in addition to the tried and true one of inserting write statements in the code using the standard modules `STextIO` and `SWholeIO`).

Also, note that the compiler will identify and generate no code for source statements that are clearly unreachable. To be able to eliminate easily almost all of the run-time overhead of a debugging statement, declare a constant at the top of your module such as

```
CONST DEBUG = TRUE;
```

and surround your debugging statements with **IF DEBUG THEN ... END**. Simply changing the constant to `FALSE` and re-compiling will remove the debugging code from the output object file.

Note that you can also turn such debugging statements on/off on a per procedure basis simply by including the `CONST` assignment locally within the procedure itself. At compile-time the value will revert the global one outside of that procedure.

Other debugging possibilities include:

Module SymbolsRTS
Modules Debugging and Assertions
Module DebugIO
Verbose Compile and Link
Code Listing Compile
Ranger
Grim Reaper Info
GDB

[SymbolsRTSDebug](#)
[ModDebug](#)
[ModDebugIO](#)
[VerboseCompile](#)
[CodeListing](#)
[Ranger](#)
[GrimReaper](#)
[GDB](#)

SymbolsRTS

If you put an "IMPORT SymbolsRTS;" into your Program module, this enables the default error-trapping routines to report the error location in terms of the module and procedure name from the source code.

ie, you will get a message on exception like this:

```
-----  
| M2EXCEPTION |  
|-----|  
| wholeDivException |  
| ThisModule.ThisProc |  
|-----|  
|-----|
```

rather than this:

```
-----  
| M2EXCEPTION |  
|-----|  
| wholeDivException |  
| 0x7fc0df7c |  
|-----|  
|-----|
```

To get the best results, you should also use the "-incldbg" ("-g") switch with Mod2Lnk, which includes non-exported procedure names into the executable.

In addition, your own exception handlers may be able to make use of the procedure SymbolsRTS.FindSourcePos() to report in the same way on any address within the code. (The LineNum return value is not yet implemented).

Section Head: [Debugging](#)
Next subsection: [ModDebugIO](#)

DEFINITION MODULE Debugging;

This debugging module can pause and show some information or write it into a debugging window.

PROCEDURE Debug(msg:**ARRAY OF CHAR**; i:**INTEGER**);
PROCEDURE DebugPause(msg:**ARRAY OF CHAR**; i:**INTEGER**);

PROCEDURE DebugOn(DebugLine:**CARDINAL**);
PROCEDURE DebugOff;
PROCEDURE DebugQuery(msg:**ARRAY OF CHAR**);

PROCEDURE IsDebugOn():**BOOLEAN**;

PROCEDURE DebugMemory(); (* NotYetImplemented *)
PROCEDURE DebugStack();

DEFINITION MODULE Assertions;

PROCEDURE Assert(b:**BOOLEAN**; msg:**ARRAY OF CHAR**);
(* EXCEPTION if b is not true *)

PROCEDURE AssertWarn(b:**BOOLEAN**; msg:**ARRAY OF CHAR**);
(* Warns if b is not true, then continues *)

PROCEDURE IsAssertException():**BOOLEAN**;

Section Head: [Debugging](#)Next subsection: [VerboseCompile](#)**DEFINITION MODULE DebugIO;**

Use this debugging module when you want to minimize the dependency on other working M2 code. It is simply a call to the AOS4 library procedure TimedDosRequester() with a 60 second timeout.

PROCEDURE DebugIOMsg(Title, Text:**ARRAY OF CHAR**; Buttons:**ARRAY OF CHAR**):**INTEGER**;

(* Use "|" to separate multiple buttons *)

(* returns 1, 2, 3, ..., N, 0 indicate which choice, left to right, was taken *)

(* NOTE: > Leftmost choice always 1, rightmost always 0. *)

Section Head: [Debugging](#)
 Next subsection: [CodeListing](#)

Verbose Compile Listing

Using the "-verbose" compiler switch will display the name of each imported symbol file as it is read, and, more importantly, will display each procedure name being processed as well as the phase of processing it is in.

If, in the unfortunate (and hopefully, increasingly unlikely) case that the compiler runs into an error from which it cannot recover, you can still discover in which procedure the problem occurred, as well as the compiler operation (parsing, optimizing, code generation, or file output).

Here is an example of the output for a verbose compile:

Mod2 v3.0 Compiler Alpha0 (3.4.2008)
Copyright (c) 2004 by Thomas Breeden
GuideMaker.mod

```
<- M2Lv3:SystemRTS.SBM
<- M2Lv3:Assertions.SBM
... <etc>
<- AmigaGuideCmdStructs.SBM
<- AmigaGuideFormat.SBM
<- AmigaGuideParser.SBM
<- AmigaGuideSyntax.SBM
<- AmigaGuideSyntax.SBM
<- GuideMakerDefs.SBM
<- GuideMakerGUI.SBM
<- GuideMakerInput.SBM
<- GuideMakerNodeProcess.SBM
<- GuideMakerShow.SBM
... <etc>
<- M2Lv3:Str1.SBM
```

```
Delay Parse-Optim-Gen-Out
InitPgm Parse-Optim-Gen-Out
GetArgs Parse-Optim-Gen-Out
GetFile Parse-Optim-Gen-Out
NodeIndexArg Parse-Optim-Gen-Out
CmdIndexArg Parse-Optim-Gen-Out
NewNodeArgs Parse-Optim-Gen-Out
ShowNodeArgs Parse-Optim-Gen-Out
EditNodeArgs Parse-Optim-Gen-Out
NewLinkArgs Parse-Optim-Gen-Out
NewAttrArgs Parse-Optim-Gen-Out
UpdCmdArgs Parse-Optim-Gen-Out
NewCmdArgs Parse-Optim-Gen-Out
```

FileNameArgs Parse-Optim-Gen-Out
WriteItOut Parse-Optim-Gen-Out
ProcessLoop Parse-Optim-Gen-Out
GuideMaker Parse-Optim-Gen-Out
-> GuideMaker.asm

Optimize Setting: DeadCode
SDK:gcc/bin/as -o GuideMaker.o GuideMaker.asm 0

.

Section Head: [Debugging](#)
 Next subsection: [GrimReaper](#)

Compiler Code Listing

Using the "-list" compiler switch will produce a detailed listing including the source lines, the intermediate code, and the PPC Native code.

For example:

```

128
129 (*-----*)
130 PROCEDURE Delay(secs, millis:CARDINAL);
131 (*-----*)
132
                                |-----Delay Entry-----
                                procentry Delay
                                ;DetermineFrameSize
                                ;save LR in caller's frame
                                mfspr %r0,8
                                stw %r0,4(%r1)
                                ;alloc stack frame and save regs (STATIC)
                                or %r11,%r1,%r1
                                stwu %r1,-64(%r1)
                                stw %r29,-12(%r11)
                                stw %r30,-8(%r11)
                                stw %r31,-4(%r11)
                                ;establish the mp linkage
                                stw %r14,12(%r1)
                                addi %r14,%r1,12
                                ;Move the overflow params
                                ;Move the HIGH values
                                ;DoStructureParmCopies
                                ;Copy the static refcpyparams
                                ;Copy the dynamic params
                                ;ExitBasicBlockPre
                                stw %r3,8(%r14)
                                stw %r4,12(%r14)
                                ;ExitBasicBlockPost
                                |-----Delay BlockCode-----
133 VAR  t      :AmigaTimer.TimerHandle;
134      bRes   :BOOLEAN;
135
136 BEGIN
137
138 bRes := AmigaTimer.OpenATimer(t, 1);
                                prepcall OpenATimer
                                paramref t OpenATimer $c0(0H)
                                addi %r3,%r14,16
                                paramcpy $c1(1H) OpenATimer $c1(1H)
                                addi %r4,0,1
                                ;ExitBasicBlockPre
                                callext OpenATimer -> $t1
                                bl 0
                                or %r31,%r3,%r3
                                ;ExitBasicBlockPost
                                copy $t1 -> bRes
                                or %r30,%r31,%r31

```

```

139 bRes := AmigaTimer.SleepATimer(t, secs, 1000*millis);
    prepcall SleepATimer
    mul $c1000(3E8H) millis -> $t2
    lwz %r29,12(%r14)
    mulli %r31,%r29,1000
    paramcpy t SleepATimer $c0(0H)
    lwz %r3,16(%r14)
    paramcpy secs SleepATimer $c1(1H)
    lwz %r4,8(%r14)
    paramcpy $t2 SleepATimer $c2(2H)
    or %r5,%r31,%r31
;ExitBasicBlockPre
    stb %r30,20(%r14)
    callext SleepATimer -> $t3
    bl 0
    or %r31,%r3,%r3
;ExitBasicBlockPost
    copy $t3 -> bRes
    or %r30,%r31,%r31
140 bRes := AmigaTimer.CloseATimer(t);
    prepcall CloseATimer
    paramref t CloseATimer $c0(0H)
    addi %r3,%r14,16
;ExitBasicBlockPre
    stb %r30,20(%r14)
    callext CloseATimer -> $t4
    bl 0
    or %r31,%r3,%r3
;ExitBasicBlockPost
    copy $t4 -> bRes
    or %r30,%r31,%r31
141
142 END Delay;

;ExitBasicBlockPre
    stb %r30,20(%r14)
;ExitBasicBlockPost
|-----Delay Exit-----
procexit Delay
    lwz %r14,0(%r14)
    addi %r11,%r1,64
    lwz %r29,-12(%r11)
    lwz %r30,-8(%r11)
    lwz %r31,-4(%r11)
    lwz %r1,0(%r1)
    lwz %r0,4(%r1)
    mtspr 8,%r0
    bclr 20,0
;ExitBasicBlockPre
;ExitBasicBlockPost

```


Section Head: [Debugging](#)
Next subsection: [GrimReaper](#)

Ranger

Every OS4 programmer should have a copy of the System diagnostic tool, "Ranger", by Steven Solie.

See OS4 Depot at:

<http://os4depot.net/index.php?function=showfile&file=utility/misc/ranger.lha>

Grim Reaper

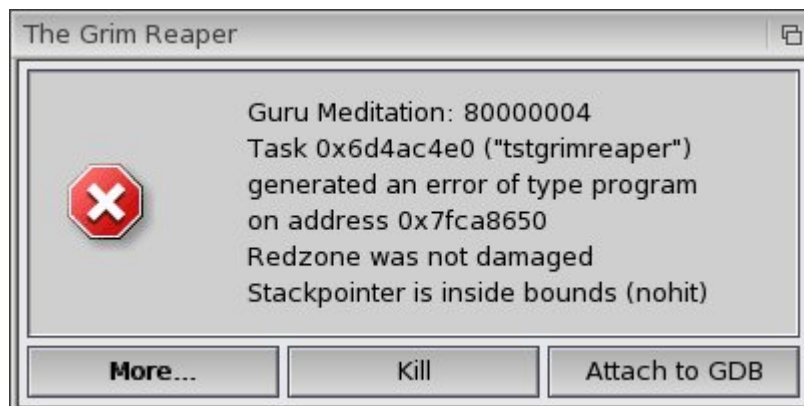
In some (desperate) situations inserting a "breakpoint" into the source code, along with knowledge of [LowLevel](#) compiler information and a bit of PPC Assembly language, can throw light on an obscure malfunction.

The inline assembly language feature of *Aglet M2 PPC* responds to the (non-standard) PPC instruction, "illegal", by breaking into the Grim Reaper at exactly that point in the code sequence.

for example the program:

```
(*#####*)
MODULE TstGrimReaper;
(*#####*)
VAR i, j:INTEGER;
  (*-----*)
  PROCEDURE P(j:INTEGER; VAR i:INTEGER);
  (*-----*)
  BEGIN
    <*Asm(ON)*>
    illegal
    <*Asm(OFF)*>
    i := j+17;
  END P;
BEGIN
i := 11;
j := 13;
P(j, i);
END TstGrimReaper.
```

Will produce this upon reaching the "illegal":



The screenshot shows the 'The Grim Reaper' application window with the 'Registers' and 'Crash Location' tabs selected. The 'Registers' tab displays a table of registers and their values, with r3 highlighted. The 'Crash Location' tab shows details about the crash, including the instruction pointer and crashed task.

Register	Value
r0	0x7FCA86D4
r1	0x6BA63ED0
r2	0x016B7504
r3	0x0000000D
r4	0x6CD4FF90
r5	0x7FCA7094
r6	0x02000000
r7	0x00000000
r8	0x00000008

Crash Location

Instruction Pointer	0x7FCA8650
Crashed Task	tstgrimreaper
Crash Type	program
Function Name	<unknown>
Function Offset	<unknown>
Binary Type	<unknown>
Section	<unknown>
Module	tstgrimreaper
Alert	80000004

Buttons at the bottom: Kill Program, Reboot, Continue Program, Attach to GDB, Ignore DSI errors, Write Crash Log.

The screenshot shows the 'The Grim Reaper' application window with the 'Memory Dump' tab selected. It displays a memory dump with addresses, hex values, and ASCII characters. The address 6cd4ff90 is highlighted, showing a value of 0000000d.

```

6cd4ff20  5f5f5f5f 5f5f5f5f 5f5f5f5f 5f5f5f5f "          "
6cd4ff30  5f5f5f5f 5f5f5f5f 5f5f5f5f 5f5f5f5f "          "
6cd4ff40  5f5f5f5f 5f5f5f5f 5f5f5f5f 5f5f5f5f "          "
6cd4ff50  5f5f5f5f 5f5f5f5f 5f5f5f5f 5f5f5f5f "          "
6cd4ff60  5f5f5f5f 5f5f5f5f 5f5f5f5f 5f5f5f5f "          "
6cd4ff70  5f5f5f5f 5f5f5f5f 5f5f5f5f 5f5f5f5f "          "
6cd4ff80  5f5f5f5f 5f5f5f5f 5f5f5f5f 5f5f0000 "          "
6cd4ff90  0000000d 0000000d 5f5f0000 00000000 "          "
6cd4ffa0  00000000 00000000 00000000 00000000 "          "
6cd4ffb0  00000000 00000000 00000000 00000000 "          "
6cd4ffc0  00000000 00000000 00000000 00000000 "          "
6cd4ffd0  4e756100 f0f66000 00024e5d 285f4e75 "Nua. ðö`...N] (_Nu"
6cd4ffe0  deded4cb 3a41676c 65742f4d 322d7633 "FP0B: Aglet/M2-v3"
6cd4fff0  2f436f6d 70696c65 722f7465 737400ec "/Compiler/test.i"
6cd50000  5359532d 4448353a 53797374 656d2f47 "SYS-DH5: System/G"
6cd50010  72696d52 65617065 72000834 00520057 "rimReaper..4.R.W"
    
```

Buttons at the bottom: Kill Program, Reboot, Continue Program, Attach to GDB, Ignore DSI errors, Write Crash Log.

If r3 does not contain 13, or the mem location pointed to by r4 does not contain 11 at that point, you know that something has gone seriously wrong before then.

You can "Continue Program" after this "breakpoint" with no ill-effects.

GDB

A determined and patient programmer with a good knowledge of assembly code may be able to make some use of GDB.

Even without debugging symbols, the Elf executable will have a symbol table that includes the addresses of the entry points of all modules and procedures. You can use the GDB command "info functions" to see these addresses, and then insert breakpoints as desired.

Nevertheless, I find GDB extremely difficult to use, but have been able to use it to tease out information I needed about what was happening at program startup, before the modules DebugIO or Debugging are usable.

.

Prev section: [Aglet_Implementation](#)

Next section: [IDE_Tutorial](#)

Some Example and Test Programs

Classic

HelloWorld

Intuition etc.

ReadFontPrefs

-Lists the five user-settable Prefs fonts (WB Icon, Drawer Icon, Drawer Text, System Default, and Screen Font)

ReadPubScreens

-Lists the names of all the Public Screens the user has set up with Prefs.

ScanIFF

-List outline of the IFF file or clipped material.

TstAutoRequest

-Calls AutoRequest and EasyRequest.

TstDataTypes

-Classifies a given file via DataTypes, and displays picture datatype files into a window.

Reaction

CheckBoxExample

-Reaction example translated from the SDK.

DateBrowserExample

- <ditto>

FuelGaugeExample

- <ditto>

GetColorExample

- <ditto>

GlyphExample

- <ditto>

Compiler

TstZeroDivide

-Shows exception handling of a divide by zero exception.

TstRandomNumbers

-Generates one set of ten random numbers from 0 to 99, and another from 0 to 9999.

Aglet Support

TstArgsSupport

-Parses its argument. Template is "-file/A"

TstDirUtil2

-Tests GetFileSize() function from module DirUtil2.

TstDirUtil3

-Tests AbsoluteFileSpec() function from module DirUtil3.

TstRandInts

-Tests RandInts module distribution of x objects into y cells.

Aglet Experimental

TstSimpleGUI

-Demo of some gadget windows created with module SimpleGUI.

TstRandomBoxes

-Generates a display of a window with x boxes randomly placed and colored within it.

TstGetFile

-Shows module SimpleRequesters access to the GetFile gadget.

TstSimpleGraphics

-Demo of 2D drawing package layered on Amiga GraphicsLib drawing routines.

TstSimpleImageHandler

-Shows images and glyphs supported currently in this module.

TstSimpleMenus

-Puts up a window with NewMenus menus created by this module.

Prev section: [M2_Examples](#)

Next section: [M2_IDE](#)

IDE Tutorial

This tutorial takes you through the steps in a coding task: adding a missing feature in a working program (in this case *M2IDE* itself). It assumes that the project has already been created and used to build one or more earlier versions of the program. (see [IDE_Using](#) below for setting up a new project)

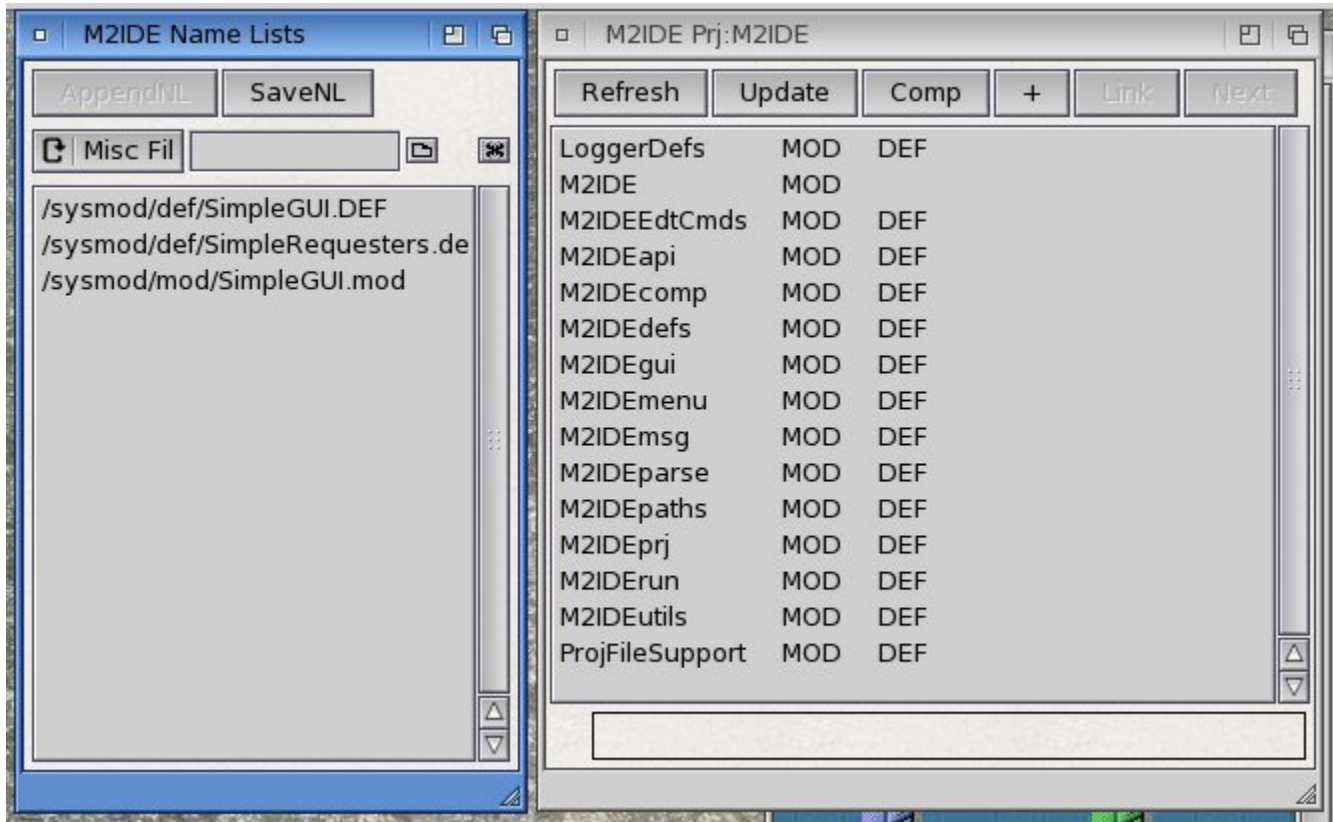
The problem fixed in this tutorial is that *M2IDE* has three windows, the Proj Window, the NameList Window, and the BuildPrefs Window, but on right-press the program menu did not appear when the BuildPrefs Window is active.

Evidently, I had neglected to attach the menustrip to that window. In this tutorial, we will correct that using the IDE environment.

Start from the CLI with this command:

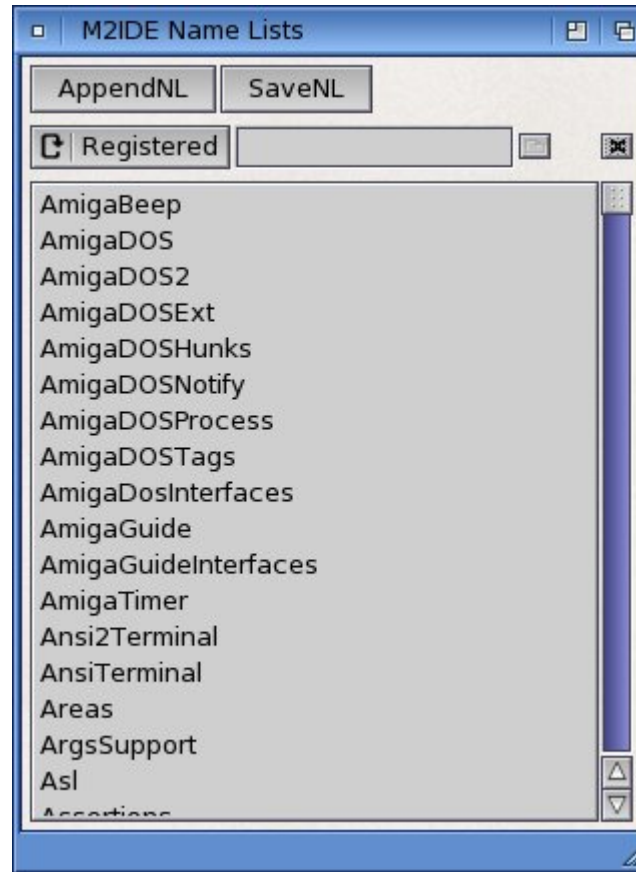
```
4.Work:Aglet/M2-v3/Compiler/wrk> run m2ide m2ide
```

A file "M2IDE.prj" does exist in this directory, so it is opened and read, displaying the two windows below (as well as the logging window).



The Proj window on the right shows the project specific files. The NameList window on the left starts off showing the Miscellaneous file list, into which in this case I had put two "library" support Definition module sources, so I can quickly refer them them while programming.

Using the cycle gadget in the NameList window, we can see the list of "registered" modules which were read in from the *.prj file. These are essentially all the Amiga/Reaction/ISO/Aglet support modules that we assume to be a fixed context and do not want or need to edit or re-compile while we are working at this project.



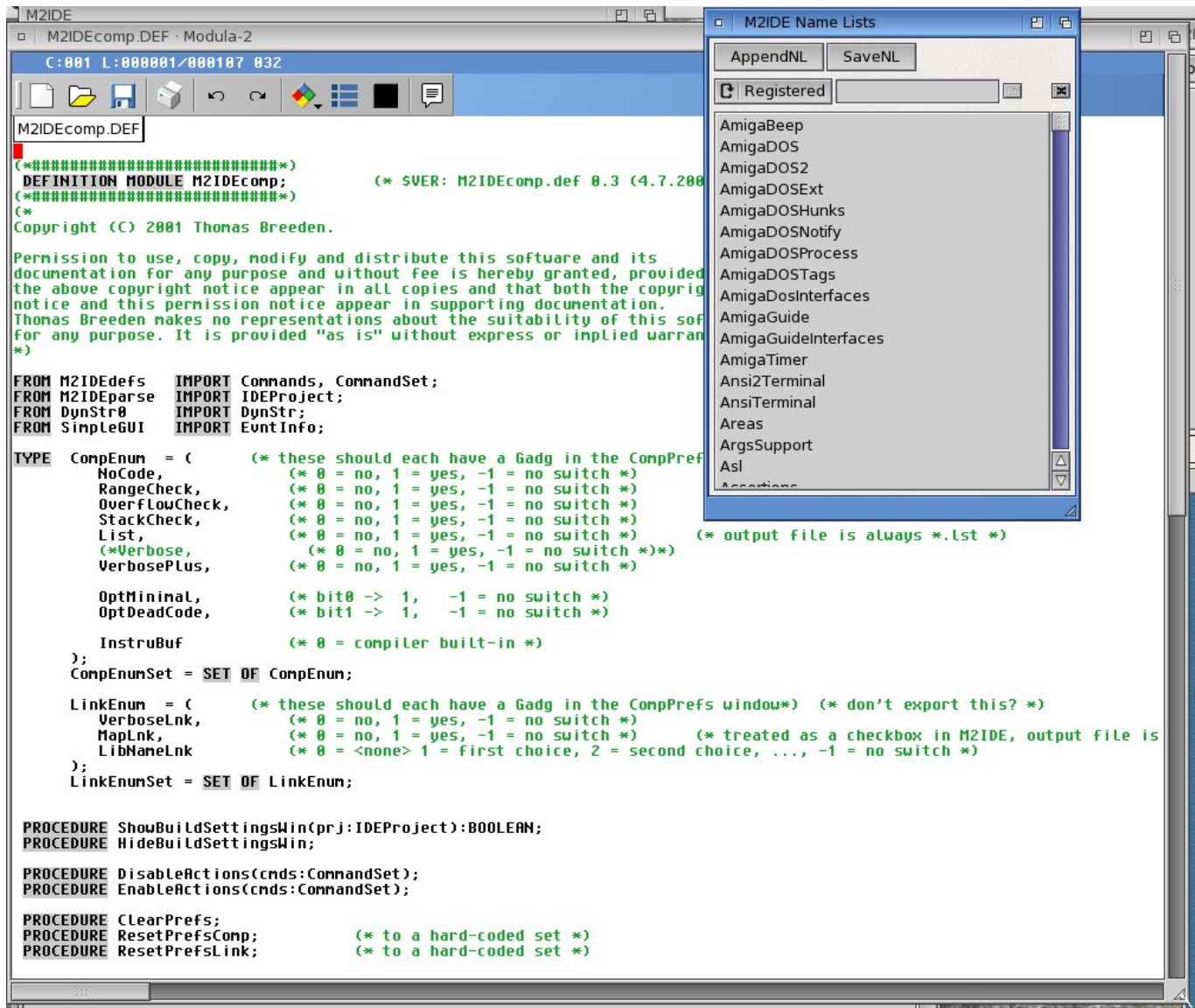
Note that, at this point, the Proj window shows all its modules as up-to-date.

Deciding where to start

In this case the first programming change stems from the fact that the BuildPrefs window display is handled entirely within module M2IDEcomp, and that it uses module SimpleGUI, which monitors window, gadget, and menu events for it. The program commands are generated within M2IDEcomp, but passed onto module M2IDEmsg for processing.

We need to add a way to also pass on menu events from M2IDEcomp. Thus we will have to change a procedure declaration in the Definition module to include an menu support in the window.

So, a double click on the DEF column of the M2IDEcomp row in the Proj window gets that file opened into the editor.



IDE_Tut1

IDE Tutorial, continuedThe Definition Module Edit

The procedure interface to be changed is that of GetPrefsWndoCmd(). An extra parameter must be added to return the menu EventInfo.

ie, this:

```

| PROCEDURE GetPrefsWndoCmd(VAR cmd:Commands;
| VAR args:DynStr):BOOLEAN;
| (* args must exist *)
|

```

is edited to this:

```

| PROCEDURE GetPrefsWndoCmd(VAR cmd:Commands; VAR args:DynStr;
| VAR ev:EvtInfo):BOOLEAN;
| (* args must exist *)
| (* if returns TRUE and NullCmd, ev will carry a Menu event *)
|

```

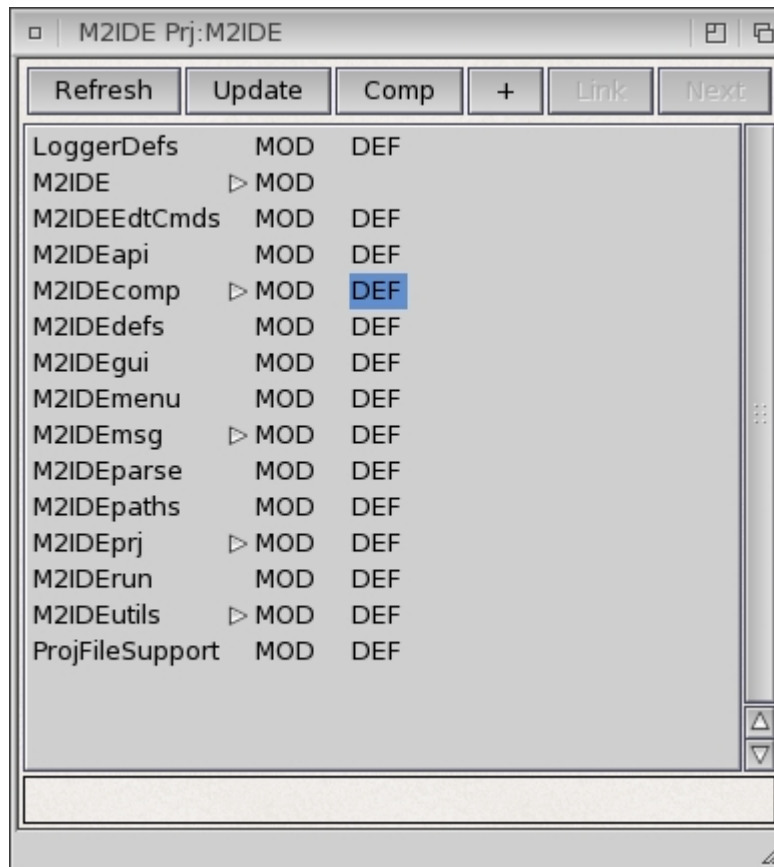
Now with the DEF column selected in the M2IDEcomp row, hitting the COMP button in the Proj Window compiles the desired Definition file, giving this text in the logging window:

```

| Aglet Mod2
| PROGDIR:Mod2 -rngchk -ovflchk -instrubuf 2500 -optlev 2
| M2IDEcomp.DEF
|
| Mod2 v3.0 Compiler Alpha0 (29.5.2008)
| Copyright (c) 2004 by Thomas Breeden
| M2IDEcomp.DEF
| <- M2Lv3:SimpleGUI.SBM
|
| -> M2IDEcomp.SBM
|

```

M2IDE now has updated the Project Window to show that the modules dependent on *M2IDEcomp.def* need to be recompiled.



IDE Tutorial, continued

Attaching the Menu

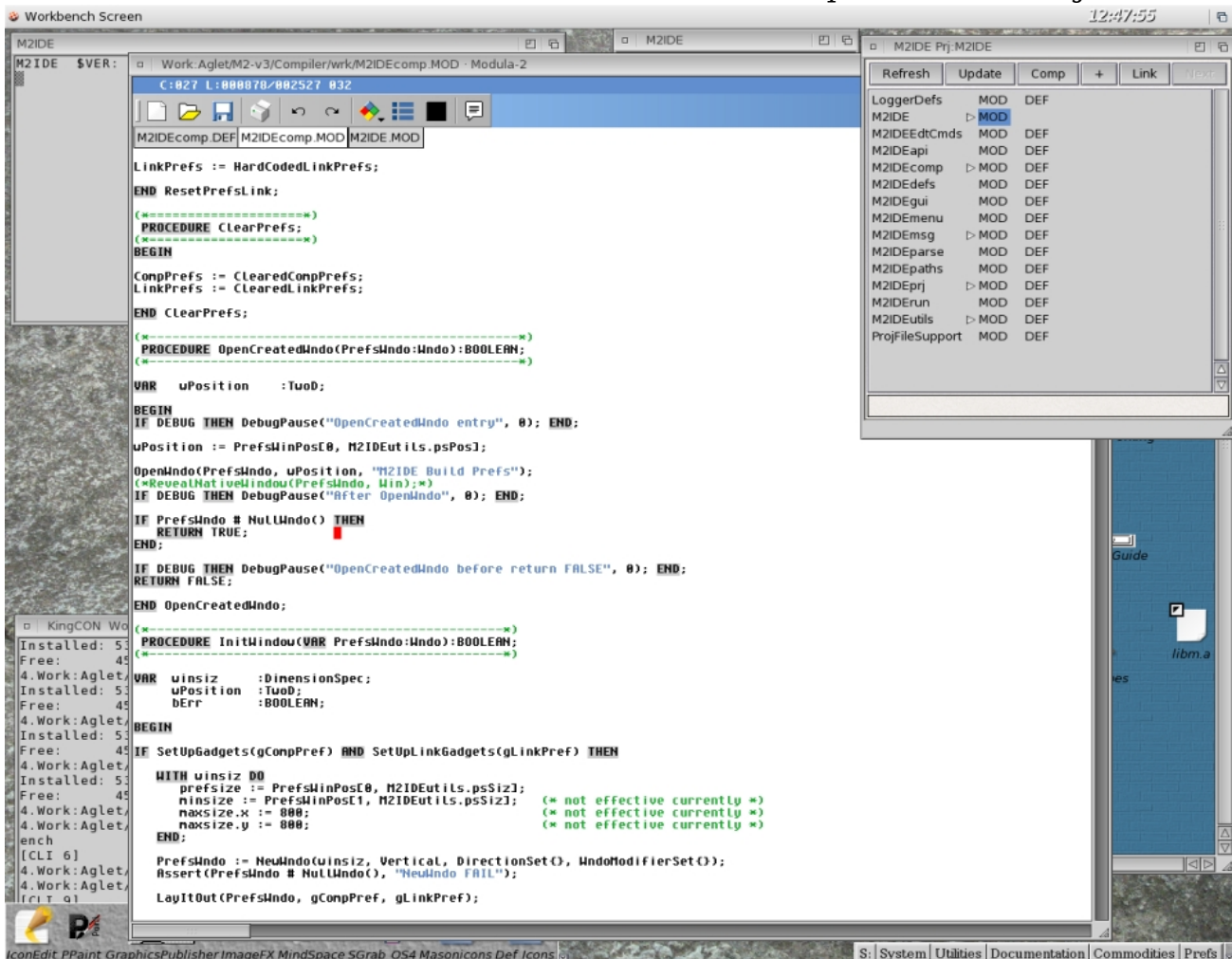
File M2IDEcomp's window handling implementation code must be edited so that the program menu appears when the BuildPrefs window is active.

This is easy to do using these two procedure already existing in module M2IDEmenus :

```
PROCEDURE ShareMenu(AccessoryWndo:Wndo);
PROCEDURE UnshareMenu(AccessoryWndo:Wndo);
```

The main program window, handled by module M2IDEgui, creates and owns the menu structure, so all we have to do in M2IDEcomp is make sure it is attached to the BuildPrefs window when it is opened, and removed before it is closed.

Double click on the MOD column of the M2IDEcomp row in the Proj window to get



that file opened into the editor.

There is one place where the window is opened, that we just edit to this:

```

OpenWndo(PrefsWndo, wPosition, "M2IDE Build Prefs");
IF PrefsWndo # NullWndo() THEN
  ShareMenu(PrefsWndo);
  RETURN TRUE;
...

```

The unshare call needs to go before the CloseWndo() in the exported procedure M2IDEcomp.HideBuildSettingsWin().

```

(*=====*)
PROCEDURE HideBuildSettingsWin;
(*=====*)

BEGIN

IF IsWndoOpen(PrefsWndo) THEN
  UnshareMenu(PrefsWndo);
  CloseWndo(PrefsWndo);
END;

END HideBuildSettingsWin;

```

Another place to put the unshare call is into the termination code of the module M2IDEcomp, in case the program closes down while the BuildPrefs window is open.

```

FINALLY
...
IF PrefsWndo # NullWndo() THEN
  IF IsWndoOpen(PrefsWndo) THEN
    UnshareMenu(PrefsWndo);
  END;
  DisposeWndo(PrefsWndo);
...

```

IDE Tutorial, continued

Handling the Menu Event

In M2IDEcomp.GetPrefsWndoCmd(), events that SimpleGUI has been accumulating for the BuildPrefs window are being unqueued. The update needed here is simply to check for and return the EventInfo parameter in the case of a MenuEv.

Just add a check into the WHILE loop:

```
WHILE GetWndoEvnt(PrefsWndo, ev) DO  
  
  ...  
  
  ELSIF ev.ev = MenuEv THEN  
  
    cmd := NullCmd;    (* but return TRUE *)
```

GetPrefsWndoCmd() is called by a procedure that acts as the main input loop for the program, M2IDEmsg.GetCmd(), defined as:.

```
PROCEDURE GetCmd(wndo:Wndo; VAR cmd:Commands; VAR  
args:DynStr);
```

So, in that procedure within M2IDEmsg, we change the call to GetPrefsWndoCmd() to receive the EventInfo parameter and use a procedure, HandleMenuEvent, to change it into a program command (as is done with the other two windows):

```
IF GetPrefsWndoCmd(cmd, args, ev) THEN  
  IF (cmd = NullCmd) AND (ev.ev = MenuEv) THEN  
    HandleMenuEvent(ev, done, cmd, args);  
  ELSE  
    done := cmd # NullCmd;  
  END;  
  
  ...
```

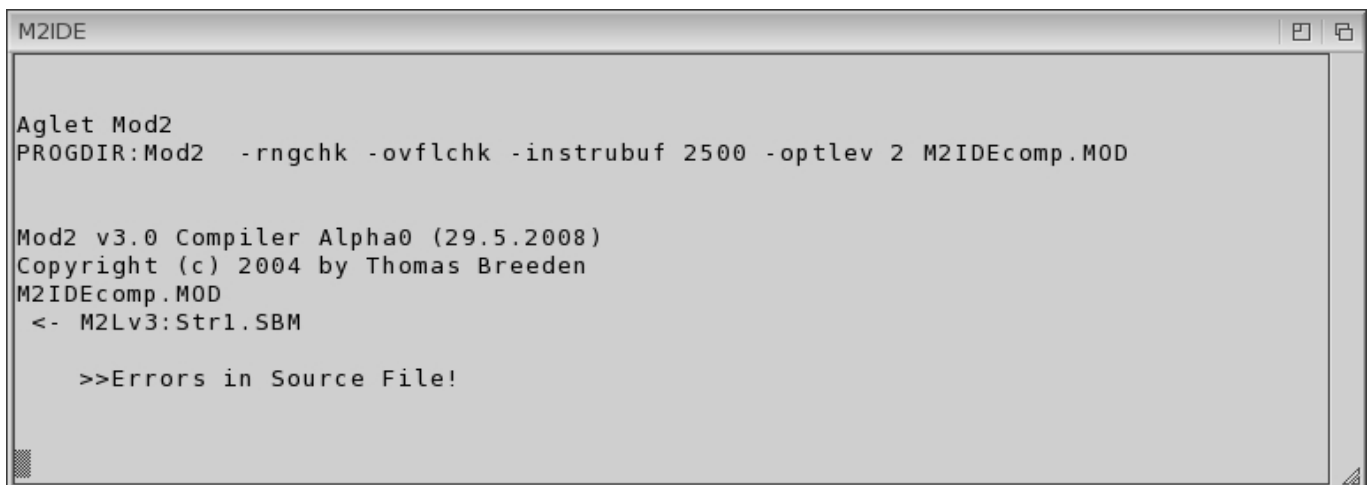
IDE Tutorial, continued

Rebuilding the Program

The above should be the necessary edits.

Now, in the Proj window, selecting the MOD column in the row with M2IDE (the program module), and clicking on the UPDATE button does all the necessary recompiles.

The IDE compiles all the pending to-be-compiled Implementation modules successfully except for M2IDEcomp.mod. The Logger window shows an error was found,

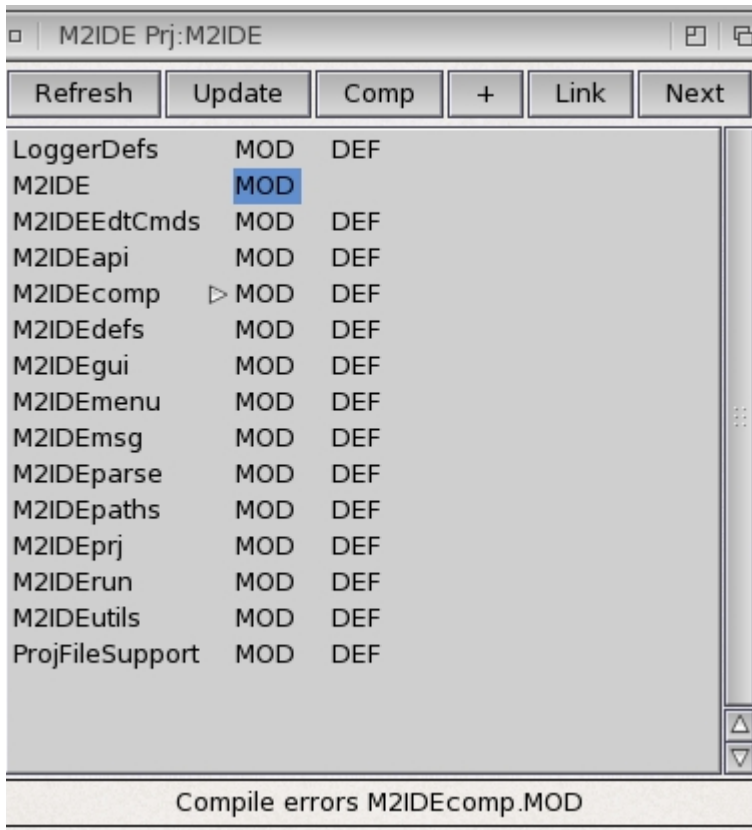
A screenshot of a terminal window titled "M2IDE". The window contains the following text:

```
Aglet Mod2
PROGDIR:Mod2 -rngchk -ovflchk -instrubuf 2500 -optlev 2 M2IDEcomp.MOD

Mod2 v3.0 Compiler Alpha0 (29.5.2008)
Copyright (c) 2004 by Thomas Breeden
M2IDEcomp.MOD
<- M2Lv3:Str1.SBM

    >>Errors in Source File!
```

and the project window shows M2IDEcomp.mod still requires compiling.



The IDE has sent a message to the editor and checking the editor window for M2IDEcomp.mod we see that it is positioned at the point of the compile error, with the M2 error message in the window title bar.

```

M2IDEcomp.MOD · Modula-2
C:023 L:000878/002528 041 identifier not declared or not visible
M2IDEcomp.MOD
LinkPrefs := HardCodedLinkPrefs;
END ResetPrefsLink;
(*=====*)
PROCEDURE ClearPrefs;
(*=====*)
BEGIN
CompPrefs := ClearedCompPrefs;
LinkPrefs := ClearedLinkPrefs;
END ClearPrefs;
(*-----*)
PROCEDURE OpenCreatedWndo(PrefsWndo:Wndo):BOOLEAN;
(*-----*)
VAR   wPosition   :TwoD;
BEGIN
IF DEBUG THEN DebugPause("OpenCreatedWndo entry", 0); END;
wPosition := PrefsWinPos[0, M2IDEutils.psPos];
OpenWndo(PrefsWndo, wPosition, "M2IDE Build Prefs");
(*RevealNativeWindow(PrefsWndo, Win);*)
IF DEBUG THEN DebugPause("After OpenWndo", 0); END;
IF PrefsWndo # NullWndo() THEN
  ShareMenu(PrefsWndo);
  RETURN TRUE;
END;
IF DEBUG THEN DebugPause("OpenCreatedWndo before return FALSE", 0); END;
RETURN FALSE;

```

Oops. The error is clear: a typo of "ProfsWndo" instead of "PrefsWindo". The NEXT button in the Proj windows does not find another error.

The BuildPrefs have been set up previously, and were read in with the *.prj file, so we should be able to complete the build immediately. Since there is only one module to be compiled, we can select M2IDEcomp MOD and click on the Proj window "+" button, which causes the IDE to first compile the selected Implementation module and then run *Mod2Lnk* on the project. This is successful; the logger window shows a successful link.

```

M2IDE
-----
Aglet Mod2
PROGDIR:Mod2 -rngchk -ovflchk -instrubuf 2500 -optlev 2 M2IDEcomp.MOD

Mod2 v3.0 Compiler Alpha0 (29.5.2008)
Copyright (c) 2004 by Thomas Breeden
M2IDEcomp.MOD
  <- M2Lv3:Str1.SBM

  -> M2IDEcomp.asm

Optimize Setting: DeadCode
as -o M2IDEcomp.o M2IDEcomp.asm 0

Aglet Mod2Lnk
PROGDIR:Mod2Lnk -libname SDK:clib2/lib/libm.a M2IDE

Mod2Lnk Amiga 0.2 (30.5.2008) OS 4.0 Copyright (c) 2004 Tom Breeden

M2IDE
Warning MutualImport: IOLink IOChan
Warning MutualImport: SimpleGUIHidden SimpleGUI
Warning MutualImport: SimpleGUISupport SimpleGUI
Warning MutualImport: M2IDEutils M2IDERun M2IDEcomp
Warning MutualImport: M2IDEgui M2IDEutils
Warning MutualImport: M2IDEprj M2IDEgui M2IDEutils M2IDERun M2IDEcomp
Warning MutualImport: M2IDEpaths M2IDEprj M2IDEgui M2IDEutils
Warning MutualImport: M2IDEpaths M2IDEprj
Warning MutualImport: M2IDEprj M2IDEgui M2IDEutils
Warning MutualImport: M2IDEmsg M2IDEapi M2IDEprj M2IDEgui M2IDEutils M2IDERun M2IDEcomp
Warning MutualImport: M2IDEmsg M2IDEapi M2IDEprj M2IDEgui
Warning MutualImport: M2IDEmsg M2IDEapi M2IDEprj M2IDEgui M2IDEutils
Warning MutualImport: M2IDEmsg M2IDEapi M2IDEprj
Warning MutualImport: M2IDEapi M2IDEprj M2IDEgui M2IDEutils M2IDERun
Warning MutualImport: M2IDEapi M2IDEprj M2IDEgui M2IDEutils
Warning MutualImport: M2IDEprj M2IDEgui
Warning MutualImport: M2IDEutils M2IDERun
as -o M2IDE_start.o M2IDE_start.asm
ld -o M2IDE T:M2IDE.lnk -q -nostdlib -x
Mod2Lnk done

```

Evidently, the design of the inter-module calls of *M2IDE* could use some attention, but these warnings have been investigated and the calls determined not to be a problem.

Now if we quit the running *M2IDE* and restart it, the problem is fixed: when the BuildPrefs window is open and active, the program Menu still appears and is effective!

Prev section: [IDE_Tutorial](#)
Next section: [Version_History](#)

Integrated Development Environment

M2IDE	IDE
Using M2IDE	IDE_Using
Project Window	Project Win
Name List Window	Name List Win
Build Prefs Window	BuildPrefs Win
Logger Window	Logger Win
Text Search Window	Text Search Win
Project Menu	Project Menu
Settings Menu	Settings Menu
Tools Menu	Tools Menu
Notes	IDE_Other

.

M2IDE

M2IDE is the first step toward an Integrated Development Environment for *Aglet Modula-2*. Though it currently lacks a lot of what one would want in an IDE, I think it still can make developing with Modula-2 easier and faster.

Requirements

M2IDE makes use of ARexx for communicating with the selected editor. REXXMaster must have been started and be working well.

The logger window utilizes the PIPE device. That device should be in your SYS:Devs/DOSDrivers directory.

Features

- > Analyzes all the directly and indirectly imported project modules for a program and determines the appropriate compile order for definition and implementation modules.
- > Allows standard and other "library" modules to be excluded, as you normally don't want to change or re-compile these.
- > Displays the set of project modules in a "point-and-click" window.
- > Has "Compile" (and "Link") buttons to act on the selected module via a button click.
- > Communicates with an editor (currently GoldEd, Annotate, CygnusEd, or TurboText) to automatically save any of the concerned files before the compile and to move the editor's cursor to the error position of a compile error.
- > Has an "Update" button to do a build on the selected module, ie automatically compile all (and only) the out of date modules and their dependencies in the correct order to create the ready to be linked set of object modules.
- > Provides a simple GUI to most of the compiler and pre-linker switches.
- > Global text search all project files.
- > Menu item to open M2IDE.guide
- > Help hint bubbles

Using IDE

Starting

The program template is: "FILE,-EDITOR/K,-D=-d/S".

"FILE", is the name of the the project (.PRJ) file. If no <File>.prj is found, a basic project will be created in the current directory with the program module assumed to be named <File>.mod and located as described below.

"-EDITOR" specifies on startup the name of the editor interface program (plugin) eg, "PROGDIR:GedFront", or "PROGDIR:TtxFront". If none is given, "PROGDIR:EdtFront" will be used.

(the -D switch is for internal debugging of M2IDE itself)

Setting up a New Project

Any non-trivial program will be broken up into multiple modules, each of which will need to be compiled (ie, not including stable support modules). These can be created with the editor as needed. Once they are imported into <File>.mod (or one of its dependencies) a Refresh will bring them into the project.

My suggested practice is to create a project directory with two sub-directories, one named "mod/" and the other "def/". If there is an existing program Module copy it into the "mod/" subdirectory, and any existing project IMPLEMENTATION and DEFINITION modules into the "mod/" and "/def" sub-directories respectively.

See notes under the "Open" in [Project Menu](#).

On prj file creation, all the library and other established support modules from Aglet will be automatically read into Registered Modules name list (from the file "regmods.nl") so the *M2IDE* will not try to re-compile them even if used in your project.

The symbol files (.SBM) needed for compiling with IMPORTed module are searched for in the project's directory, and in the assignment "M2LV3:", set up as a multi-assign to the Aglet-supplied support modules, as described in the installation instructions.

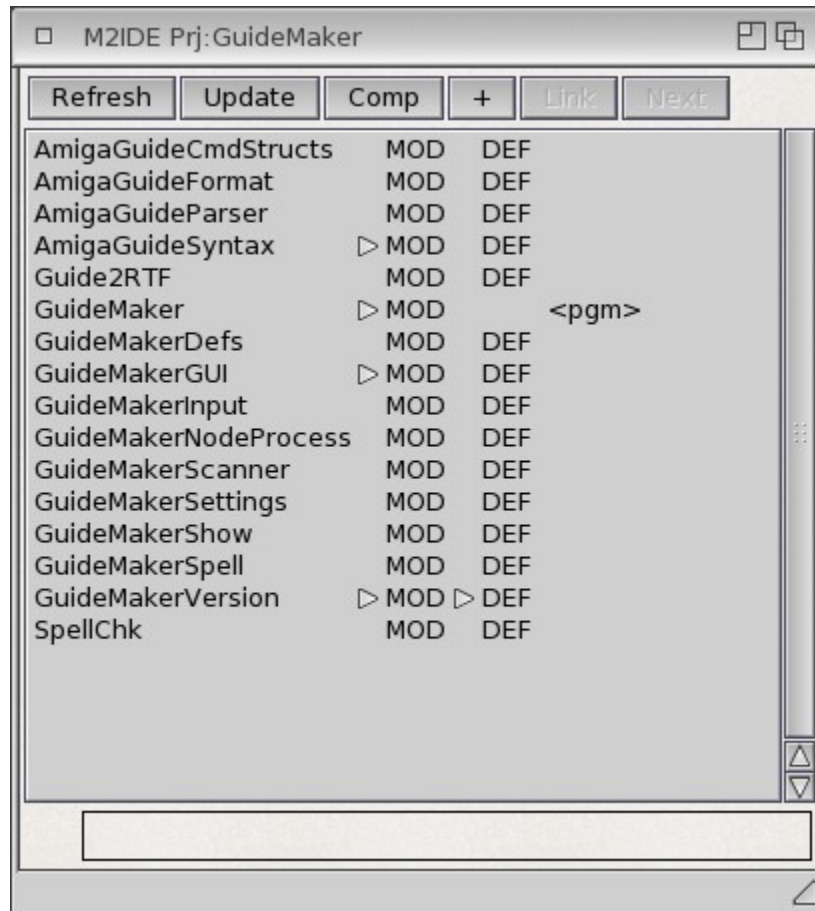
Finally, you may want to put some files into the Misc Files name list for quick access to reference and note files useful for the new project.

Windows

Project Win, Name List Win, Logger Win, BuildPrefs Win, Text Search Win

Menu

Project Menu, Settings Menu, Tools Menu

M2IDE Project Window**Def/Mod List**

Double Clicking on one of the DEF or MOD column entries will send a message to the editor to open (or bring-to-front) the corresponding source file.

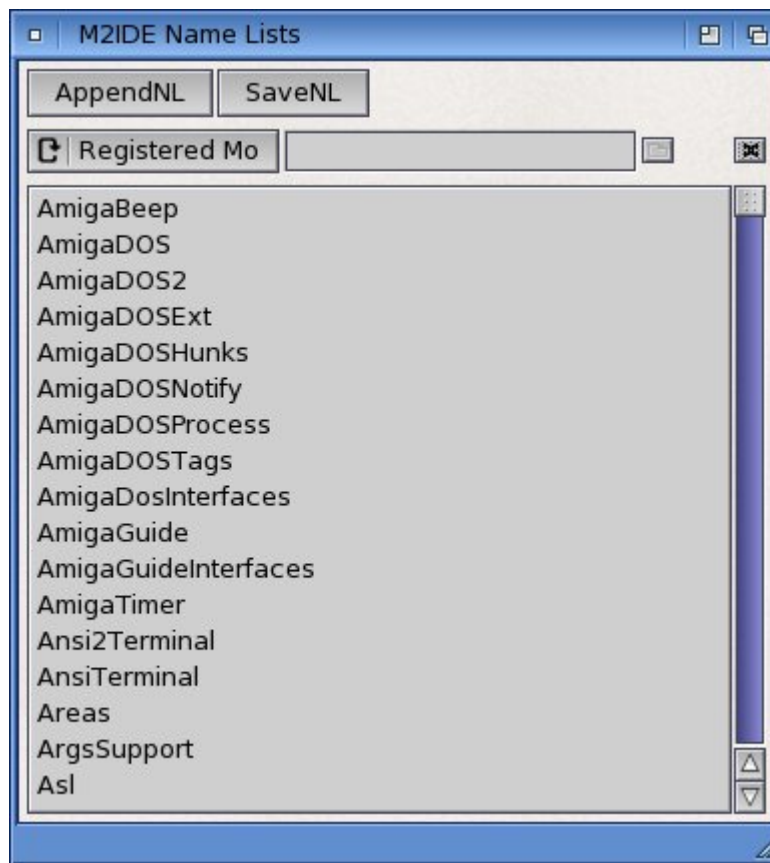
Buttons

- | | |
|---------|--|
| Refresh | Re-check the file dates, and import dependencies, and re-calculate the compiles needed for the entire project. |
| Update | Compile, in appropriate sequence, all Definition and Implementation modules used or referenced by the selected module. |
| Comp | Compile the selected module. If it is a Definition module, mark as "to-be-compiled" any modules dependent on this one. |
| Link | Link the root module (program). |

"+"	Compile the selected module and then Link the root module (program).
Next	Look for an T:*.ERR file for the selected module, and if found, open the editor at the next error position indicated for the source file.

NOTE:

1. The buttons (except for "Refresh") require a currently selected DEF or MOD column entry.
2. The "Comp" and "Update" buttons automatically send a request to the editor to save of the source file(s) before compiling.

M2IDE Name List Window**Name Lists**

Sources Path

List of paths (besides the project file directory) *M2IDE* is to use in searching for DEFINITION and IMPLEMENTATION source files for the project.

LibSources Paths

<NYI>

Registered Mods

These modules are "registered" with this project as "library modules", ie, they are not brought into the Project Window and will not be compiled in building the project. If used in the project, their symbol and object files are expected to be found in one of the directories of the multi-assign, "M2LV3:".

Root Mods

<Currently one root module is allowed, the program file.>

Misc Files

A list of files useful for the project, but not part of the project source. eg, documentation, notes, reminders, library DEFINITION modules. Double clicking on one of these names will open the file into the editor.

Buttons

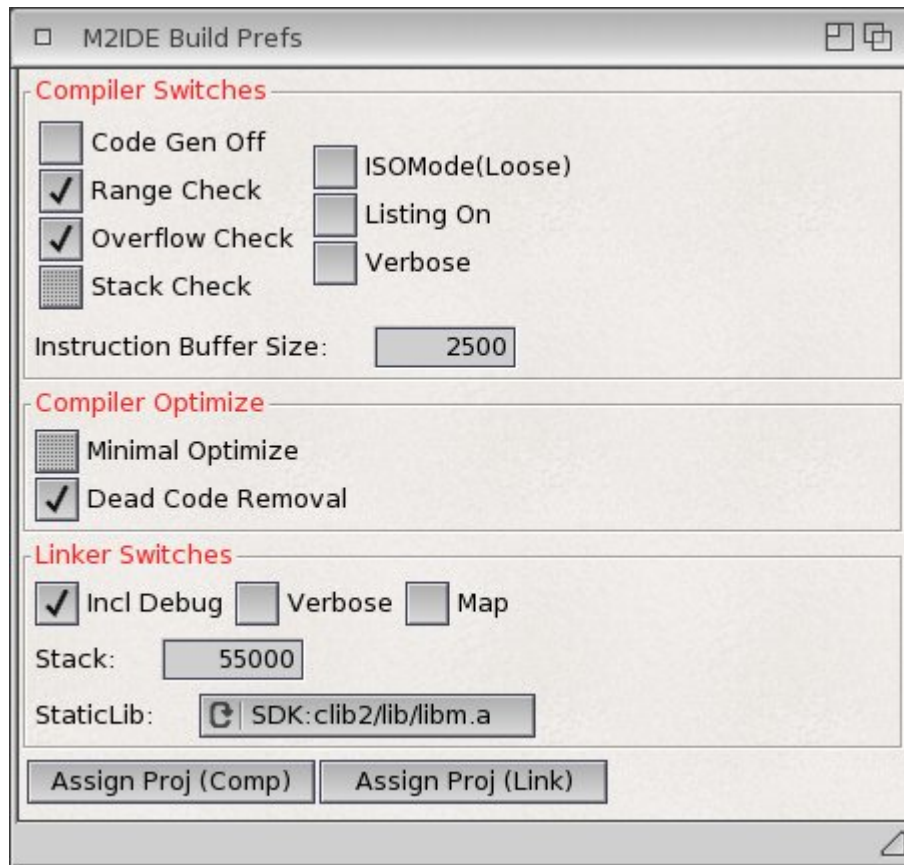
AppendNL	Open a file of names, and merge them into the current project Name List
SaveNL	Save the current project Name List as a text file of names.

Supplied Name List Files

For use in the Registered Mods name list, the following text files of names are copied over during the installation:

System.nl	Compiler RTS modules
Amiga.nl	Amiga library modules
ISO.nl	ISO Standard Library modules
Reaction.nl	Reaction Support
Sysmod.nl	General programming support modules distributed with <i>Aglet M2 PPC</i>
Experimental.nl	In-progress modules distributed with <i>Aglet M2 PPC</i>
RegMods.nl	All the above name lists combined. The program automatically reads in this .nl file on creation of a new project. Generally, this is what you want in order not to edit or compile any of the supplied support Modules, which remain stable across your own projects.

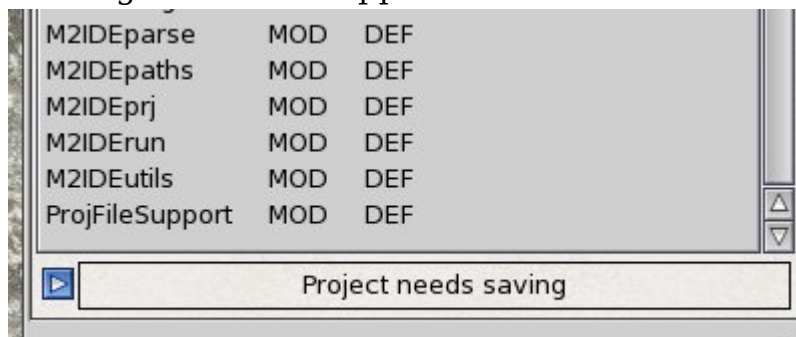
M2IDE BuildPrefs Window



To permanently associate the comp or link settings with the open project you need to hit the "Assign Proj (Comp)" or the "Assign Proj (Link)" buttons.

As of now, only a project-global compiler/linker settings configuration is available. In other words, if the .prj file contains compiler settings, they will be used for all modules in the project (unless changed by the user via the Build window).

Note: These buttons assign the setting to the project, but there is no project auto-saving; you still need to save the project itself for the assigned settings to be available next time you start the project up. You can do this from the File | Save menu item or from the Needs Saving button that appears at the bottom of the main project window.



The global environmental settings (see [Settings Menu](#)) are used if the project being opened has no associated project specific setting.

Currently, from the M2IDE GUI, there are only two settings for the "StaticLib" to be included in the link, "<none>" or "SDK:clib2/lib/libm.a".

M2IDE Logger WindowA screenshot of a window titled "M2IDE" with standard window controls (minimize, maximize, close) in the top right corner. The window contains a text area with the following text:

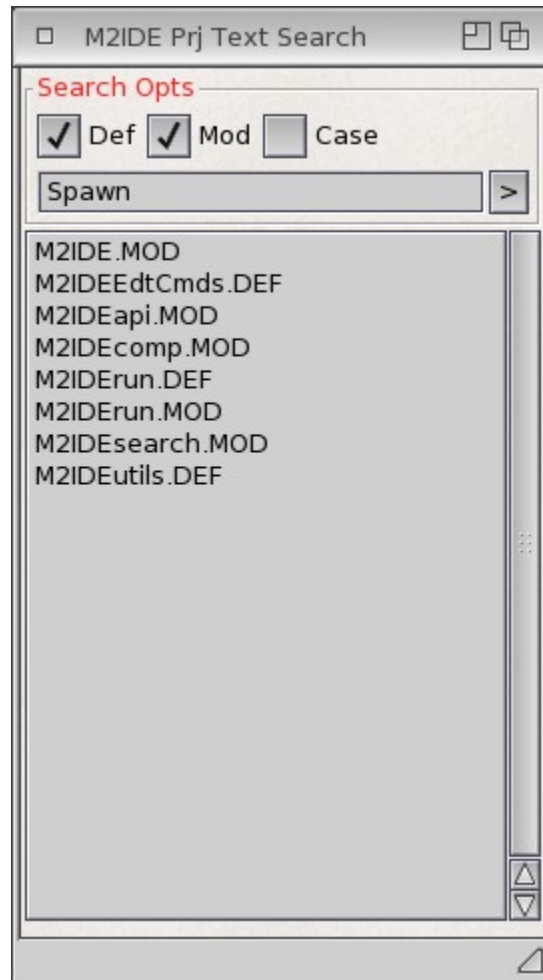
```
Aglet Mod2
PROGDIR:Mod2 -rngchk -ovflchk -instrubuf 5000 -optlev 2 GuideMaker.MOD

Aglet PPC Mod2 v3.1 Compiler Beta1 (15.7.2009)
Copyright (c) 2004 by Thomas Breeden
GuideMaker.MOD
<- M2Lv3:Str1.SBM

-> T:GuideMaker.asm

Optimize Setting: DeadCode
SDK:gcc/bin/as -o GuideMaker.o T:GuideMaker.asm 0
```

All output from the compiles, links, and the Get Clients command appears in this window.

M2IDE Text Search Window**Controls**

String Input Box	Enter the text for which you want to search.
">" Button	Start the search (equivalent to the Enter key in the String box).
Def Box	Check this to include project DEFINITION modules in the search.
Mod Box	Check this to include project IMPLEMENTATION modules in the search.
Case Box	Check this make the search case-sensitive.

FileList

Files found by the search are listed in the bottom part of the window.

Double clicking on one will ask the editor to display that file at the position of the first instance of match. "FindNext" commands in the editor will enable you to step through all matches in the file.

Project Menu

Open

Open an existing (or create a new) *.prj file. Only one can be open at a time.

If no .prj file exists in the directory selected, M2IDE will create (but not yet save) a new project.

It expects to find the program module in a subdirectory "mod/" and will automatically add "mod/" and "def/" to the sources path list. The usual setup is that all the project's IMPLEMENTATION modules and the program module will go into a "mod/" subdirectory under the project's directory. Ditto for DEFINITION modules and a "def/" subdirectory. *M2IDE* automatically adds these two sub-directories into the Sources Path list in the Name List Window. (You will need to create these directories yourself, however).

You can, however, put everything in the project directory itself, or set up a different arrangement via changes to the Sources Paths list in the Name Lists Window. eg,

Multiple project files could go into the same directory, with specific mod and def sub directories for each one, such as "mod-PgmA/", "def-PgmA/", "mod-PgmB/", "def-PgmB/", "mod-PgmC/", "def-PgmC/". You need to explicitly put the appropriate sub-directories into the Sources Path list in the Name List Window for the appropriate project.

If there is no existing program module with the project name, M2IDE will offer to create a template program module file, preferably in the "mod/" subdirectory, but instead in the *.prj file directory if there is no "mod/" subdirectory.

Save

Save the currently open *.prj file.

About

Guide

Runs Multiview on the AmigaGuide AgletM2PPC (this document).
M2IDE expects that AgletM2PPC.guide resides in "PROGDIR:Docs/".

Close

Closes the currently open Project without quitting the program.

Quit

Exit the program. If the *.prj file settings have changed, you will be asked if you want to save it.

Settings Menu

Save Window Layout (ENVARC:)

The current positions and sizes of the M2IDE Project, Name Lists, and Build Settings windows will be saved in ENVARC.

Save Comp Setting (ENVARC:)

Save the compiler setting into ENVARC. These will be used for all new projects and for any project that has not had an explicit set of compiler settings assigned to it via the Build Prefs window button.

Save Link Setting (ENVARC:)

As above, but for linker settings.

Make Icons?

If checked, then when the project file is saved, an icon will be created for it, with *M2IDE* as its default tool.

Also, after a successful link, an Tool icon will be created for the executable file if none already exists.

Show Hints?

If checked, then gadget "bubble" hints are turned on.

Show Build Prefs ...

Open the Build Prefs window if it is not currently open or bring it to the front if it is.

Show Name Lists ...

As above, but for the Name Lists window.

Tools Menu

Get Clients

List all the client modules (in the Logger Window) of Proj window's currently selected DEF file.

Mark Mods Uncompiled

Mark all the MOD files listed in the Proj window as needing compilation.

This will only affect the IMPLEMENTATION modules. The modules' public DEFINITION keystone is unchanged, so any mods not in your project which might depend on project modules will not need recompile.

Note that the mark-uncompiled action does not change the time-stamps on the files themselves, so a Refresh in the Proj window will undo this marking.

Mark All Uncompiled

Mark all files listed in the Proj window as needing compilation, both DEF and MOD.

As above, except that any module not in your project that imports one of the project mods will require re-compiling.

Make Build File

Write out a ADOS shell script which can be used to compile the whole project from scratch, in the appropriate order.

StringSearchFiles...

Opens up the project Text Search Window which can be used to automate a global string search through the project files.

M2IDE Other Info

Environment Variables

M2IDE/DefaultRegMods Name of a Names list file to use to load registered modules upon new project creation. Default is "PROGDIR:RegMods.NL"

M2IDE/CompPrefs	<set by M2IDE>
M2IDE/LinkPrefs	<set by M2IDE>
M2IDE/PgmSettings	<set by M2IDE>
M2IDE/WinPos	<set by M2IDE>

Installation

Executables: M2IDE, EdtFront, M2IDELogger

Uses "PROGDIR:" for starting compiler, error lister, and pre-linker as well, so the above executables should be installed in the same directory as Mod2, M2Err, and Mod2Lnk.

AmigaGuide

Currently (AOS v4.1 upd 4), display of an image link by Multiview (or other program) using the AmigaGuide datatype into the AG window is OK *so long as the user does not attempt to resize the window while the image is up* . Resizing will usually hang the OS.

The AgletM2PPC.guide file has a number of these linked images. You can safely view the image and click on the "Retrace" button, just don't click on the resize gadget while the image is displayed.

KingCON

If you run *M2IDE* from the *KingCON* shell it is a good idea to redirect output to NIL:, eg, "run M2IDE filename > NIL:".

Otherwise, *KingCON* will block *M2IDE* whenever a partial line is typed into the *KingCON* window (even though *M2IDE* redirects standard output).

Release History - Current

Feb 15, 2012 - Release 3.2

Compiler - Mod2 v3.2 Beta (13.2.2012)

> Bugfixes:

WB Started Pgms

- Some programs, if started from WB, were crashing the system on exit. SystemRTS now saves/restores CR.

CAP

- Standard func CAP() was generating bad code in some circumstances.

SYSTEM.Setsreg()

- Proc SYSTEM.Setsreg(), set special PPC reg, works now (except for FPSCR).

DIV/MOD

- Code generated for structured variables was not calculating DIV and MOD correctly, eg. "ar[2] MOD ar[3]" could give the wrong result.

Compile Listing

- The compiler was sometimes giving an index exception when the "-list" CLI switch was used.

POINTER declares

- Some declarations of pointers to never declared types were not being detected. e.g. in
"TYPE arrtyp = ARRAY[0..9] OF POINTER TO <fwdtype>"

FOR limit

- By ISO, the FOR loop limit must be expression compatible with the index variable. This was not being checked by the parser.

FOR index

- Counting down the the MIN(CARDINAL/INTEGER) or up to the MAX(CARDINAL/INTEGER) was not working correctly.

VAL() Glitch

- The compiler was not accepting the valid syntax "VAL(REAL, r)" if r is a REAL.

Overflow Checking

- For the small sized whole number types (INTEGER8, CARDINAL8, INTEGER16, CARDINAL16), some arithmetic expressions with constants were not detecting overflow.

Amiga Version String

- The executable's Amiga Version string is now really Amiga standard.

> Updates:

Local Modules

- Local modules are now fully implemented. An empty BEGIN section is accepted and the FINALLY sequence follows ISO specs.

ISO Tokens

- now "(!" is accepted as "[" and "(:" as "{", similarly the ending bracket/brace. Compiler will now accept any character code within a string literal except 0C, LF, CR, and FF.

Embedded assembler

- Added PPC opcodes for fsqrt and fsqrts.

FOR loop code gen

- Code generation for FOR loops improved. Overhead of

- TSIZE() empty FOR loop reduced about 30%.
- SIZE()
 - SYSTEM.TSIZE() function with one parameter is implemented now.
 - Per ISO, SIZE() and TSIZE() call on a variable must be an "entire designator".

Pre-Linker - Mod2Lnk v0.5 (12/21/2011)

- Exit status - Now Mod2Lnk returns an appropriate Amiga status code reflecting its own processing, the assemble of <pgm>_start, and ld 's run.
- Newlib .so - Mod2Lnk will link with "libc.so" if requested.

Aglet Modules

- IconSupport
 - BUGFIX: mode UpdateWIM was not preserving all the current tooltypes.
 - UPDATE: New proc, "ReadProjIcon()".
 - UPDATE: New proc, "WriteToolIcon()".
- SimpleGUI
 - BUGFIX: two SelectEv events could be queued for the StringSG.
 - BUGFIX: ShowWndoBusy() to turn busy icon off was not working.
- HashT
 - BUGFIX: CreateHT() was ignoring parameters hfunc and cfunc.
 - UPDATE: Increased the MaxTableSize to 500000.
 - UPDATE: Replaced the exported string hash funcs with a better one.
- PipeIO
 - UPDATE: proc OpenPipeUnique() was added.
 - UPDATE: procs ReadTimeoutExceptionOn() and ReadTimeoutExceptionOff() were added. When "On" then Reads can timeout, and Read() will return FALSE. ReadPipeLines() can return with an empty or incomplete line.
- MachineEnv
 - UPDATE: proc GetTotalmemory() was added.
- ArgsSupport
 - UPDATE: proc dProcessCLI() added to support long lines.

Amiga Modules

- SDK 53.20 - Most DEFINITION modules synched with .H files from SDK release 53.20.

Reaction Modules

- IconSupport - STRINGA_DisablePopup tag added.

M2IDE - M2IDE v0.6 (02/02/2012)

- Proj Global Text Search - New feature, and window for results: Text search of all project files (currently using system "Search" program).
- Help Hint Bubbles - Most gadgets have pop-up help hints.
- AmigaGuide docs - Will open via a menu option.
- Executable File Icon - If "MakeIcons" is checked, creates a icon for the executable after linking. Before, only the .prj file got an icon.
- Unfound File - Now displaying a "<?>" in the last column of the main window if one or both of the DEF and MOD files for the module was not found.
- Import Parsing Fails - Pop-up warning when parsing does not succeed for the IMPORT section of a module; project dependencies will not be complete.
- Import Parsing Stack - Size of parsing stack doubled. M2IDE can now parse very big projects (bigger than the compiler).
- Unique PIPE Names - Pipe communication glitches no longer will require rebooting.
- Mark All Mods Uncompiled - To complement the Mark All Files Uncompiled option, which includes all the DEFINITION files as well.
- Pgm Module Template - On creating a new project, if the program module is not found in the selected directory, or a "mod/" subdirectory, then M2IDE will create a simple skeleton program file for the project.
- Link Result Status - M2IDE message box reports success or failure of the whole linking operation.
- NewLib - C NewLib .so library can be chosen to implement ISO RealMath instead of static CLib2 if desired.
- Spaces in Paths - Spawned compiler and linker commands and ARexx messages should all have appropriate quoting for files/paths with spaces.
- Import Parsing - Certain non-ISO (and non-Aglet) IMPORT syntax will be tolerated without error by M2IDE dependency analysis, eg, "IMPORT S : SYSTEM;" as found in M2Amiga. Here the alias name is ignored.

Release History - 2011Apr 5, 2011**Compiler** - v3.1 Beta (13.2.2011)

Bugfixes:

- TwoPower const Expressions "CARDINAL8 * TwoPowerConst -> CARDINAL32", were not generating correct code.
- Char Consts "CAP(c)", where c was a CONST CHAR was giving a compile error.
- Const params to std funcs "INT(801.0)", would malfunction on the second use.

- M2IDE**
- New editor "plug-in", AnnFront, integrates editor *Annotate* into M2IDE.
 - Status message now reports "ok" on successful compiles.

- Amiga Libs**
- updated for release 53.20
 - DOS FileHandle and FileLock are now opaque types rather than ADDRESS.
 - Rasters is using AreaInfoPtr instead of ADDRESS for appropriate parameters.
 - GraphicsInterfaces VectorBuffer parameter for InitArea() changed from ARRAY OF POINT to ADDRESS. (It was not really an array of point).

- Reaction**
- updated for release 53.20
 - ReactionPrefs and UserInterfPrefs moved to directory "Reaction" from directory "Amiga".

- ISO Mods**
- RealStr v0.3 (10.10.2010)
 - > Rounding is now implemented fairly correctly.
 - > Changed code to compensate for a very infrequent glitch in compiler's TRUNC() code generation.
 - > RealToFixed() is now working for reals bigger than approx 4294967300 (ie, $2^{32}-1$)
 - > WriteReal() and RealToFloat() now use the output space as ISO specifies.

- Aglet Mods**
- AmigaTimer v1.6 (13.12.2010)
 - > Supporting new timer modes for "waituntil" and "entropy". Changed mode from a number to an enumeration type, TimerFlavors.
 - > New procs GetEClock(), GetEClockFreq() and GetEntropyATimer()
 - > A TimerHandle is no longer needed as a parameter for

- procs GetSysTime() and GetSysTOD()
- StrMacros v0.2 (3.4.2011)
 - > missing comma between parameters in a macro call no long hang the program.
- SimpRexx1 v0.3 (13.3.2011)
 - > Added automatic single quotes around Port name (necessary if port name does not happen to be all caps.
- PipeIO v0.6 (16.2.2011)
 - > Added ReadPipeLine() for more efficient buffered reading line by line.
 - > Similarly, added dReadPipeLine() unlimited line lengths.
 - > added a Timeout Exception feature.

Experimental

- OsRun v0.3 (27.3.2011)
 - > added Proc SpawnRedirect() to provide for specifying the standard in/out/err
 - > added Proc BreakOffspring() to send ^C to all Spawned children.
 - > added Procs WaitForStarted() and CheckForProcessRunning().
- SimpleGraphics v1.3 (16.2.2011)
 - > Implemented BufferedScreens, at least enough for straightforward animations.
 - > Methods ForeColor Get/Set moved from class GraphPlot up into parent class GraphRegion.
 - > other bug fixes
- Obj v1.1 (16.5.2010)
 - > added func ooClassSameOrDescendent() so that you can test this, not just assert it.

Release History - 2010Feb 28, 2010**Compiler** - v3.1 Beta1 (22.2.2009)

Bugfixes:

- Literal Set Expressions - expressions like "3 IN BITSET{0,3}" were not generating correct code.
- Shift code - SHIFT() to left for set sizes < 32 was resulting in erroneous relation expressions and shift amount of 0 was generating incorrect code.
- Runtime checking - wrong bits in XER register were being referenced
Subrange of CHAR - compiler was rejecting subrange with literal chars like ['a'..'z']
- Set of Boolean Comparison Comparisons - compiler was rejecting type def SET OF BOOLEAN
- expressions like "(i = 5) = (j = 10)" were not handled correctly.
- Const Boolean Expression - const boolean expressions like "TRUE OR TRUE" and "DEBUG OR FALSE" were not being handled correctly.
- Real LiteralExpressions - compiler rejected expressions like "12.343 - b", where b is REAL because the literal number was being treated as LONGREAL only.

Changes:

- Default instruction buffer size - Changed the default to 5000. Use the "-instrubuf" switch to compile with a different size

M2IDE

- new CLI startup switch, "-Editor"
Specifies on startup the name of the "...Front" editor interface program eg, "PROGDIR:GedFront", or "PROGDIR:TtxFront". If none is given, "PROGDIR:EdtFront" will be used as before. The EditorInterfacePort still must be "M2IDEFront".
- Removed LogWindowToFront() calls before each compile and each link. These made it extremely hard to do anything else while compiling a big batch of modules.
- Changes in editor interface program, GedFront, for Golded v8:
 - > Error display in editor had broken with switch from GoldEd v7 to v8.
 - > On startup of GoldEd, the "HIDE" switch was removed - does not work in GoldEd v8.

System Mods

- Bugfix: The default input/output window opened when a program is started from Workbench were not taking input from the keyboard. Now using CON: rather than RAW: for this window.
- Module DebugIO (and thus also Debugging.DebugPause) now has a 60 second timeout.

Aglet Mods

- DateSupport module procs now interpret "AM" and "PM" in a time string.
- StrSubstitutes.DoSubst() now implements backslash as an "escape" char for brackets. This also affects modules StrMacros and SimpleStrSubstitutes.

Experimental

- New module for programatically starting independent CLI processes: OsRun.
- New module, DirUtilsDyn, using dynamic rather than static strings for file specs.

TestManager

- Created a q+d program to manager automated runs of the tgM2 test generator program. A dozen or so tgM2 driver files put together to create the beginning of an effective compiler regression testing suite.

Release History - 2009

Dec 16, 2009

Compiler

- v3.1 Beta1 (16.12.2009)
- The compiler itself is now **PPC native**, and is significantly faster!
- The size limit on procedures is now probably large enough for any sane program.
- A bug which caused extremely excessive stack use was fixed. Now 40 or 50K should be enough for almost any compile.
- Some code generation errors were fixed.
- Fixed problem with Opaque pointers resolved within the IMPLEMENTATION module by imported pointer types.
- Added compiler warning: OpenArrayCopyWarn.
- SDK 53.8+ now using the GNU assembler v2.18 vs. 2.14 in earlier SDKs. This required that some instructions be output slightly differently.
- Certain "recording" forms of FP instructions no longer generated, since Sam 440ep does not support them.

(Pre)Linker

- *Mod2Lnk* recognizes a "**-stack**" **switch** and inserts the "\$STACK:xxxx" cookie into executables.
- The .asm files no longer need be kept around. All (pre)linking information is in the object file now. Asm files are written to T:
- Added the "**-g**" **debug switch**, which causes both exported and non-exported symbols to be put into the Elf symbol table so that **SymbolsRTS** can find them as well.

M2IDE

- v0.3 (30.8.2009)
- Option "**Make Icons?**" added for the project file save.
- Fixed problem of GR on exit if M2IDE changed its current directory.
- Multiselect now supported for the "Misc Files" file requester.

Amiga Modules

- Added Definition files for about **25 more Amiga Libraries**, and all supplied Amiga definition files (over 150 of them) were brought up to the v53.13 SDK.
- The **TextEditor gadget** is now working much better, as the Definition file now adjusts to a glitch in the SDK's .h file.

Aglet Modules

- System module SymbolsRTS was introduced for better debugging of exception locations.
- CLI program arguments and Workbench ToolTypes are transparently (almost) unified when the ArgsSupport module is used to read startup arguments.
- Many improvements in the Simple... modules (SimpleGUI,

SimpleScreens, SimpleMenus, SimpleGraphics, etc), but considering their overall incompleteness and fluidity I moved them into their own folder, "Experimental".

- new modules: **BigInt**, for 155 bit integers; **IconSupport**, for writing out icons.

Release History - 2008

Oct 26, 2008

M2IDE - Fixed the intermittent, irritating, "broken PIPE" errors that tended to show up in the IDE.

Aug 10, 2008

Compiler - v3.0 Beta0 (3.8.2008)

Documentation - Expanded, corrected, and supplied as a PDF file as well as AmigaGuide.

M2IDE - Usability improvements in GUI.

Aglet Modules - small improvements in SimpleGUI and SimpleImageHandler modules
- RTFWrite supports many more RTF constructs
- DynStr2B: new procedures dAddQuotes(), dRemoveQuotes().

July 5, 2008

- first one: v3.0 Alpha0 (22.6.2008)

ISO Modula-2 Syntax

Thanks to special permission of ISO/CS in Geneva, WG13 is allowed to make the concrete syntax of Modula-2 and the text of all the definition modules in the standard (ISO/IEC 10514-1) available.

compilation module =
program module | definition module | implementation module ;

program module =
"MODULE", module identifier, [interrupt protection], semicolon,
import lists,
module block, module identifier, period ;

module identifier =
identifier ;

definition module =
"DEFINITION", "MODULE", module identifier, semicolon,
import lists, definitions,
"END", module identifier, period ;

implementation module =
"IMPLEMENTATION", "MODULE", module identifier,
[interrupt protection], semicolon,
import lists,
module block, module identifier, period ;

interrupt protection =
left bracket, protection expression, right bracket ;

protection expression =
constant expression ;

module block =
declarations, [module body], "END" ;

module body =
initialization body, [finalization body] ;

initialization body =
"BEGIN", block body ;

finalization body =
"FINALLY", block body ;

block body =
normal part, ["EXCEPT", exceptional part] ;

normal part =

ISO Modula-2 Syntax

statement sequence ;

exceptional part =
statement sequence ;

import lists =
{ import list } ;

import list =
simple import | unqualified import ;

simple import =
"IMPORT", identifier list, semicolon ;

unqualified import =
"FROM", module identifier, "IMPORT", identifier list, semicolon ;

export list =
unqualified export | qualified export ;

unqualified export =
"EXPORT", identifier list, semicolon ;

qualified export =
"EXPORT", "QUALIFIED", identifier list, semicolon ;

qualified identifier =
{ qualifying identifier, period }, identifier ;

qualifying identifier =
module identifier ;

definitions =
{ definition } ;

definition =
"CONST", { constant declaration, semicolon } |
"TYPE", { type definition, semicolon } |
"VAR", { variable declaration, semicolon } |
procedure heading, semicolon ;

procedure heading =
proper procedure heading | function procedure heading ;

type definition =
type declaration | opaque type definition ;

opaque type definition =
identifier ;

ISO Modula-2 Syntax

declarations =
{ declaration } ;

declaration =
"CONST", { constant declaration, semicolon } |
"TYPE", { type declaration, semicolon } |
"VAR", { variable declaration, semicolon } |
procedure declaration, semicolon |
local module declaration, semicolon ;

constant declaration =
identifier, equals, constant expression ;

type declaration =
identifier, equals, type denoter ;

variable declaration =
variable identifier list, colon, type denoter ;

variable identifier list =
identifier, [machine address], { comma, identifier, [machine address] } ;

machine address =
left bracket, value of address type, right bracket ;

value of address type =
constant expression ;

procedure declaration =
proper procedure declaration | function procedure declaration ;

proper procedure declaration =
proper procedure heading, semicolon,
(proper procedure block, procedure identifier | "FORWARD") ;

procedure identifier =
identifier ;

proper procedure heading =
"PROCEDURE", procedure identifier, [formal parameters] ;

formal parameters =
left parenthesis, [formal parameter list], right parenthesis ;

formal parameter list =
formal parameter, { semicolon, formal parameter } ;

proper procedure block =

ISO Modula-2 Syntax

declarations, [procedure body], "END" ;

procedure body =
"BEGIN", block body ;

function procedure declaration =
function procedure heading, semicolon,
(function procedure block, procedure identifier | "FORWARD") ;

function procedure heading =
"PROCEDURE", procedure identifier, formal parameters,
colon, function result type ;

function result type =
type identifier ;

function procedure block =
declarations, function body, "END" ;

function body =
"BEGIN", block body ;

formal parameter =
value parameter specification | variable parameter specification ;

value parameter specification =
identifier list, colon, formal type ;

variable parameter specification =
"VAR", identifier list, colon, formal type ;

local module declaration =
"MODULE", module identifier, [interrupt protection], semicolon,
import lists,
[export list],
module block, module identifier ;

type denoter =
type identifier | new type ;

ordinal type denoter =
ordinal type identifier | new ordinal type ;

type identifier =
qualified identifier ;

ordinal type identifier =
type identifier ;

ISO Modula-2 Syntax

new type =
new ordinal type |
set type |
packedset type |
pointer type |
procedure type |
array type |
record type ;

new ordinal type =
enumeration type | subrange type ;

enumeration type =
left parenthesis, identifier list, right parenthesis ;

identifier list =
identifier, { comma, identifier } ;

subrange type =
[range type], left bracket, constant expression, ellipsis,
constant expression, right bracket ;

range type =
ordinal type identifier ;

set type =
"SET", "OF", base type ;

base type =
ordinal type denoter ;

packedset type =
"PACKEDSET", "OF", base type ;

pointer type =
"POINTER", "TO", bound type ;

bound type =
type denoter ;

procedure type =
proper procedure type | function procedure type ;

proper procedure type =
"PROCEDURE",
[left parenthesis, [formal parameter type list], right parenthesis] ;

function procedure type =
"PROCEDURE", left parenthesis, [formal parameter type list],

ISO Modula-2 Syntax

right parenthesis, colon, function result type ;

formal parameter type list =
formal parameter type, { comma, formal parameter type } ;

formal parameter type =
variable formal type | value formal type ;

variable formal type =
"VAR", formal type ;

value formal type =
formal type ;

formal type =
type identifier | open array formal type ;

open array formal type =
"ARRAY", "OF", open array component type ;

open array component type =
formal type ;

array type =
"ARRAY", index type, { comma, index type }, "OF", component type ;

index type =
ordinal type denoter ;

component type =
type denoter ;

record type =
"RECORD", field list, "END" ;

field list =
fields, { semicolon, fields } ;

fields =
[fixed fields | variant fields] ;

fixed fields =
identifier list, colon, field type ;

field type =
type denoter ;

variant fields =
"CASE", tag field, "OF", variant list, "END" ;

ISO Modula-2 Syntax

tag field =
[tag identifier], colon, tag type ;

tag identifier =
identifier ;

tag type =
ordinal type identifier ;

variant list =
variant, { case separator, variant },
[variant else part] ;

variant else part =
"ELSE", field list ;

variant =
[variant label list, colon, field list] ;

variant label list =
variant label, { comma, variant label } ;

variant label =
constant expression, [ellipsis, constant expression] ;

statement =
empty statement |
assignment statement |
procedure call |
return statement |
retry statement |
with statement |
if statement |
case statement |
while statement |
repeat statement |
loop statement |
exit statement |
for statement ;

statement sequence =
statement, { semicolon, statement } ;

empty statement =
;

assignment statement =
variable designator, assignment operator, expression ;

ISO Modula-2 Syntax

procedure call =
procedure designator, [actual parameters] ;

procedure designator =
value designator ;

actual parameters =
left parenthesis, [actual parameter list], right parenthesis ;

actual parameter list =
actual parameter, { comma, actual parameter } ;

actual parameter =
variable designator | expression | type parameter ;

type parameter =
type identifier ;

return statement =
simple return statement | function return statement ;

simple return statement =
"RETURN" ;

function return statement =
"RETURN", expression ;

retry statement =
"RETRY" ;

with statement =
"WITH", record designator, "DO", statement sequence, "END" ;

record designator =
variable designator | value designator ;

if statement =
guarded statements, [if else part], "END" ;

guarded statements =
"IF", boolean expression, "THEN", statement sequence,
{ "ELSIF", boolean expression, "THEN", statement sequence } ;

if else part =
"ELSE", statement sequence ;

boolean expression =
expression ;

ISO Modula-2 Syntax

case statement =
"CASE", case selector, "OF", case list, "END" ;

case selector =
ordinal expression ;

case list =
case alternative, { case separator, case alternative }, [case else part] ;

case else part =
"ELSE", statement sequence ;

case alternative =
[case label list, colon, statement sequence] ;

case label list =
case label, { comma, case label } ;

case label =
constant expression, [ellipsis, constant expression] ;

while statement =
"WHILE", boolean expression, "DO", statement sequence, "END" ;

repeat statement =
"REPEAT", statement sequence, "UNTIL", boolean expression ;

loop statement =
"LOOP", statement sequence, "END" ;

exit statement =
"EXIT" ;

for statement =
"FOR", control variable identifier, assignment operator, initial value,
"TO", final value,
["BY", step size],
"DO",
statement sequence, "END" ;

control variable identifier =
identifier ;

initial value =
ordinal expression ;

final value =
ordinal expression ;

ISO Modula-2 Syntax

step size =
constant expression ;

variable designator =
entire designator |
indexed designator |
selected designator |
dereferenced designator ;

entire designator =
qualified identifier ;

indexed designator =
array variable designator,
left bracket, index expression, { comma, index expression },
right bracket ;

array variable designator =
variable designator ;

index expression =
ordinal expression ;

selected designator =
record variable designator, period, field identifier ;

record variable designator =
variable designator ;

field identifier =
identifier ;

dereferenced designator =
pointer variable designator, dereferencing operator ;

pointer variable designator =
variable designator ;

expression =
simple expression, [relational operator, simple expression] ;

simple expression =
[sign], term, { term operator, term } ;

term =
factor, { factor operator, factor } ;

factor =

ISO Modula-2 Syntax

left parenthesis, expression, right parenthesis |
logical negation operator, factor |
value designator |
function call |
value constructor |
constant literal ;

relational operator =
equals operator |
inequality operator |
less than operator |
greater than operator |
less than or equal operator |
subset operator |
greater than or equal operator |
superset operator |
set membership operator ;

term operator =
plus operator |
set union operator |
minus operator |
set difference operator |
logical disjunction operator |
string catenate symbol ;

factor operator =
multiplication operator |
set intersection operator |
division operator |
symmetric set difference operator |
rem operator |
div operator |
mod operator |
logical conjunction operator ;

value designator =
entire value |
indexed value |
selected value |
dereferenced value ;

entire value =
qualified identifier ;

indexed value =
array value, left bracket,
index expression, { comma, index expression },
right bracket ;

ISO Modula-2 Syntax

array value =
value designator ;

selected value =
record value, period, field identifier ;

record value =
value designator ;

dereferenced value =
pointer value, dereferencing operator ;

pointer value =
value designator ;

function call =
function designator, actual parameters ;

function designator =
value designator ;

value constructor =
array constructor | record constructor | set constructor ;

array constructor =
array type identifier, array constructed value ;

array type identifier =
type identifier ;

array constructed value =
left brace, repeated structure component,
{ comma, repeated structure component },
right brace ;

repeated structure component =
structure component, ["BY", repetition factor] ;

repetition factor =
constant expression ;

structure component =
expression |
array constructed value |
record constructed value |
set constructed value ;

record constructor =

ISO Modula-2 Syntax

record type identifier, record constructed value ;

record type identifier =
type identifier ;

record constructed value =
left brace,
[structure component, { comma, structure component }],
right brace ;

set constructor =
set type identifier, set constructed value ;

set type identifier =
type identifier ;

set constructed value =
left brace, [member, { comma, member }], right brace ;

member =
interval | singleton ;

interval =
ordinal expression, ellipsis, ordinal expression ;

singleton =
ordinal expression ;

constant literal =
whole number literal |
real literal |
string literal ;

ordinal expression =
expression ;

constant expression =
expression ;

assignment operator = as defined in the Lexis, section 5

case separator = as defined in the Lexis, section 5

colon = as defined in the Lexis, section 5

comma = as defined in the Lexis, section 5

dereferencing operator = as defined in the Lexis, section 5

div operator = as defined in the Lexis, section 5

division operator = as defined in the Lexis, section 5

ellipsis = as defined in the Lexis, section 5

equals = as defined in the Lexis, section 5

equals operator = as defined in the Lexis, section 5

ISO Modula-2 Syntax

greater than operator = as defined in the Lexis, section 5
greater than or equal operator = as defined in the Lexis, section 5
identifier = as defined in the Lexis, section 5
inequality operator = as defined in the Lexis, section 5
left brace = as defined in the Lexis, section 5
left bracket = as defined in the Lexis, section 5
left parenthesis = as defined in the Lexis, section 5
less than operator = as defined in the Lexis, section 5
less than or equal operator = as defined in the Lexis, section 5
logical conjunction operator = as defined in the Lexis, section 5
logical disjunction operator = as defined in the Lexis, section 5
logical negation operator = as defined in the Lexis, section 5
minus operator = as defined in the Lexis, section 5
mod operator = as defined in the Lexis, section 5
multiplication operator = as defined in the Lexis, section 5
period = as defined in the Lexis, section 5
plus operator = as defined in the Lexis, section 5
real literal = as defined in the Lexis, section 5
rem operator = as defined in the Lexis, section 5
right brace = as defined in the Lexis, section 5
right bracket = as defined in the Lexis, section 5
right parenthesis = as defined in the Lexis, section 5
semicolon = as defined in the Lexis, section 5
set difference operator = as defined in the Lexis, section 5
set intersection operator = as defined in the Lexis, section 5
set membership operator = as defined in the Lexis, section 5
set union operator = as defined in the Lexis, section 5
sign = as defined in the Lexis, section 5
string catenate symbol = as defined in the Lexis, section 5
string literal = as defined in the Lexis, section 5
subset operator = as defined in the Lexis, section 5
superset operator = as defined in the Lexis, section 5
symmetric set difference operator = as defined in the Lexis, section 5
whole number literal = as defined in the Lexis, section 5
