

Artificial Intelligence + Dynamic and Evolved Code +  
Distributed / Parallel / Concurrent Computing +

Languages + Precision + C++ MML +  
**AndroMeta**

Modeling and Simulation + Visualization +  
Simplicity + Flexibility + High Performance

---

## AndroMeta 2.0 User's Guide

January 7, 2012

Copyright 2012 AndroMeta LLC. All rights reserved.

This guide covers AndroMeta 2.0 and may not be up-to-date with the current release.

<b>Introduction</b>	<b>6</b>
Thank You!	6
Overview	6
Getting Started	7
About This Guide	9
Conventions	9
<b>Framework Overview</b>	<b>11</b>
Data Types	11
Languages	12
Task Concurrency	13
Networking	14
Math, Machine Learning, and General AI	17
And Much More	18
Thread Safety	18
<b>MML (Meta Modeling Language)</b>	<b>19</b>
Overview	19
Commands	19
Data Representation and Operations	20
Numerics	20
Vectors and Lists	21
Maps	22
Built-in mvar Methods	22
Strings	25
Libraries	26

<b>Classes</b>	<b>27</b>
<b>User-Defined Classes</b>	<b>27</b>
<b>Extending Classes</b>	<b>28</b>
<b>Structure of an MML Program</b>	<b>29</b>
<b>Events and Communication</b>	<b>31</b>
<b>Event Handling</b>	<b>31</b>
<b>Locality and Interactions</b>	<b>32</b>
<b>Disconnections and Memory Management</b>	<b>32</b>
<b>More on MMLEntity</b>	<b>32</b>
<b>Graph-based Operations</b>	<b>33</b>
<b>Syntax</b>	<b>33</b>
<b>Overview</b>	<b>33</b>
<b>Loops</b>	<b>34</b>
<b>Conditionals</b>	<b>34</b>
<b>Vectors</b>	<b>35</b>
<b>Default Parameter Values</b>	<b>35</b>
<b>Named Arguments</b>	<b>35</b>
<b>Comments</b>	<b>36</b>
<b>MProgram Arguments</b>	<b>36</b>
<b>Overview</b>	<b>36</b>
<b>Scenes</b>	<b>37</b>
<b>Overview</b>	<b>37</b>
<b>Plotting</b>	<b>37</b>
<b>GUI Controls</b>	<b>37</b>
<b>MPEG Capture</b>	<b>37</b>

<b>Delay and Pausing</b>	<b>37</b>
<b>Advanced Features</b>	<b>38</b>
<b>Threading</b>	<b>38</b>
<b>Closures</b>	<b>38</b>
<b>Data Files</b>	<b>39</b>
<b>Modeling Paradigms</b>	<b>40</b>
<b>MML and the Larger Framework</b>	<b>40</b>
<b>MML++</b>	<b>42</b>
<b>A First Program in MML++</b>	<b>42</b>
<b>Subclassing MObject</b>	<b>44</b>
<b>Distributed Objects</b>	<b>46</b>
<b>MPL</b>	<b>47</b>
<b>Link-level Compatibility</b>	<b>50</b>
<b>MMage</b>	<b>50</b>
<b>Acknowledgements</b>	<b>54</b>
<b>References</b>	<b>55</b>
<b>Appendix</b>	<b>56</b>
<b>Interfacing MMage with Mathematica</b>	<b>56</b>
<b>MProgram Framework Arguments</b>	<b>56</b>
<b>Copyrights and Licenses</b>	<b>58</b>



# Introduction

## Thank You!

If you have purchased an AndroMeta license, we would like to take this opportunity to kindly thank you. Your contribution greatly helps to support our research and software development efforts. AndroMeta is a relatively new software product but we have a lot of ideas and new functionality planned for the future. We are committed to the continued effort of providing you with innovative and high quality software. Your feedback is important to us so please don't hesitate to send any suggestions, problems, or questions to us at: [aminfo@andrometa.net](mailto:aminfo@andrometa.net).

## Overview

AndroMeta is a software system which spans a diverse range of fields including: machine learning, knowledge bases, and artificial intelligence in general, distributed, concurrent, and GPU computing, languages and language design, modeling and simulation, and more – all of which are unified into a powerful yet easy-to-use C++ framework and collection of integrated languages. The constructs that make up AndroMeta's specialized higher-level functionality are all made available as general-purpose classes. In this way, while AndroMeta focuses on technical computing, it has also proven to be a valuable tool for general programming situations as well as it includes several high-quality reusable components. AndroMeta (pronounced like Andromeda) is available for Mac OS X and Linux.

AndroMeta is intended to be accessible to programmers at various levels – one does not have to be a C++ expert to make nearly full use of the system, while at the same time, it provides enough flexibility to facilitate advanced applications. Users may choose to use a multitude of its features or just a few in isolation. AndroMeta is a multi-purpose software system. Some of its intended uses and primary features include:

- An LLVM / Clang-based compiler providing the **MML++** language which is a full superset of C++ and makes several of the powerful features of the AndroMeta framework available as direct language features.
- An extremely high performance sub-framework and embedded language system called **MPL** which harnesses the power of GPU and CPU multiprocessing using easy-to-use interfaces and abstraction layers. MPL stresses highly optimized code generated at runtime, executed in the most efficient manner.
- An interpreted language called **MML** for usages where performance is secondary to flexibility and ease of use. MML incorporates functional programming, higher-level mathematics, event-based modeling and simulation, and easy-to-use three-dimensional visualization in a weakly-typed language similar in form to C++.
- Agent-based modeling, discrete-event simulation, and hybrid simulation. Coding may be done in MML, C++/MML++, or MPL, or using a combination of the three.

-Powerful yet easy-to-use networking constructs for building client/server applications, distributed objects, and peer-to-peer systems. Distributed objects are easily accomplished and are integrated directly into MML++.

-Exact and high-precision numerics. AndroMeta makes use of unified variant data types. Exact math can be performed as well as performing calculations with arbitrary precision.

-Language design: AndroMeta eases some of the difficulties associated with implementing a domain-specific language. AndroMeta provides a powerful interpretable AST called M. Designing a new front-end language is as easy as writing a parser which produces M code which can then take advantage of the various code generators that the framework provides.

-AndroMeta uses advanced artificial intelligence techniques to provide the ability to evolve M code based on supplied training data or using unsupervised learning by generating code modules which are “plugged” into a larger program for evaluation.

-AndroMeta includes a dynamic multi-user knowledge base system, MDL (Meta Domain Language). Domains define event-based attributes and are hierarchical in that they can contain other domains and link to one another.

-AndroMeta features a powerful graph-based task concurrency system which allows an application to easily take advantage of the processing power of multiple cores/CPU's.

## Getting Started

AndroMeta is freely available for academic, non-commercial, or evaluation download from the AndroMeta web site [downloads page](#). If you use AndroMeta for a long-term project, or wish to distribute code or applications that link to the AndroMeta framework, we ask that you first purchase a license. Please contact us about our licensing options if you are interested in using AndroMeta as we are quite flexible.

The AndroMeta shared libraries (and `AndroMeta.framework` on the Mac) are included in the `AndroMeta/lib` directory. For your convenience we have also included the external libraries that AndroMeta links to at this location. Users have the option of either installing the core AndroMeta and external libraries into a standard system directory such as `/usr/local/lib` or setting the appropriate environment variables `$LD_LIBRARY_PATH` (`$DYLD_LIBRARY_PATH` on Mac).

We assume that your system is already set up with standard development tools such as `make`, `g++` or `clang++`, etc. (on Mac as part of XCode) and Python.

After downloading the appropriate AndroMeta archive – either Mac OS X (Intel) or 64-bit (x86\_64) Linux, move the `.tar.gz` file to the location where you'd like your AndroMeta root directory to reside and unpack it, e.g:

```
% tar xzf AndroMeta-2.0.0-Mac.tar.gz
```

AndroMeta requires a few environment variables to be set, the most important of which is `ANDROMETA_HOME`, which gives the path to your AndroMeta root directory as unpacked in the above. Most likely you'll also want to add `AndroMeta/bin` to your `PATH`. So for instance, if you're using the bash shell, the following could be added to your `.bash_profile` (or equivalent):

```
export ANDROMETA_HOME=~/.AndroMeta
export PATH=$PATH:$ANDROMETA_HOME/bin
```

If you do not want to install AndroMeta libraries into a standard system location such as `/usr/local/lib`, then you must:

-On the Mac, set `DYLD_LIBRARY_PATH` and `DYLD_FALLBACK_FRAMEWORK_PATH` to `$ANDROMETA_HOME/lib`, e.g:

```
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:$ANDROMETA_HOME/lib
export DYLD_FALLBACK_FRAMEWORK_PATH=$DYLD_FALLBACK_FRAMEWORK_PATH:/Library/
Frameworks:$ANDROMETA_HOME/lib
```

-On Linux, set `LD_LIBRARY_PATH` to `$ANDROMETA_HOME/lib`, e.g:

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

On the other hand, if you wish to install the AndroMeta and third-party libraries into a standard system location, simply copy the contents of `$ANDROMETA_HOME/lib` to your library directory, e.g: `/usr/local/lib`, making sure that you have the proper privileges, e.g: with `sudo` or as the root user. On Linux you should then run `ldconfig` after performing the copy. On the Mac, you should copy the files named with the `.framework` extension into a standard frameworks location such as: `/Library/Frameworks`.

If you intend to use the `mml++` compiler and do not install the AndroMeta libraries or frameworks in a standard location, you must also set `ANDROMETA_FRAMEWORKS` to the location where the AndroMeta framework / library resides, e.g:

```
export ANDROMETA_FRAMEWORKS=$ANDROMETA_HOME/lib
```

If you use `emacs`, `C++` mode is typically used when editing `MML` and `MML++` source files, so it might be helpful to add the following to your `.emacs` file:

```
(setq auto-mode-alist (cons '("\\.mml$" . c++-mode) auto-mode-alist))
(setq auto-mode-alist (cons '("\\.mcpp$" . c++-mode) auto-mode-alist))
(setq auto-mode-alist (cons '("\\.mh$" . c++-mode) auto-mode-alist))
(setq auto-mode-alist (cons '("\\.mmm$" . objc-mode) auto-mode-alist))
```

If everything was installed and configured correctly, you should now be able to run the AndroMeta interpreter command as a test:

```
% meta
```

MML statements and expressions may be entered at the prompt, e.g:

```
>>> Sqrt(2r)
>>> x = [6,7]
>>> x.push(8)
>>> x + 1
>>> Integrate(x^2 + 1, x)
```

AndroMeta provides `metac` as a convenience command which may be used to run the MML examples. To run an example, `cd` into its directory and type `metac run`, e.g:

```
% cd $ANDROMETA_HOME/examples/high-precision
% metac run
```

You can pass extra options and flags to a command this way:

```
% cd $ANDROMETA_HOME/examples/equilibrium
% metac run -mThreads 4
```

For examples which are compiled with `mml++`, i.e: those examples which include a `Makefile`, simply run `'make'` then run the appropriate binary.

The AndroMeta distribution contains a number of code resources designed to help familiarize you with AndroMeta's various features. Examples written in MML, MML++, and MPL are included in the `$ANDROMETA_HOME/examples` directory. You can also examine the framework header files in:

```
$ANDROMETA_HOME/include/AndroMeta
```

## About This Guide

This guide is a work in progress and does not comprehensively cover all aspects of AndroMeta. However, it does cover MML, MML++, MPL and key components of the framework extensively. It is intended to supplement the examples, headers, and doxygen API reference included with the software distribution. If you notice something missing that you feel needs explaining or have specific questions, please contact us.

## Conventions

We believe that coding conventions and consistency are important to a framework's design. All class and method names in AndroMeta are named according to a well-defined pattern. Abbreviations are avoided for the most part, but are used selectively in the naming of some items. For example, `mvar`, the most common AndroMeta datatype, is named as such instead of the more unwieldy name `mvariant`. We avoid functions, preferring static methods associated with an appropriate class. However, extremely commonly

used calls are the exception, e.g: `mfunc()`, for creating a functional node. Every call is defined in a logical place.

All classes that are part of the official framework are prefixed with an M. This is done in part, because you may wish to create a subclass of one of the `AndroMeta` classes with the same name. Plus, we anticipate that you may be using `AndroMeta` classes in conjunction with other libraries, so M-prefixed classes pose a lesser risk for name conflicts even though all `AndroMeta` code resides under the `Meta` namespace.

All C++ and MML methods are named like this: `someMethod()` while M methods/functions which typically make use of functional programming are named `SomeFunctionalMethod()`. This is done in part to avoid name clashes with potential user-defined methods and to avoid confusion by the different semantics entailed by M calls – namely, in an M call, parameters make use of delayed execution (closures) to pass unevaluated symbols or functions whereas conventional methods are always named beginning in lower-case and have their parameters executed before the call executes. In fact, MML treats function/method calls that begin with an upper-case as special cases whereby it does not evaluate their arguments before the call is made.

An *accessor* returns the value of an object attribute and in `AndroMeta`'s C++ framework classes, accessors are simply the name of the attribute, e.g: `size()`. *Mutators* change the value of an attribute, and follow the convention of: `setSize(value)`. Code defined in MML typically uses the same naming style except accessors are usually named like `getSize()` in order to avoid namespace conflicts. Throughout the framework, classes that provide a `process()` method are able to have their methods called dynamically by an M-based language/interpreter such as MML.

# Framework Overview

## Data Types

-AndroMeta includes a variety of data type classes, covering numerics, strings, containers, variants, and symbolic nodes.

-These classes provide an extensive library of operations, many taking advantage of C++ operator-overloading. STL-style containers such as **MVector** make use of templates to facilitate generic programming.

-The types provide a high degree of interoperability and strike a proper balance between implicit and explicit type conversion to enable concise coding.

-**MVar**, perhaps the most centrally used component throughout the framework, provides a union of the more basic types, allowing recursion for representation of arbitrarily complex data.

-In many cases MVar alleviates the need for static data structures which provides greater dynamic/run-time flexibility and simplifies many common programming situations, however in some cases such as for performance reasons, static data structures are employed.

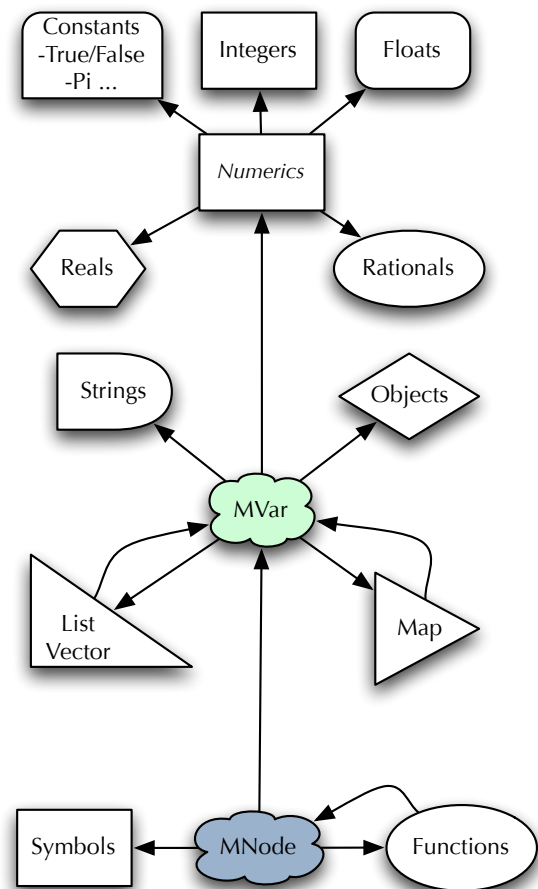
-The following is an example of an MVar with nested data using maps and vectors, represented in MML code:

```
[color:[0.306309,0.885713,0.0972619],  
p:[u:[-5.82034,2.44742,14.1256],  
v:[0.0239208,0.214343,-0.0160897]],  
[u:[-11.9655,6.41404,9.85531],  
v:[-0.0626363,-0.016249,0.403458]],  
senderId:372]
```

-**MNode** is the most complex type, encompassing mvar by adding symbols and recursively defined functions. MNode allows code to be constructed and executed dynamically. The following is an example MNode / M function:

```
Set(z,Add(Sub(x,1),Mul(2,y)))
```

Numeric operations (**MVar**, **MMath**, and elsewhere) involving disparate operand types yield an appropriate output type, tending to preserve precision/exactness. For example, arithmetic operations on integers and rationals (**MRational**) always produce integers or rationals as appropriate. However, when an operation involves a float, integer/rational operands are internally converted to floating point in order to produce a float output.



# Languages

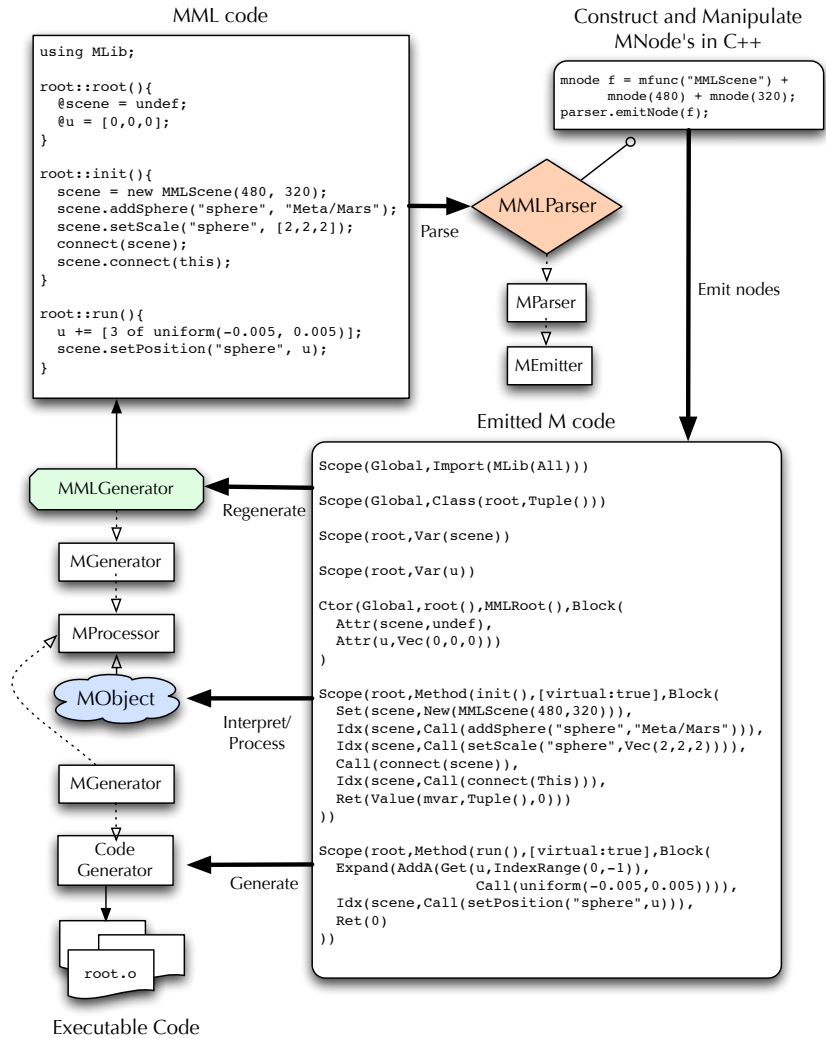
-The AndroMeta framework offers a unique approach to language design, allowing dynamic construction and interpretation of code, and translation to various target languages and code generators.

-Central to this system is AndroMeta's M language. Based on functional programming principles and a minimal syntax, M serves as the input that is both directly executable by interpreters, and at the same, the AST (abstract syntax tree) used for code generation (**MGenerator**, **MMLGenerator**, **MPLKernel**, etc.)

-Using this approach, AndroMeta provides a powerful language construction system which eases some of the difficulties typically associated with programming language design and implementation. AndroMeta provides the AST and semantics side such that a major part of the task of designing a language has already been completed. The remaining portion is to supply the front-end parser which constructs and emits M code in the form of MNode's. Parsers (**MParser**, **MMLParser**, etc.) conforming to the M code interface can then immediately take advantage of interpretation and translation to supported target languages and code generators.

-MML (Meta Modeling Language), specializing in modeling and simulation, and MPL (the "performance and parallelism" language are two such language derived from M and the AndroMeta language framework.

-Each **MObject** subclass is effectively a thread-safe M interpreter with complete state and scoping. Subclassing an MObject allows symbols/attributes and functions/methods to be dynamically defined and processed within.



## Task Concurrency

-AndroMeta features a powerful graph-based concurrent processing system via **MProc** and **MProcTask**.

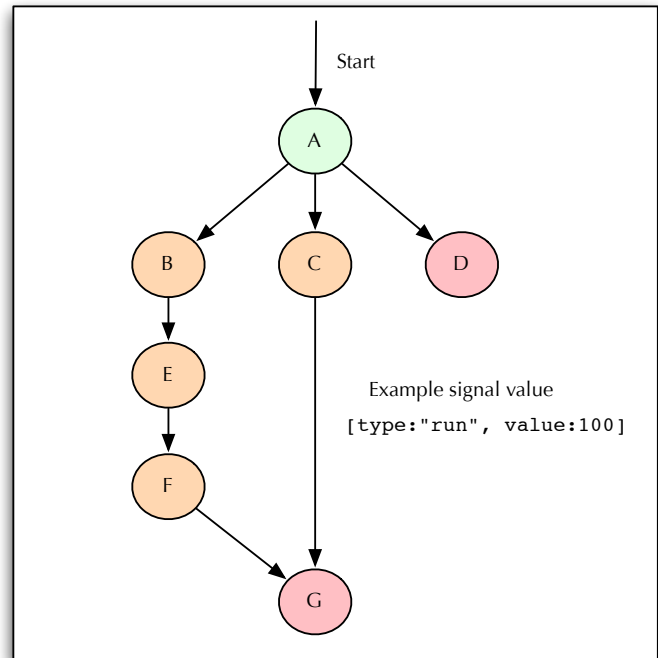
-MProc's are connected to each other to form a "network" or *task* where chaining order determines execution sequence whereby parallel execution paths may be formed or joined with other paths.

-The semantics of the MProc system are highly configurable allowing maximum application flexibility and general usability.

-MProc's signal each other with MVar's. The signals provide a mapping that the MProc uses to determine whether or not to activate when signaled and also allows state to be passed enabling the passage of data to be entirely localized, if desired, such that passing state through signals allows several instances of the MProc to be run unhindered in parallel.

-MProc's once activated, are queued to an associated MProcTask. MProcTask's coordinate a specific processing goal, providing high-level thread-pooling and queueing with optional application-specific priority. Tasks may be halted, cleared, resumed or waited upon for completion.

MProcTask

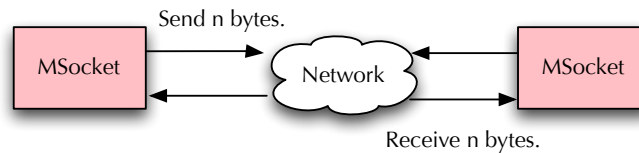


Above: A "network" or "task" of MProc's A, B, C, D, E, F, and G are connected as shown – all of which are configured such that they activate upon receiving signals from all of their incoming edges. To start this process, A is queued to the specified MProcTask. MProc A runs and at its completion, signals its outgoing edges, activating B, C, and D. Depending on the queue priority and the number of threads the task was set to run with B, C, and D may essentially run simultaneously, causing a new execution path from B to F to run in parallel to C and D. These parallel execution paths rejoin at G which is not activated until receiving a signal from both F and C. In this scenario, one would typically perform a blocking call to await completion on G.

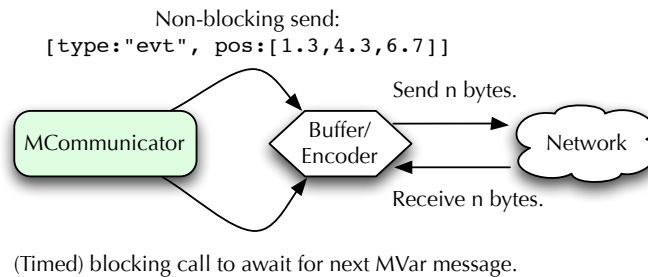
## Networking

AndroMeta supports networking at varying levels of complexity and abstraction allowing applications to choose whichever form is appropriate.

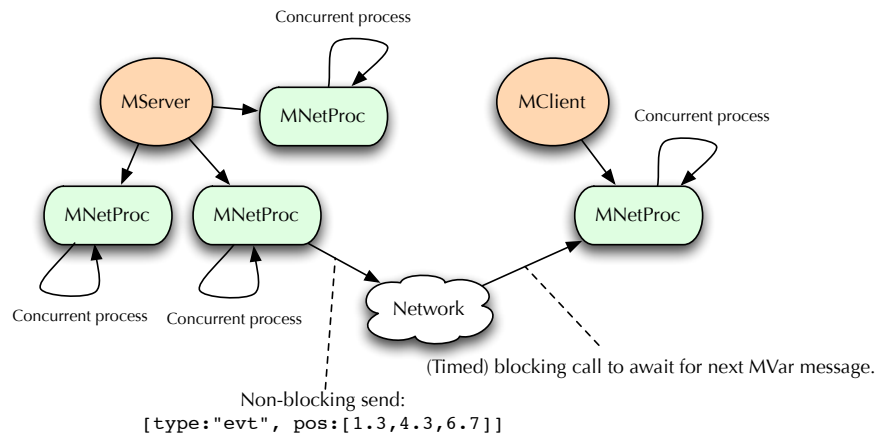
1. The most basic form of networking is accomplished with TCP/IP sockets sending and receiving binary data. **MSocket** provides a typical socket implementation.



2. **MCommunicator** provides an easy-to-use socket abstraction allowing arbitrarily complex data in the form of MVar's to be sent and received asynchronously. The communicator internally handles buffering, encoding/decoding and transmission in compressed binary form.



3. **MServer** and **MClient** provide a general-purpose means to implement networked client/server applications. Upon receiving or initiating a connection, an **MNetProc** is spawned. MNetProc's combine the functionality of MCommunicator and MProc to provide a flexible means to implement concurrent and (recurrent) request/response "threads".



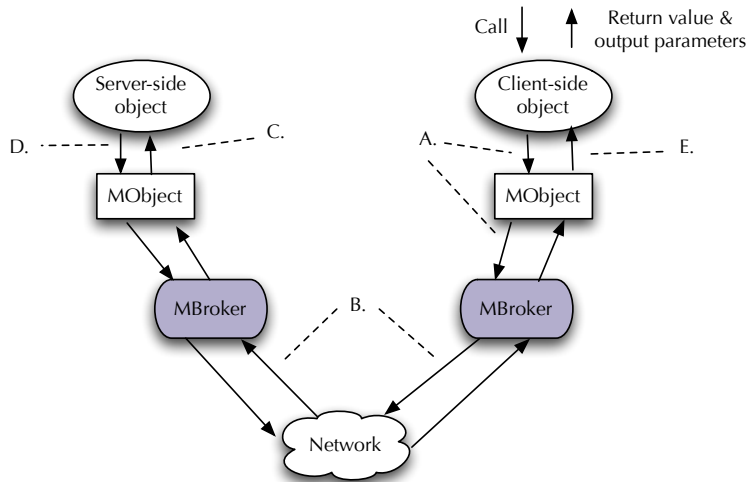
4. AndroMeta leverages its technical foundations as discussed so far to implement the **MBroker** – a distributed object broker which allows objects to be distributed and obtained across a TCP/IP network. Setting up a class for distributed objects in MML++ is as simple as subclassing MObject, and specifying the **mdist** return type (effectively an mvar). The client uses an MBroker to obtain a remote object that the server-side MBroker has distributed.

```
class DistributedObject : public MObject{
public:
    mdist increase(int delta){
        mutex_.lock();
        int count = sharedCount_ += delta;
        mutex_.unlock();
        return count;
    }

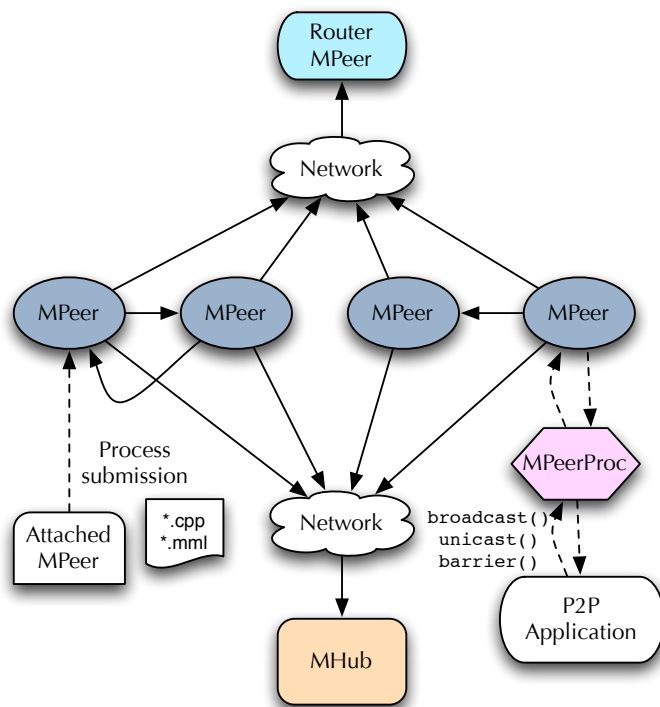
private:
    MMutex mutex_;
    int sharedCount_;
}
```

The following diagram details the sequence of events that occur under the hood when a remote call is performed. To start the process off, externally, `increase()` is called as if it were any ordinary C++ method, then:

- A. MObject's determines that this is a client-side object therefore it forwards the request to the MBroker.
- B. The client's MBroker communicates the request to the server-side MBroker. The server-side broker forwards the request to the appropriate object.
- C. The MObject dispatches the call to the `increase()` method.
- D. MObject's determines that this is a server-side distributed object. Proper locking must be performed in user code because the server allows multiple clients will call `increase()` simultaneously. `increase()` returns its value and the communication chain is followed back to the client.
- E. Finally, the client receives the return value. If the call included output reference/pointer parameters these would be passed back to the caller as well.



5. AndroMeta features a specialized peer to-peer system as provided by **MHub** and **MPeer** which may be run either over a local network to provide a “cluster” or over a wider area network such as the Internet. The hub provides the meeting place for peers whereby peers connect to a hub in order to receive updates about other connected peers who may wish to use their services. The hub is capable of connecting and updating a large number of peers as it is a low network-traffic agent which does not provide centralized messaging. For messaging between peers to be possible, they must either be directly reachable over the network in question or there must exist a **MPeer** that is reachable, connected to the hub which acts as a “router.”



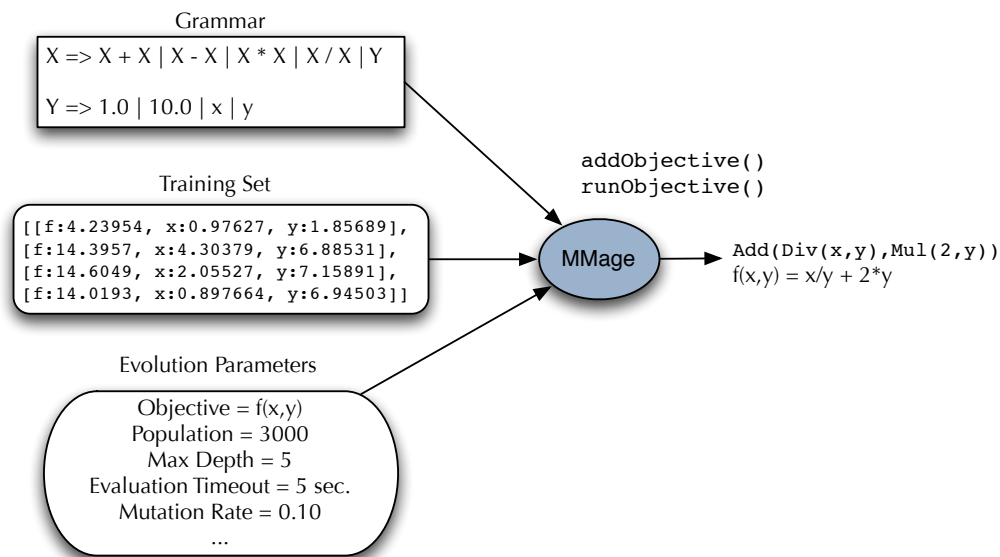
The peer-to-peer system is flexible enough to be used in a variety of scenarios. The most common scenario is to use the peer-to-peer system as a “cluster”, allowing users to submit a compute-intensive program/process to be run on a designated number of peers. The peer-to-peer application uses the **MPeerProc** class to communicate with the cluster. **MPeerProc** supports unicast, broadcast, and barrier synchronization operations, similar to MPI, but at a higher level of abstraction.

Another scenario is to use the peer-to-peer system in conjunction with an **MPeerBroker**. This is the most complex and powerful form of networking that AndroMeta supports whereby users perform calls on remote objects as distributed by other peers with communication being routed through the peer-to-peer system.

## Math, Machine Learning, and General AI

Machine learning and artificial intelligence in general are major drivers in AndroMeta's development. Building on AndroMeta's core foundation, the **MMage** class (Math and Genetic-algorithms Evolver) provides the functional backbone for much of AndroMeta's AI capabilities, adding sophisticated algorithms based on neural networks, genetic algorithms including genetic programming, and more.

Central to this effort, is AndroMeta's ability to evolve M code. Using supervised learning, MMage is supplied a training set of data, a grammar which determines the rules whereby the evolved code is generated, and a number of parameters which control the evolutionary process.



MMage also supports code evolution via unsupervised learning. Usually in this context, the code to be evolved is part of a larger M program. In each iteration, the program is passed the generated code “module”, which it executes in the context of a larger program. Here the program judges its performance in any way appropriate then passes back to the MMage a metric measuring the generated module's *error* which it seeks to minimize. The evolutionary process may be configured such that it halts upon achieving a desired target error or after a timeout is reached.

Another major facet of MMage is its higher-level mathematical facilities. For example, integrals, derivatives, and much more can be computed as well as a large number of operations which rely on symbolic manipulation such as reducing mathematical functions to simpler form. MMage relies on functional programming principles to pass and receive closures in the form of MNode's. For example, we can use the MMage to compute an integral by calling:

```
Integrate(x^2 + 1, x)
```

Which returns the MML-generated code containing:

$$x + 1/3 * x^3$$

## And Much More

AndroMeta includes a variety of features which we did not cover in this introduction, but which are described in the documentation that follows:

- A mechanism for storing/restoring objects to MVar.
- Precise/exact math routines.
- Extensive exception classes, exception handling, and exception safe code.
- Message buffer and message handling classes.
- For those who have purchased Wolfram Mathematica, the ability to interface MMage with Mathematica.
- Random number generation and probability distributions.
- The LLVM/Clang-based MML++ compiler which is fully compatible with C++ (in fact a complete superset of C++ with many C++ 11 features) and incorporates many of the features of the AndroMeta framework as first-class language features.
- Embedded languages such as MPL - the Meta “Performance” and “Parallelism” language for achieving high performance via the GPU (using OpenCL under the hood), CPU multiprocessing, and highly optimized code generation using LLVM.

## Thread Safety

Many of the framework classes were designed from the beginning to be used in the context of threading while finer-grained operations such as operations on types, for performance reasons, do not have thread-safety built in. Operations such as submitting a message to a **MMessageBuffer** (in conjunction with **MMessageHandler**) or queuing an MProc to a task are inherently thread-friendly – in fact all methods on MMessageBuffer and MProc are completely thread-safe so that users need not worry about wrapping calls to such classes with mutual exclusion constructs. Another aspect in which the framework is thread-friendly is with calls which are required to be executed on the main thread, e.g: GUI / OpenGL functions. The framework uses events to transparently defer, as needed, such calls to the main thread.

# MML (Meta Modeling Language)

## Overview

MML, AndroMeta's premier interpreted language, is based on the syntax of C++ and was designed specifically for modeling and simulation, but it has become useful in other contexts as well. MML emphasizes simplicity, code readability/clarity, and what would seem to run contrary to these goals: high performance (at least as much as possible for an interpreted language) and flexibility. It is built atop the foundation of AndroMeta's M language and language design system – that its the MML front-end parser converts MML code into M / AST code which is then directly interpretable by MObject subclasses. MML is not only a language, but a sub-framework that allows users to harness the power of the larger AndroMeta framework without necessarily having to code in C++.

MML is similar to C++ in syntax and in concept but departs from it in several ways. It offers a more concise syntax and is weakly-typed. Effectively, one of the consequences of this, is that types are omitted in specifying method/function parameters and in declaring variables as variables are and instance of an mnode or mvar. MML supports a variety of modeling paradigms including agent-based, discrete-event, and continuous or hybrid simulation. This is made possible by its flexible graph-based system, and event handling/communication model as provided by the MML entity base class. A variety of included *scenes* allow for easy 3D visualization, dynamic plotting, and the ability to link GUI controls to the emission of events.

Even if you are not interested in using MML – perhaps you would rather code in C++ or use the MML++ in conjunction with the AndroMeta framework, it is worth reading this section as MML is used to describe important foundations of AndroMeta which are pertinent regardless of which language you are using with AndroMeta. Also MML, like other languages that AndroMeta includes, can be embedded within a C++ or MML++ program – in this this sense, it is a type of meta language.

## Commands

One of the best ways to get acquainted with MML is through the `meta` command. The `meta` program uses an MML parser and MML interpreter to allow MML statements and expressions to be entered interactively or a project run from multiple source files. The source code can be viewed or modified to suite your tastes and is available in `$ANDROMETA_HOME/src/meta`. By now you should have AndroMeta properly installed on your system and `$ANDROMETA_HOME/bin` in your `$PATH`. Simply type `meta` and you are greeted with the `>>>` prompt. The return value of an entered expression or set of statements is echoed back, e.g:

```
>>> 1 + 1
2
```

Throughout this document we list examples and code snippets which may be tested this way.

`metac` is helpful utility command which uses a configuration file similar to that of a `Makefile` for running an MML program and associated arguments. `metac` scripts typically pass a number of files to the `meta` command. `metac` is useful for making changes to MML code and quickly testing the results. Most `metac.config` files, including those used by the examples, define a run command. So we typically type:





```
>>> x = [:30, 1, 2, (4,5,6)]
[:30, 1,2,(4,5,6)]
```

The difference between vectors and lists comes down to a matter of performance determined by how the data is intended to be used. For smaller data it doesn't make much of a difference, but for larger data sets the difference may be significant. In general, vectors are used when random read access is important and lists when a large number of insertions at locations other than the end are required. We refer to the list or vector portion of an mvar as its *sequence*. As a special case, the following syntax is used to represent a list containing one item: (60,).

An mvar's sequence is indexed with the [] operator. To extract the 5 in this sequence, we use:

```
>>> x[2][1]
5
```

We may also re-assign elements of the sequence this way:

```
>>> x[2][0] = 50
```

Keeping with the conventions of mainstream programming languages, 0 references the first element of the sequence.

## Maps

While holding a sequence, an mvar may at the same time, store elements referenced by a named string/key in its map. Let's add a key "a" which holds the value 40:

```
>>> x.a = 40
40
>>> x
[:30, 1,2,(4,5,6), a:40]
```

The x.a syntax is a short-hand for x["a"]. For keys which are not a lexical identifier, we use the full form:

```
>>> x["not an identifier"] = 50
50
>>> x
[:30, 1,2,(4,5,6), a:40, "not an identifier":50]
```

## Built-in mvar Methods

Normally, a method call on an mvar directs that call to the object that the mvar points to. However there are a number of built-in methods that operate directly on the mvar's data.

```
>>> x = [1,2,3]
[1,2,3]
>>> x.pushBack(4)
[1,2,3,4]
```

```
>>> x.push(5)
[1,2,3,4,5]
```

`push()` is a shorthand for `pushBack()` and inserts an mvar at the end of an mvar's list/vector. `pushFront()` adds an element at the beginning of its sequence.

```
>>> x.pushFront(5)
[5,1,2,3,4]
```

`popFront()` and `popBack()` remove the first or last element from the sequence and return its value. `pop()` is an alias for `popBack()`.

```
>>> x.pop()
4
>>> x.popFront()
5
>>> x
[1,2,3]
```

The `size()` method returns the size of an mvar's sequence (not including its map):

```
>>> x.size()
3
```

`hasKey()` checks whether an mvar has a specified index or key:

```
>>> x.hasKey(22)
false
>>> x.hasKey(0)
true
>>> x.hasKey("a")
false
>>> x.a = 30
30
>>> x.hasKey("a")
true
```

`keys()` returns a vector containing all of the mvar's map keys:

```
>>> x = [a:33, b:44, c:55]
[a:33, b:44, c:55]
>>> x.keys()
["a", "b", "c"]
```

`head()` extracts the value at the head of the mvar.

`x.isEmpty()` returns true if x does not hold any items in its map or sequence.

`x.isDefined()` returns true if the head of x is not equal to the undefined value `undef`.

`x.isString()` returns true if the head of x is a string and `isNumeric()` returns true if the head of x is numeric.

`x.clearSeq()` clears the sequence portion of `x`.

`x.clearMap()` clears the map portion of `x`.

`x.clear()` clears both the map and sequence of `x`.

`x.indexMap(true)` enables `x`'s map to be indexed as if it were part of its sequence.

`x.index(value)` returns the index position within `x`'s sequence for the first element matching `value`, or `-1` for no match:

```
>>> x = [5,6,7,8]
[5,6,7,8]
>>> x.index(2)
-1
>>> x.index(7)
2
```

`x.erase(k)` erases from `x`'s sequence or map, the index or key referred to by `k`.

`x.merge(y)` merges the map and sequence of `x` and `y`. With regards to their maps, in the case of key collisions, `x`'s values are preserved. As far as the sequence is concerned, `y` is appended to `x`:

```
>>> x = [1,2,3,a:23,b:45]
[1,2,3, a:23, b:45]
>>> y = [44,55,66,77,c:99,b:23]
[44,55,66,77, b:23, c:99]
>>> x.merge(y)
[1,2,3,44,55,66,77, a:23, b:45, c:99]
```

Set operations may be applied in a recursive fashion over an mvar with the following functions:

```
>>> x = [1,2,3,4,5,a:[1,2,3]]
[1,2,3,4,5, a:[1,2,3]]
>>> y = [3,4,5,6,7,a:[2,1,7]]
[3,4,5,6,7, a:[2,1,7]]
>>> x.unite(y)
(1,2,3,4,5,6,7, a:(1,2,3,7))
```

```
>>> x = [1,2,3,4,5,a:[1,2,3]]
[1,2,3,4,5, a:[1,2,3]]
>>> y = [3,4,5,6,7,a:[2,1,7]]
[3,4,5,6,7, a:[2,1,7]]
>>> x.intersect(y)
(3,4,5, a:(1,2))
```

```
>>> x = [1,2,3,4,5,a:[1,2,3]]
[1,2,3,4,5, a:[1,2,3]]
>>> y = [3,4,5,6,7,a:[2,1,7]]
[3,4,5,6,7, a:[2,1,7]]
>>> x.complement(y)
```

```
(1,2, a:(3,))
```

## Strings

An mvar may also be assigned to a string:

```
>>> x = "hello"
"hello"
```

The + operator is used for string concatenation:

```
>>> x = "hello"
"hello"
>>> y = " there"
" there"
>>> x + y
"hello there"
```

The value of another expression may be interpolated within a string with the `${...}` and `%{...}` tokens:

```
>>> id = 32
32
>>> name = "object#{id}"
"object32"

>>> value = 0.01234567901234568
>>> "value is: #{value}"
"value is: 0.0123457"
>>> "value is: ${value}"
"value is: 0.01234567901234568"
```

`%` gives a concise string representation of an interpolated value while `$` is used for a high-precision string representation.

Single-quoted strings are semantically identical to double-quoted strings and are useful when the string itself contains quotes:

```
>>> x = 'another string'
"another string"
```

Any MML expression may be interpolated within a string with `%{...}` or `${...}` and single-quoted strings are useful within this context:

```
>>> a = ["field one":22, "field two":33]
["field one":22, "field two":33]
>>> "first is: %{a['field one']}"
"first is: 22"
```

Otherwise double-quotes within a string need to be escaped:

```
>>> x = "a \"string\" within a string"
"a \"string\" within a string"
```

`x.split(delimiter)` splits a string of items separated by `delimiter` into a vector, and `x.join(delimiter)` performs the inverse:

```
>>> x = "a,b,c,d"
"a,b,c,d"
>>> y = x.split(",")
["a","b","c","d"]
>>> y.join(":")
"a:b:c:d"
```

## Libraries

We have now covered the most essential mvar data handling syntax and methods. More extensive functionality is provided by way of libraries and classes. A scope contains a list of symbols and functions which the caller has access to at its current point of execution. A library is a set of functions grouped together for a specific purpose which may be imported into a scope. `Mlib` is a general-purpose library that contains a number of functions useful in the context of MML. An MML program typically imports `Mlib` into the global scope at the top of one of its source files:

```
>>> using Mlib;
Tuple(abs(1), acos(1), arg(1), arg(2), argDefault(2), args(0), asin(1),
atan(1), atan2(2), bernoulli(1), binomial(2), bpascal(2), ceil(1),
chisquare(1), choice(1), clamp(3), cos(1), cosh(1), crossProduct(2),
degrees(1), distance(2), dotProduct(2), equilikely(2), erlang(2),
exponential(1), floor(1), formatTime(1), formatTime(2), fromStr(1),
fromStr(2), fromStr(3), geometric(1), hasArg(1), inf(0), listRange(1),
listRange(2), listRange(3), log10(1), lognormal(2), mergeSymbolicMaps(2),
mixedDistribution(1), nearInteger(2), negInf(0), normal(2), pareto(2), pi(0),
poisson(1), pow(2), print(1), randomSequence(2), repeatString(2), round(1),
round(2), scaleVector(2), setArg(2), setArgs(1), setDefaultPrecision(1),
sin(1), sinh(1), sleep(1), sqrt(1), srand(1), systemTime(0), tan(1), tanh(1),
toVar(1), toVar(2), uniform(0), uniform(2), vectorRange(1), vectorRange(2),
vectorRange(3))
```

A list of the imported functions is returned which may now be called from anywhere in the program. The number within the parentheses of each function gives the number of arguments each call takes. If we do not wish to import all of a library's functions, we may list them selectively. This is useful because importing all the calls could potentially cause name clashes such as if we desired to create a function of the same name. Suppose for this reason, we just wanted to import a few trigonometric functions, then we could use:

```
>>> using Mlib(cos,sin,tan);
Tuple(cos(1), sin(1), tan(1))
```

We now make a call to one of these functions with:

```
>>> cos(2.9)
-0.970958
```

Here we are calling the lower-precision (but faster) math calls that MLib provides. For the high-precision equivalent we would use:

```
>>> Cos(2.9r)
```

MObject.h provides a listing of the standard exact/high-precision/functional programming calls that are supported.

We do not provide a listing of the functions that MLib provides here, although some of MLib's key functions will be discussed in later sections. The reader is referred to the header documentation in MLib.h for a complete description of each call. These C++ header files, should be easy to understand for those unfamiliar with C++. Just treat each parameter type which is not an mvar as if it were. References (&) are used for efficiency's sake and non-const references are used for output parameters, i.e: the function called changes the value of parameter with a change that persists after the call is made.

## Classes

Libraries allow us to make a call to C++ code from MML, however, more powerful functionality is provided via classes. Classes deal not only with isolated calls but a series of calls to an object which has a persistent state. One example of this is a random number generator whose state consists of a random number stream as determined by an initial seed. Each method call takes the next random number from the stream. All MML-accessible classes are derived from MObject which means their methods may be called dynamically from an M interpreter (i.e: MObject subclass). Importing a class into MML is easy whereby an mvar is made to hold a pointer to an MObject. Let's create an MRandom object with a seed of 10:

```
>>> r = new MRandom(10)
0x320bae0
```

Note that we did not have to specify an include or using directive at the top-level – external class names are named uniquely, allowing classes to be imported whenever they are instantiated. Here we make calls on the instantiated object to obtain a uniformly distributed random numbers in the range of [-1,1] :

```
>>> r.uniform(-1, 1)
0.542641
```

## User-Defined Classes

So far we have limited our discussion of MML to entering statements at the top-level, however a complete MML program is built through the definition of classes. The MMLEntity class is the essential building block and base class for user-constructed classes. Defining a class in MML is simple — we simply begin creating its methods. Typically a constructor is specified first, and like C++, Java, and similar languages, it is named the same as the class name. The following code creates an entity named Object by declaring its constructor:

```
Object::Object(){
}
}
```

A constructor, like any other method, may take parameters:

```
Object::Object(s){  
  
}
```

A constructor is typically used to declare an object's attribute variables. Attribute variables are attached to the object and make up its state. Unlike the other types of variables we have used thus far, attributes persist after the constructor is called. Attributes are referred to with the @ operator. Here we declare an attribute `size` which takes the value of the passed `size`:

```
Object::Object(size){  
    @size = size;  
}
```

Attributes can be defined in any method, but it is good practice to declare them up front in the constructor.

Let's define another method on `object` that operates on the `size` attribute:

```
Object::triple(){  
    size *= 3;  
}
```

Then, to instantiate an `object` and perform this call on it, we use:

```
>>> o = new Object(20);  
>>> o.triple();
```

Suppose in our `triple()` method we needed to use a value which is calculated locally as opposed to being a permanent attribute attached to the object. Locals are variables which are defined relative to their enclosing `{ ... }` scope/block and exist temporarily throughout the duration of the execution of that block. We use the `= (or =:)` operator to simultaneously declare a local and set its initial value. Revising `triple()` we have:

```
Object::triple(){  
    factor = 1 + 1 + 1;  
    size *= factor;  
}
```

## Extending Classes

Similar to C++ and other mainstream object-oriented languages, MML supports the ability to extend classes using a familiar syntax. We say that a rectangle *is a* shape or shape is the *base class* of rectangle. Base classes are specified in an entity's constructor. In MML, all attributes are *protected* which means they are private and may not be accessed directly from outside of the class but are made available for direct access to the the classes which extend them. All methods are virtual/polymorphic meaning that they may be redefined or *specialized* by derived/extended classes so that when we call a method on a shape that is actually a rectangle, the rectangle's specialized version of the method gets called. Unless otherwise specified, the base class of an entity is `MMLEntity`, except for the root entity which derives from `MMLRoot`. The following code illustrates a basic usage of these concepts:

```

Shape::Shape(){
}

Shape::area(){
}

Rectangle::Rectangle(width, height) : Shape{
    @width = width;
    @height = height;
}

Rectangle::area(){
    return width * height;
}

Square::Square(side) : Rectangle(side, side){
}

Circle::Circle(radius) : Shape{
    @radius = radius;
}

Circle::area(){
    return 2*pi()*radius;
}

shapes = [];
shapes.pushBack(new Rectangle(10,20));
shapes.pushBack(new Circle(23.4));
shapes.pushBack(new Square(30));

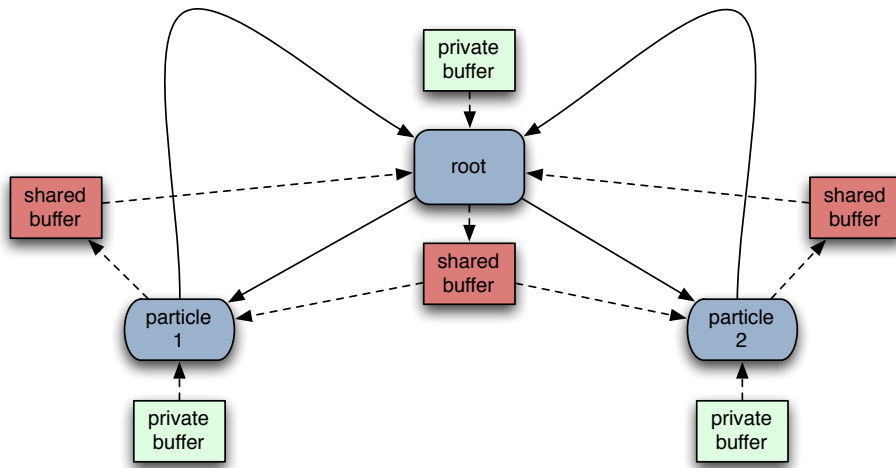
foreach(s, shapes){
    Print("area is: %x", s.area());
}

```

We can use the `super` keyword in an extended class to call the base class version of a call, e.g: `super::area()`. By extending classes this way, we can easily build up a hierarchy of reusable classes.

## Structure of an MML Program

MML programs have a special entity *root* which is responsible for the creation and initialization of all other entities. An MML program has exactly one root entity. Besides program setup, the root helps coordinate entities in various ways and provides an interface to AndroMeta framework facilities such as optimization. These features will be discussed in later sections. Entities are connected to one another in order to define their execution order and event communication and event handling properties. These connections form a graph, or in simpler cases a tree. A typical MML program structure is depicted below:



We will describe the meaning of the diagram above momentarily, after we discuss how the root entity is created and initialized. The root entity has a constructor with no parameters and must not create any entities in its constructor. The `init()` method is provided for this purpose, which is called automatically at the start of the program. Entities other than `root` may use `init()` as well, however `init()` is called automatically only on `root`. Most other entities are able to perform all of their initialization in their constructors. The structure of the above diagram is achieved in code with:

```

Root::Root(){
    @p1 = undef;
    @p2 = undef;
}

Root::init(){
    p1 = new Particle();
    connect(p1);
    p1.connect(this);
    p2 = new Particle();
    connect(p2);
    p2.connect(this);
}

```

`MMLEntity` is completely configurable in C++ but its default behavior upon creation is to create a private and shared event buffer for itself. The private buffer is used for events directly solely to the entity and the shared buffer is used for sending events to its outgoing connections.

The most essential code is provided by the `run()` method:

```

Particle::run(){
    // define behavior here
}

```

Here an entity defines its application-specific behavior which, generally speaking, is accomplished by modifying its attributes, responding to events, sending events, etc.

After an entity runs, the default behavior signals all of its outgoing connections/edges. Entities are activated or queued to run upon receiving a signal from all of their incoming edges. Thus the cycle back to root from the two particles, allows the program to run in a loop. The outgoing edges shown as solid arrows from root to the two particles form two parallel execution paths, and when running on a multiprocessor machine with threading enabled, these two entities would run in parallel. Threading is enabled by the `mThread` argument which is described in the Appendix.

## Events and Communication

An *event* is simply an `mvar` with an application-specific structure which gets scheduled for delivery to an entity after a certain delay. Each entity by default *subscribes* to its private buffer and all of the shared buffers from entities that comprise its incoming connections. To send an event to a specific entity's private buffer, the `send()` method is used as follows:

```
send(root, [position:[1,2,3], mass:23.4], 30);
```

Here an event is sent directly to the root entity to be delivered after 30 time units. Time is an abstract application-specific concept: each entity has its own clock and by default advances time by 1.0 units after its run cycle. So this would mean the event above gets queued in root's private buffer to be delivered after 30 run cycles. Delay in the event communication calls is optional and defaults to 0.

The most common communication need is to relay an event to an entity's outgoing connected entities, i.e: to send to its shared buffer. This is accomplished with the following, e.g:

```
relay([position:[1,2,3], mass:23.4], 30);
```

Going back to the diagram, if `root` performed this call, it would effectively send the event to `particle1` and `particle2`.

Sometimes an entity wishes to *post* an event to an entity's shared buffer when it is not connected in such a way that would normally permit it to do so. This is made possible with the `post()` call, specifying the target entity that owns the shared buffer as a first parameter:

```
post(root, [position:[1,2,3], mass:23.4], 30);
```

In some cases it is useful to subscribe to an entity's shared buffer without becoming connected to it. This is made possible with the `listen()` method:

```
listen(root);
```

## Event Handling

Events become queued for delivery in shared and private buffers. `handleEvents()` is called to begin cycling through the events delivered when the subscribing entity's current time is equal to or exceeds the

event's delivery time. `handleEvents()` may be called explicitly in `run()`, if it is not, then it is called implicitly after `run()`. `handleEvents()` iterates through the pending events, calling `handleEvent()` on each. `handleEvent()` is a virtual method defined in `MMLEntity` so its signature must match that which is defined in `MMLEntity.h`. Now is a good time to point out that there are several such virtual methods listed in `MMLEntity.h` that may be redefined in this way. For `handleEvent()` we use:

```
Particle::handleEvent(evt){
    // do something with evt
}
```

## Locality and Interactions

MML locality allows such things as collision detection or other locality-based interactions to be easily accomplished in MML in a high-performance manner. An entity opts to have one of its attributes, such as position tracked with radius 0.12 by the specified tracker entity, `root` in this case:

```
addLocality("position", 0.12, root);
```

When another entity  $O$  that has opted to track its attribute  $A$  on `root` enters within a radius of  $R$  of the first entity  $E$ ,  $E$  is sent an event `evt`:

```
evt.type = "locality"
evt.entity = <entity O>
evt.attribute = <attribute A>
evt.radius = <radius R>
```

The locality attribute may be a scalar or n-dimensional numerical sequence value.

The two parameter version of `addLocality`, equivalent to the above, is:

```
addLocality("position", 0.12);
```

and defaults to tracking on `root`.

`examples/locality` is a simple example to demonstrate how MML locality is used.

## Disconnections and Memory Management

The graph-based nature of an MML program also helps us manage the memory deallocation of entities. An entity is said to be reaped or *reclaimed* when it no longer has any incoming connections. The `detach()` method is used to detach an entity from its incoming edges, effectively having `root` reclaim the memory allocated for it. This can have a rippling effect: when an entity is reclaimed it implicitly has `disconnect()` called on it to sever its outgoing connections which in turn may cause other entities to be reclaimed. Therefore managing the entity graph and memory deallocation are combined and made easy – just call `detach()` on the entity to be deleted.

## More on MMLEntity

An entity contains a *tag* which is simply an attached `mvar`. By default the tag is set to the entity's unique integer id but may be changed with `setTag()`. An entity's *name* is, by default, its tag converted to a string, retrieved with `name()`. An entity's name need not be globally unique to the program but it is often helpful when this is the case. The name is used for recursive operations such as `find()` and MML scenes, discussed later, require a unique name to be associated with graphical objects, which is most conveniently, an associated entity's name. The name is also used for logging with `log()` which prints out the calling entity's time and name, e.g:

```
>>> log("finished");
23.0:root: finished
```

## Graph-based Operations

A typical task an entity is faced with upon creation is to find an object/entity that another entity has already created in order to make calls on it. For example `root` has created a scene object and we need to obtain a reference to it in order to perform visualization:

```
Particle::Particle(){
    scene = root.find("scene");
}
```

The above call searches `root`'s outgoing connections for an entity matching the name "scene". To make this call recursive, we pass `true` as a second parameter, and `false` as a third parameter to not include `root` in the search. The results returned are actually a list of matches with the head of the `mvar` containing the first match.

A similar operation is to traverse the graph and perform a call on each object. For this we use the `execute()` method:

```
Particle::run(){
    time = 51.2;
    root.execute({stop(time)});
}
```

Similar to `execute()`, `query()` performs a call on each object but gathers the results in a vector:

```
Particle::run(){
    times = root.query({getTime()});
}
```

The braces syntax denotes that the call to `stop()` or `getTime()` is a *closure* – the function's execution is delayed until it reaches the object. The parameters of each of the above calls are similar and the reader is referred to the `MMLEntity` header for a complete description of each.

## Syntax

### Overview

We have already covered the most important syntax for handling data, making calls, defining classes, etc. Many of the areas of syntax yet to be covered are based on C++ and are best illustrated by the examples, however we will mention them briefly here. We also discuss in this section syntactical constructs unique to MML.

## Loops

A while loop repeatedly loops through a block so long as its condition is true:

```
x = 0.1;
while(x < 2){
  y = 0.01;
  log("%{x}");
  x += y;
}
```

Note that in the above we have defined a local `y` within the loop's block. Each iteration through the loop creates a new `y` and sets its value to `0.01`.

Any for loop can be easily expressed in terms of a while loop but for loops provide a convenient shorthand. The for loop equivalent is:

```
for(x = 0.1; x < 2; x += 0.01){
  log("%{x}");
}
```

Typically a for loop is used to iterate through a sequence:

```
x = [10,20,30,40];
for(i = 0; i < x.size(); ++i){
  xi = x[i];
  log("%{xi}");
}
```

MML provides a further short-hand for the above:

```
x = [10,20,30,40];
foreach(xi, x){
  log("%{xi}");
}
```

## Conditionals

MML's conditional statements are identical to those of C++ and C, e.g:

```
if(x > 10){
  log("x is large");
}
else if(x > 5){
  log("x is medium");
}
```

```
else{
  log("x is small");
}
```

## Vectors

MML features some concise vector handling methods. Suppose we have a vector which is defined as:

```
v = [1,2,3];
```

We can apply an operation over all elements of the vector with:

```
v += 10.0;
```

We may simultaneously assign the values of multiple values using vector list notation, e.g:

```
[a,b,c] = [3 of uniform(-1,1)];
```

As the above hints at, we could produce the vector:

```
[0,0,0]
```

with:

```
[3 of 0].
```

Another useful shorthand is the following. Suppose we wanted to produce a vector containing the first 10 integers squared. Then we could use:

```
[i^2 for i in 1..10]
```

## Default Parameter Values

When defining a method we can specify a default value for a parameter and when doing so, must then specify default values for all subsequent parameters, e.g:

```
object::setVec(x,y,z=0,s=1){
  u = s*[x,y,z];
}
```

The above method may now be called with:

```
o.setVec(1,2);
o.setVec(1,2,3);
o.setVec(1,2,3,5.6);
```

## Named Arguments

Suppose, we wished to perform a call to a function (caveat: it must be user-defined and not a C++ defined call) by naming the arguments rather than specifying them based on position. Then we could use the following equivalent for the last example:

```
o.setVec(s:5.6, z:3, y:2, x:1);
```

## Comments

Using C++ syntax, multi-line and single-line comments may be placed anywhere in the code:

```
/*  
  a  
  multi-line  
  comment  
*/  
  
// increment x  
x++;
```

## MProgram Arguments

### Overview

MML does not allow globals to be defined, instead, it provides a thread-friendly alternative by way of MProgram arguments. MProgram arguments can be thought of as an mvar whose data has been protected for shared access. Arbitrarily complex data may be defined as arguments which are globally accessible to all objects. When an MProgram starts up, it merges arguments from a config file with those passed by way of the command-line. Command-line keys replace those previously defined in the config files. By default an MProgram uses its "name" argument (defaulting to process name) to look for a <name>.config first in the current working directory then in \$ANDROMETA\_HOME/config. The interested reader should take a look at some of the config files in this directory to see which arguments they use and how they are structured. Command-line arguments are passed to a command like this:

```
./program -n 100 -code="1+1"
```

The difference between the first and second argument is that the first is parsed as an mvar value, whereas the second, using the = syntax, is parsed as a complete MML code section. In most cases you will want to use the former notation. MLib provides an interface to MProgram arguments which may be retrieved and manipulated with:

```
// get a copy of the n argument  
arg("n");  
  
// set the value of n  
setArg("n", 100.0)  
  
// provide a default value of 50 in case n was not  
// specified in the config or command-line  
argDefault("n", 50)
```

Please see MLib.h for further details.

The appendix section “MProgram Framework Arguments” gives a detailed list of MProgram arguments that are used to control various framework parameters, some of which are useful in the context of MML.

## Scenes

### Overview

MML features a number of *scenes*: visual environments containing graphical objects/actors which may be manipulated by an easy-to-use MML call interface. Scenes are used to perform visualization, often in a three dimensional setting. Depending on the scene, various aspects of the environment, rendering details, position of objects, animation state of actors, lighting, and more, may be controlled. Most scenes deal with persistent graphical objects and are usually linked to an MML entity so that the graphical properties of the object may be changed, e.g: repositioned, reoriented, scaled, or adjustments made to color properties, as the corresponding MML entity state changes. The current AndroMeta distribution contains a number of scenes: `MMLScene`, `MMLGraphScene`, `MMLSpaceScene`, and `MMLRobotScene` for Linux, and `MMLScene` for Mac. The reader is referred to the examples and header files for a more thorough description of each.

### Plotting

The Mac release of AndroMeta includes a class `MMLPlot` for performing dynamic two dimensional line plotting. A plot may contain multiple data sets plotted simultaneously. Plots are updated and displayed dynamically as the MML program progresses and various parameters control plotting behavior such as how often the plots get updated to the scene window. The Mac MML examples provide numerous instances of this usage and `MMLPlot.h` documents the exact call interface. If there is an interest in bringing back a similar plotting system for Linux, please contact us.

### GUI Controls

Similar to plotting, all scenes have built-in GUI control handling which is linked to the sending of MML events. Currently sliders and push buttons are available and may be added with `addButton()` and `addSlider()`. `MMLSceneBase.h` lists the exact parameters to these calls. When a slider is adjusted it emits/posts an event to the scene’s shared buffer. The event is structured like:

```
[type:"slider", name:"x", value:55.6]
```

Where name is as given to `addSlider()`, and value represents the slider’s current value.

Similarly, buttons when clicked emit an event whose structure is:

```
[type:"button", name:"reset"]
```

### MPEG Capture

Computationally complex scenes that do not render well in real-time may be captured to MPEG for fluid playback. Please see `MVideoEncoder.h` for how this is accomplished.

### Delay and Pausing

Scenes have the ability to pause for a specified delay during each run cycle via `setDelay()` which is typically called once upon instantiating the scene. This feature is useful for the visualization of programs which would ordinarily run too fast to be effectively observed. Similarly, the tab key can be pressed when the scene has focus to halt or resume execution of the program.

## Advanced Features

### Threading

It's easy to enable threading when running an MML program, but in some cases a few extra measures must be taken to make a program thread-safe. MProgram arguments are used in place of globals which have built-in protection for shared data in the presence of threading. An entity's attributes are always private/protected but this data can be modified and read externally by method calls which causes a potential threading concern. An entity has a built-in mutex/lock that can be used to protect its shared data. If we have a method that can potentially be called by multiple entities running at the same time, we simply lock that entity before calling its methods, and unlock it afterwards. This approach is also useful for ensuring a transaction or series of calls that must be executed all at once by a calling thread, e.g:

```
a.lock()
a.setColor([0.1,0.2,0.3]);
a.draw();
a.unlock();
```

If you don't intend to use threading in your program, then don't bother with locking, as it complicates the code somewhat and introduces a small performance penalty.

### Closures

A *closure* is a section of code whose execution is delayed. In MML, a function whose name begins with an upper-cased value entails different semantics than calling a function or method which does not begin in uppercase. The former does not have its call parameters evaluated before being passed while the latter always does. This makes it possible, for instance, to call a function to compute a derivative, e.g:

```
dx = Derivative(x^2, x);
```

For the latter, when call parameters would otherwise be evaluated, we can enclose them in braces to create an explicit closure, e.g:

```
sum({x + y}, 2);

>>> f = {x + y}
{x + y}
```

A closure itself is an expression, although the code it contains may be any valid segment of MML code. Closures are used throughout various calls to the framework and often appear embedded within an mvar, e.g:

```
data = [x:2,
        y:5,
        code:{
```

```

        if(x < 2){
            ++y;
        }
        else{
            --y;
        }
    }
}
]

```

A closure is evaluated with the `eval()` method, e.g:

```

>>> x = 2
2
>>> y = 3
3
>>> f = {x*y}
{x * y}
>>> eval(f)
6
>>> x = 4
4
>>> eval(f)
12

```

Similarly, a symbol can be bound to an unevaluated expression with the `=` operator.

```

>>> z = {a + b};
a + b
>>> a = 2;
2
>>> b = 3;
3
>>> z
5
>>> a = 10
10
>>> z
13

```

The difference here is that `z` is processed automatically whenever it is used in an expression. Functions may also be declared, using the `=` operator, e.g:

```

>>> f(x,y) = {x + y}

```

## Data Files

MML provides an extension to its data definition syntax that has yet to be discussed which makes it better suited for defining large data sets and data files. We include a section of data definition code to illustrate this:

```

[ // begin data
  dimensions: [50,50],
  locality: {[

```

```

        cell.pos,
        cell.pos + [0,1],
        cell.pos + [1,0],
        cell.pos + [0,-1],
        cell.pos + [-1,0],
    ]}
,
cells:
[ // begin cells
  [ // begin cell
    pos: [0,0],
    vegetation: 0
  ] // end cell
,
  [ // begin cell
    pos: [0,1],
    vegetation: 1
  ] // end cell
] // end cells
,
agents:
[ // begin agents
  [ // begin agent
    pos: [0,4],
    type: "herbivore",
    energy: 0.27135
  ] // end agent
,
  [ // begin agent
    pos: [0,20],
    type: "carnivore",
    energy: 0.582704
  ] // end agent
] // end agents
] // end data
;
```

As shown in the above, an arbitrary comment may be attached to each opening or closing bracket, or both. However, unlike ordinary comments, these labels are retained within the mvar data so that when the mvar is written back out, the labels are included. Without labels, largely nested data definitions can easily become unwieldy, as it is easy to lose track of the context of each opening and closing bracket. This syntax makes a large data file easier to read and modify, much like an XML file.

## Modeling Paradigms

MML supports multiple modeling and simulation paradigms including agent-based, discrete-event systems, continuous and hybrid simulation with its flexible time control mechanism, versatile event communication/handling, and object-oriented features. MML has access to an extensive library of stochastic functions.

## MML and the Larger Framework

MML may be fully utilized as a standalone programming language, however it can be quite powerful when used in conjunction with C++. Most of the MML-prefixed classes use virtual methods extensively which means the behavior of classes such as `MMLEntity` can be thoroughly customized. MML code is parsed to M code which means MML can be integrated with a C++ program where M code can be manipulated, interpreted, or regenerated back to MML.

# MML++

AndroMeta 2.0 introduces several new important features including: the MML++ language and compiler, the embeddable MPL language and sub-framework for high performance computing, and the MMage (Math and Genetic-algorithms Evolver). In addition, much of the existing AndroMeta code base has been improved including several fixes, performance optimizations, and design refinements and simplifications. We group these new additions to AndroMeta 2.0 under the umbrella term MML++ because many of them are best used in conjunction with the MML++ language rather than C++. In addition, some of these new features require the use of the MML++ compiler and cannot be utilized from other C++ compilers.

MML++ is a powerful, self-contained, and complete C++ compiler based on the LLVM / Clang compiler infrastructure. The MML++ compiler can be found at: `$ANDROMETA_HOME/bin/mml++`. Its name is derived from both MML and C++ because it can be thought of a sort of C++ with various language extensions, some inspired by MML, others unique in their own right. On the other hand, MML++ is not an extension of MML, but MML++ *is* a complete superset of C++. MML++ is capable of compiling virtually any C++ program and incorporates several of the newest C++ 11 features. MML++ extends C++ to provide specialized functionality which interoperates with the AndroMeta framework.

The following is a brief description of the major features that MML++ offers.

For subclasses of MObject, it:

- automatically allows for distributed objects using MBroker with a minimal amount of extra configuration.
- automatically allows for instantiation of these objects from an M-based interpreter.
- automatically generates `store()` methods and restore constructors so they can be saved to disk, or transmitted across a network, for instance.
- automatically allows methods of MObject subclasses to have their methods and functions called by M-based interpreters.
- automatically cast to and from appropriate MObject types.
- automatically allows attribute variables to be accessible from M-based interpreters.

In addition, MML++ makes extensive use of embedded meta-languages. For example, you might have an embedded language section written in MPL that specifies an n-body kernel that runs on the GPU. These embedded languages make use of the `mnode` type to provide a common basis of code representation and compilation. MML++ also allows `mvar` notation for concise vector handling or recursively nested `mvar` data definitions. In the following sections we introduce the core MML++ features and describe how embedded languages are used in conjunction with MML++

## A First Program in MML++

In the following sections we assume at least a basic familiarity with C++. Most people coming from a C++ background should find MML++ intuitive and should be able to get up to speed with the language in no

time. MML++ is perhaps best learned by example, therefore we introduce the major features of the language with MML++ examples included the AndroMeta distribution.

The following “Lucky Numbers” program demonstrates a minimal but complete MML++ program:

```
#include <AndroMeta/MLib.h>

using namespace std;
using namespace Meta;
using namespace MFunc;

int main(int argc, char** argv){
    MProgram program(argc, argv);

    mvar args = program.args();

    cout << "seed is: " << args.seed << endl;

    MLib::srand(args.seed);

    mvar numbers = [5 of equilikely(1, 100)];

    cout << "numbers are: " << numbers << endl;

    return 0;
}
```

To compile the program, we invoke the MML++ compiler with the following::

```
$ mml++ -o lucky-numbers main.mcopp
```

Note that mml++ takes the standard g++ / clang++ options and flags. We can then run the program with:

```
$ ./lucky-numbers -seed 579
seed is: 579
numbers are: [42,53,92,70,34]
```

This is a fairly basic example, but there are a few important things worth describing. Note that we named the source file with the designated .mcopp extension rather than the typical .cpp, .c, etc. extension. The .mcopp extension instructs the compiler that this file is using MML++ features and therefore should have certain parsing and transformations done on it that a regular C or C++ file would not. We can use mml++ as a standard C++ compiler by passing source files with the .cpp, .C, .c, etc. extensions to it. For MML++ header files the .mh extension is used. Additionally, MML++ can handle Objective C++ with MML++ intermixed using the .mmm file extension.

MML++ programs always link to the AndroMeta framework, so the MML++ compiler performs this linking automatically and also automatically specifies the proper header search paths for the AndroMeta framework. If you are having problems running this example, please revisit the “Getting Started” section. We still have to explicitly specify that we are using the AndroMeta namespace named Meta. The minimal headers that are required for MML++ to function properly are included automatically as well, but we did have to specify the include for the MLib utility function library. All AndroMeta framework components reside under the Meta namespace, but some items may be additionally nested under further namespaces such as we have specified using namespace MFunc in order to get the equilikely() function.

MML++ programs should always initialize an MProgram object near the start of the program, otherwise this can result in errors depending on which features of the framework are being used. Plus, we get some

value-added functionality like parsing the arguments vector. Note that command-line arguments are automatically coerced from strings into the proper mvar types. Then, we can get all arguments with the call to `program.args()`. MVar's are handled as a special case in MML++ so we have some specialized shorthand syntax such as `args.seed` for referring to fields in the mvar even though these are not statically defined fields on mvar. The last bit before returning demonstrates how we can send an mvar value to an output stream which is then conveniently outputted back in MML format, thus:

```
cout << "numbers are: " << numbers << endl;
```

produces the vector notation:

```
numbers are: [42,53,92,70,34].
```

This can be especially convenient for debugging MML++ programs which make extensive use of mvar values. The notation `[5 of equilikely(1, 100)]` provides a convenient vector handling shorthand which as shown, gets evaluated to a vector containing 5 distinct instances of the right-hand expression at compile-time. Another important feature of MML++ is that the square-bracket notation can be used to specify an arbitrarily complex mvar specification, e.g:

```
mvar x = [1,2,3];  
mvar y = [a:1, b:2, 4,5,6, [10,11,12]];
```

Please refer to the MML section for more information on mvar data specifications. This concludes our first basic example of MML++.

## Subclassing MObject

Subclassing an MObject (or descendant of MObject such as MMage) allows the MML++ compiler to automatically generate code that allows for the ability to call that object's methods in M-based languages such as MML, to allow for instantiation of that object in M (using `New()`, `Create()`, etc.), to allow the object to be stored to mvar then restored, to allow mvar attributes to be accessible in M, and more. This example, illustrates the power of these features along with a simple example of how MML++ handles embedded languages, or alternatively, how M code can be programmatically generated at runtime.

```
#include <AndroMeta/MMLParser.h>  
  
using namespace std;  
using namespace Meta;  
  
class Object : public MObject{  
public:  
    Object()  
        : c(1){  
  
    }  
  
    int foo(int a, int b){  
        return c*(a + b);  
    }  
  
    void bar(int v){  
        c = v;  
    }  
  
private:  
    int c;
```

```

    mvar d;
};

int main(int argc, char** argv){
    MProgram program(argc, argv);

    Object object;

    mnode n = mml{
        bar(3);
        d = 99;
    };

    object.process(n);

    mnode f = mfunc("foo") + mnode(1) + mnode(2);

    mnode x = object.process(f);

    cout << "x is: " << x << endl;

    mvar v;
    object.store(v);

    Object object2(v, Object::Restore);

    int y = object2.foo(1, 2);

    cout << "y is: " << y << endl;

    cout << "d is: " << object2.process( ml{d} ) << endl;

    return 0;
}

```

Even though we have not yet described the syntax, you might have guessed the output from running this program:

```

x is: 9
y is: 9
d is: 99

```

In the first section, we use an embedded language, MML, in this case:

```

mnode n = mml{
    bar(3);
    d = 99;
};

```

Here we have instructed mml++ to use the MMLParser at compile-time to generate an MNode (M code) which corresponds to the code in between the braces. In MML++, we can use the following embedded language designators:

```

mml{ ... } // mml language code
mml_m{ ... } // mml language code with automatic code simplification

mpl{ ... } // mpl language code
mpl_m{ ... } // mpl language code with automatic code simplification

```

```

ml{ ... } // ml language code
ml_m{ ... } // ml language code with automatic code simplification

```

We do something similar to create the mnode `f` in the lines that follow, except that the approaches differ in that we are constructing M code programmatically at runtime versus compile-time. In both cases we call the object's `process()` method to execute the code and return a result. In the last section, we use the `store()` method to pack the object's state into an mvar, then we use the *restore constructor* to create a copy of the object and verify that its state has remained intact.

We also declared an attribute variable `d` which has been exposed to the interpreter. Note that for this to be possible, the mvar type must be used. In order for methods or functions to be callable from M, the return values and parameters must be compatible with mvar / mnode – that is, they must be a built-in numeric type, string, vector, list, MObject pointer, or any of the values that mvar / mnode can be constructed with. For more details, please refer to the MVar.h and MNode.h header files.

## Distributed Objects

Another major benefit of subclassing an MObject is the ability to easily create distributed objects. The MML++ compiler automatically takes care of a lot of the details such that the object can be distributed on the server side or obtained on the client side using the distributed objects broker class MBroker. The underlying implementation uses a custom-built distributed objects implementation that has been tuned to handle a large variety of scenarios in a robust and high performance manner. The following example illustrates just how easy it is to set up a simple application that uses distributed objects. Here we create a simple server that runs in a separate thread, then we use the main thread to obtain the object and call some a method on the distributed object:

```

#include <iostream>
#include <cassert>

#include <AndroMeta/MEncoder.h>
#include <AndroMeta/MBroker.h>
#include <AndroMeta/MBroker.h>
#include <AndroMeta/MProcTask.h>
#include <AndroMeta/MThread.h>

using namespace std;
using namespace Meta;

class Server : public MObject{
public:
    Server(){
        a = 100;
    }

    mdist inc(){
        return a += 5;
    }

private:
    int a;
};

class ServerThread : public MThread{
public:
    ServerThread(){

```

```

}

void run(){
    Server* server = new Server;

    MProcTask task(10);
    MBroker broker(&task, 0);

    broker.distribute(server, "Server", "server");
    broker.listen(5125);
    task.run();
}

};

int main(int argc, char** argv){
    MProgram program(argc, argv);

    ServerThread thread;
    thread.start();

    mtime::sleep(0.1);

    MProcTask task(10);
    MBroker broker(&task, 0);

    Server* server =
        static_cast<Server*>(broker.obtain("localhost", 5125, "server"));

    assert(server);

    for(size_t i = 0; i < 10; ++i){
        cout << i << ": " << server->inc() << endl;
    }

    return 0;
}

```

## MPL

MPL is a high-performance language system which can be embedded within larger MML++ programs. More precisely, rather than being a complete language, it is a sort of meta language and set of reusable components and approaches which have been designed to work well together in an extremely high performance manner, while at the same time abstracting many of the details involved with generating and executing code in the most efficient manner using both the GPU and CPU multiprocessing.

MPL is an object-oriented system. Similar to the way that MObject is subclassed to automatically provide added functionality to an MML++ program, the **MPLObject** class is the unit of abstraction which holds the data for each object and which is subclassed when using MPL. This abstraction is most natural and familiar to those coming from a C++ background – whereas if you are programming with OpenCL or CUDA directly, you would work with arrays of data rather than objects and would manually manage copying data to and from the GPU device.

Typically in MPL, we are dealing with several similar (not necessarily identical) objects which run similar code – the “same program multiple data” (SPMD) paradigm. For example, you might specify an n-body kernel which executes on the GPU, followed by a section of MPL code which gets compiled to fast low-level machine code on the fly which is then executed concurrently on the CPU(s), then call ordinary C++ methods on it to perform visualization, just as you would on a typical C++ object. This is a highly flexible

approach whereby you can do a lot of the work in C++ on objects while delegating the compute intensive portions of the task to specialized MPL components.

The key idea is that we need certain parts of the program to run extremely fast while other parts are not on the critical path. For example, a typical compute-intensive simulation task is to perform operations on objects within certain distance of each other, e.g: collision detection and resolution. MPL features a class called **MPLLocality** which has been highly optimized for finding pairs of objects within a certain locality. These pairs can then be operated on by other components of MPL such by an **MPLNode** which has been compiled to react to such pairs.

MPL is implemented with OpenCL and LLVM under the hood to generate and execute code in the most efficient manner while at the time abstracting many of the details that are necessary so that the task of creating high performance MPL code and kernels is made as easy as possible.

The following snippet of embedded MPL code taken from `examples/collisions-fast` defines a kernel which executes on the GPU via **MPLKernel**:

```
static const float GRAVITY = 0.01;
static const float E = 0.5;
static const float D = 0.02;
static const float S = 0.1;

mnode code = mpl{
    :run
    float d = distance(u, #u);
    float r2 = r + #r;
    if(d > 0.0001 && d < r2){
        float3 un = normalize(#u - u);
        float3 vd = #v - v;
        v += -$E * (r2 - d) * un + $D * vd + $S * (vd - dot(vd, un) * un);
    }

    :end
    v[1] -= $GRAVITY;
    u += v;
};

class Object : public MPLObject{
public:
    Object(const float3& color, const float3& u)
        : color(color),
          u(u){

        v = [3 of uniform(-0.01, 0.01)];
        r = uniform(0.01, 0.1);
    }

    void draw(){ ... }

    float3 color;
    float3 u;
    float3 v;
    float r;
};

typedef MVector<Object*> ObjectVec;
ObjectVec objectVec;

for(size_t i = 0; i < N; ++i){
```

```

float3 color = [3 of uniform()];
float3 u = [3 uniform(-5, 5)];
objectVec.push_back(new Object(color, u));
}

```

We then pass the mnode corresponding to the kernel code and our object vector, to the MPLKernel which compiles the kernel and is now ready to run:

```

MPLKernel kernel;
kernel.compile(objectVec, code);

for(;;){
    startFrame();

    kernel.run();

    for(size_t i = 0; i < N; ++i){
        objectVec[i]->draw();
    }

    endFrame();
}

```

There are a few things worth clarifying at this point. The MPL code where the code mnode is defined, has two main types. Either it is an ordinary block of sequential code beginning with `:` or contains up to three sections labeled `:begin`, `:run`, and `:end`. Each object passed in the object vector essentially has its `:begin` section run first in parallel. The interesting part is the `:run` section, which executes next, and pairs each object up against every other object. For this reason, it is important that the code in this section be especially fast. Minor differences in the choice of codes can have a large impact, for example, consider the following:

```

float d2 = distance(u, #u);
d2 = d2 * d2;
float id = invSqrt(d2);
float id = id * id;
v * id;

```

And the functionally equivalent:

```

float d2 = distance(u, #u);
d2 = d2 * d2;
v / id;

```

Although it might not seem to be the case, the first code actually runs significantly faster. The proper choice of codes takes some experimentation and a knowledge of the different functions that MPL offers.

The `#` token is a special construct which means take the value of an attribute on the object *B* that the current object *A* is being paired up against. For this reason, the `#` operator may only appear within the `:run` section. In addition the `#`-qualified values are read-only – that is, we can read and write *A*'s values, but may only read *B*'s values. The `:end` section runs last, after `:run`, and like `:begin`, performs operations using only *A*'s values.

Note that MPLKernel has been designed to automatically manage copying memory to and from the GPU device as well as several other details associated with GPU programming. It also allows both reads and writes to variables in any part of the kernel. For instance, *u* is read in the run section, then modified in the end section while at the same time other objects might be reading the value `#u`.

This concludes our introduction to MPL. There are a variety of examples included in `$ANDROMETA_HOME/examples` which further demonstrate how MPL is used. In addition, you can take a look at the doxygen documentation for the MPL-prefixed header files.

## Link-level Compatibility

A major benefit of the MML++ compiler, is that it produces object code which is fully compatible with mainstream C++ compilers such as Clang or GCC/G++. This flexibility allows MML++ to create object files or libraries which can then be linked with other C++ programs which were not written using MML++.

## MMage

The MMage (Math and Genetic-algorithms Evolver) class extends MObject, adding symbolic and higher-level math functionality and custom-built genetic-algorithms-inspired methods for evolving and generating M code, and more. Computer algebra functionality is made possible thanks to FriCAS / AXIOM.

MMage can be used to perform higher-level math using the methods described already for constructing M code. In the following MML++ code snippet, we use the mage to compute a derivative:

```
MMage mage;

mnode d = mage.Derivative_(ml{x^2 + x}, ml{x});

cout << "derivative is: " << d << endl;
```

MMage contains an extensive library of advanced and symbolic math routines which take advantage of the M language's functional programming roots and ability to handle closures. Please refer to the MMage.h header or AndroMeta doxygen / API reference for a listing of functions supported by MMage.

One of MMage's primary features is the ability to evolve an *objective*, which is essentially a portion of M code generated using supervised (based on data) or unsupervised (code plugged into a program) using genetic programming and other machine learning techniques. `$ANDROMETA_HOME/examples/ipd` is an example of the *Iterated Prisoner's Dilemma* using the MMage's objective evolving facilities in the attempt to develop game-theoretic strategies.

The Prisoner's Dilemma is a simple game which puts two players up against each other. Each player can choose to either "defect" or "cooperate". A payoff matrix provides the rewards based on the four possible outcomes CC, DD, CD, DC. The game is most interesting when there is a high incentive to defect but punishes players when both defect, e.g: CC=3, CD=0, DC=5, DD=1. There is room for complexity when the players are modeled as agents with memories who can decide which action to take based on the past actions of opponents. The ipd example puts multiple players up against each other for several rounds. It provides a set of potential M code building blocks such as:

```
outcome(i) - the outcome of the previous round i against the current opponent
majority(n) - the majority outcome of the last n rounds against the current opponent
```

`count(n, o)` - the count of the the last `n` rounds of outcome `o` against the current opponent

... and so on. Once we have implemented these functions, we can chain them together in the attempt to create an effective strategy, e.g: `outcome(0) == CC && outcome(1) == DD`, with a boolean return value of true indicating the defect action. This example highlights the power of subclassing an `MObject` to allow its methods to be called from `M`. We can define several native methods in C++ which are now callable from `M` to generate and execute arbitrarily complex `M` code at runtime:

```
And(And(Eq(outcome(0),CC),Eq(outcome(1),CC))).
```

The first step is to create a named objective using the following method, e.g:

```
mage.addObjective("strategy", POPULATION, STRATEGY_DEPTH);
```

The `mage` maintains a population of strategies which get mutated and crossed over to iteratively evolve better solutions. Typical population setting are around a few thousand or so. Each agent is effectively an `MMage` which keeps its own population of strategies. The depth parameter determines the complexity of each generated solution. Usually this is set to somewhere around 3 - 5, as larger values take exponentially longer to generate and evaluate as this is effectively the maximum depth of the tree of the generated `M` code. Several other parameters are available which have been defaulted in the above call such as the mutation rate, selection pressure, etc. Please refer to the `MMage.h` header file for a description of each.

Once we have created a named objective, we add a variable to be evolved, `S`:

```
mage.addObjectiveVariable("strategy", "strategy", "S");
```

Even though `MObject`-based interpreters are manipulating `M` code under the hood, we can more conveniently use the higher-level language `MML` or `ML` language to specify the production rules / grammar whereby `S` is formed:

```
mage.addObjectiveProduction("strategy", "S",
    ml{ [outcome(N) == 0,
        majority(N) == 0,
        minority(N) == 0,
        count(N,0) > N,
        count(N,0) > N,
        S && S,
        S || S] }
);
```

We can break the grammar up into multiple productions:

```
mage.addObjectiveProduction("strategy", "0",
    ml{ [$CC,$DC,$CD,$DD] }
);
```

As we saw in the `MPL` example, a symbol prefixed by `$` allows an embedded language to capture or interpolate the value a variable defined in C++ scope.

```
mage.addObjectiveProduction("strategy", "N",
    ml{ [0,1,5,Add(N,N)] }
);
```

We perform each round by putting each agent up against all others for several iterations:

```

for(size_t k = 0; k < ROUNDS; ++k){
    cout << "starting round: " << k << endl;

    mnode ss = mage.iterateObjective("strategy");
    for(size_t i = 0; i < agentVec.size(); ++i){
        Agent* agent = agentVec[i];

        mnode s;

        if(i >= ss.size()){
            size_t j = MLib::equilikely(0, ss.size() - 1);
            s = ss[j][0][1];
        }
        else{
            s = ss[i][0][1];
        }

        agent->init(s);
    }

    for(size_t l = 0; l < ITERATIONS; ++l){
        for(size_t i = 0; i < agentVec.size(); ++i){
            for(size_t j = 0; j < agentVec.size(); ++j){
                if(i == j){
                    continue;
                }
                Agent* a1 = agentVec[i];
                Agent* a2 = agentVec[j];
                a1->play(*a2);
            }
        }
    }

    mage.clearObjectiveSolutions("strategy");

    for(size_t i = 0; i < agentVec.size(); ++i){
        Agent* a = agentVec[i];
        mnode s = mfunc("Tuple") + (mfunc("Tuple") +
                                   msym("strategy") + a->strategy());

        mnode rs = mage.Reduce_(a->strategy());

        cout << "strategy: " << MMLGenerator::toStr(rs) << endl;
        cout << "score: " << a->score() << endl;

        mage.addObjectiveSolution("strategy", s, 1.0/a->score());
    }

    double error;
    mnode b = mage.objectiveSolution("strategy", 0, &error);

    mnode s = mage.Reduce_(b[0][1]);

    cout << "best score: " << 1/error << endl;
    cout << "best strategy: " << MMLGenerator::toStr(s) << endl;
}

```

We calculate the score for each round as the sum of the outcomes and pass back to the mage so that this metric can be used to perform crossover and mutation in the attempt to find a better strategy in successive

rounds. We are attempting to minimize the error metric so we pass the inverse of the combined score. We use the `MMage`'s `Reduce()` function to reduce the strategy to its simplest form before outputting it.

This concludes our introduction to `MMage`. We have shown the essential features of this class, but there is a lot more depth to this class which can be learned by examining the `MMage.h` header file and experimenting with the included examples.

# Acknowledgements

We are extremely grateful to a number of high-quality open-source projects / libraries which AndroMeta utilizes, including:

- LLVM and Clang (on which the MML++ compiler is based and LLVM is used for code generation).
- The OGRE 3D Graphics Engine (for MML scenes)
- FFmpeg (for encoding MML scene animation to MPEG)
- FriCAS / AXIOM (for computer algebra)
- GMP and MPFR (for high precision numerics)
- CorePlot (for plotting on the Mac)

For a complete list of libraries and licensing information, please see the Copyrights and Licenses section included at the end of this document.

Special thanks to Bruce Graham for his thorough feedback and ideas concerning MMLGrid.

## References

1. **The Scout Programming Language**. Applied Computer Science Group, Los Alamos National Laboratory.
2. **MPFR**. <http://www.mpfr.org>
3. **GMP**. <http://gmplib.org>
4. **FriCAS / AXIOM**. <http://sourceforge.net/projects/fricas>
5. **The OGRE 3D Graphics Engine**. <http://www.ogre3d.org>

# Appendix

## Interfacing MMage with Mathematica

For those who have purchased Mathematica, Mathematica can be used with MMage by creating a symbolic link as follows:

```
$ cd $ANDROMETA_HOME/resources
$ ln -s <path to MathKernel or math command> MathKernel
```

Please refer to the Mathematica documentation for details on locating the MathKernel or math command. If you have specific questions, please contact us.

## MProgram Framework Arguments

`mThreads` - An integer giving the number of threads to be used in `MProcTask` associated with running an MML program. When threading is desired, this is typically set to the number of cores/CPU's on the host machine. Default value is 0.

`mAbort` - Set to `true` to abort execution the first time an `MError` exception is thrown, as opposed to potentially having it re-thrown which can complicate debugging. Default value is `false`.

`mStrict` - Applies to the default strict mode for newly created `MObject` and derived classes. When in strict mode, an `MProcessError` is thrown upon attempting to execute undefined symbols or functions. In non-strict mode, an undefined symbol or function simply has its node returned unevaluated. Default value is `true`.

`mExact` - Applies to exact evaluation mode for `MObject` and derived classes. When in exact mode, math-related functions calls, for example, `Cos(2)` will return `Cos(2)`, a symbolic answer, instead of a floating-point approximate answer `-0.416147`. As expected, when in exact mode, `Cos(0)` returns 1. Default value is `false`.

`mSeed` - Gives the seed of the global random number generator used by the stochastic calls such as those in `Mlib`. Note that this does not apply to instantiations of `MRandom` which have an associated state and seed. Default value is 0.

`mIndexLarge` - Set to `false` to not have the sequence portion of an `mvar` displayed with indices when its size is greater than 10. Default value is `true`.

`mImproper` - Set to `true` to have rationals displayed as improper fractions. Default value is `false`.

`mMapDuplicateKeys` - Set to `false` to have an `MError` thrown when an `mvar` is created which contains duplicate keys. Default value is `true`.

`mPrecise` - Set to `true` to have floating point values outputted with maximum precision. Default value is `false`.

`mVerify` - This flag is enabled to perform additional run-time checks on various call parameters throughout the framework. This verification is disabled by default as it could impact performance significantly. Enabling it can be useful in some situations for debugging purposes such as upon getting an exception, seg. fault, etc. Default value is `false`.

# Copyrights and Licenses

AndroMeta is free for non-commercial and academic use. If you are interested in obtaining an LGPL-style license to link to the AndroMeta libraries or if you plan to use AndroMeta for commercial purposes, please contact us at:

aminfo@andrometa.net

NOTE: The following license DOES NOT APPLY to the external shared libraries distributed with AndroMeta. Please see the separate licensing information included at the end of this document. It DOES APPLY to the libraries named libandrometa\*, AndroMeta.framework, and other software which makes up this distribution.

---

## SOFTWARE LICENSE AGREEMENT FOR ANDROMETA

IMPORTANT PLEASE READ THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT CAREFULLY BEFORE CONTINUING: AndroMeta LLC's Software License Agreement ("AGREEMENT") is a legal agreement between you (either an individual or a single entity) and AndroMeta LLC for the software product(s) identified above which may include associated software components, media, printed materials, and "online" or electronic documentation ("SOFTWARE PRODUCT"). By installing, copying, or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this AGREEMENT. This license agreement represents the entire agreement concerning the program between you and AndroMeta LLC, (referred to as "licenser"), and it supersedes any prior proposal, representation, or understanding between the parties. If you do not agree to the terms of this AGREEMENT, do not install or use the SOFTWARE PRODUCT.

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

### 1. GRANT OF LICENSE.

The SOFTWARE PRODUCT is licensed as follows:

#### (a) Installation and Use.

AndroMeta LLC grants you the right to install and use copies of the SOFTWARE PRODUCT free of charge for non-commercial purposes. Software that is distributed or made publicly available that links to the AndroMeta libraries or uses AndroMeta in any way must be accompanied by permission or a license agreement obtained by contacting AndroMeta LLC. Any type of commercial use, whether internally or through externally distributed software, must be accomplished by obtaining a proper license from AndroMeta LLC.

#### (b) Backup Copies.

You may also make copies of the SOFTWARE PRODUCT as may be necessary for backup and archival purposes.

### 2. DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS.

#### (a) Maintenance of Copyright Notices.

You must not remove or alter any copyright notices on any and all copies of the SOFTWARE PRODUCT.

#### (b) Distribution.

The SOFTWARE PRODUCT may be freely distributed so long as external software linking to it has been given the proper licensing approval by AndroMeta LLC.

(c) Prohibition on Reverse Engineering, Decompilation, and Disassembly.

You may not reverse engineer, decompile, or disassemble the SOFTWARE PRODUCT, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation.

(d) Rental.

You may not rent or lease the SOFTWARE PRODUCT.

(e) Support Services.

AndroMeta LLC may provide you with support services related to the SOFTWARE PRODUCT ("Support Services"). Any supplemental software code provided to you as part of the Support Services shall be considered part of the SOFTWARE PRODUCT and subject to the terms and conditions of this AGREEMENT.

(f) Compliance with Applicable Laws.

You must comply with all applicable laws regarding use of the SOFTWARE PRODUCT.

### 3. TERMINATION

Without prejudice to any other rights, AndroMeta LLC may terminate this AGREEMENT if you fail to comply with the terms and conditions of this AGREEMENT. In such event, you must destroy all copies of the SOFTWARE PRODUCT in your possession.

### 4. COPYRIGHT

All title, including but not limited to copyrights, in and to the SOFTWARE PRODUCT and any copies thereof are owned by AndroMeta LLC or its suppliers. All title and intellectual property rights in and to the content which may be accessed through use of the SOFTWARE PRODUCT is the property of the respective content owner and may be protected by applicable copyright or other intellectual property laws and treaties. This AGREEMENT grants you no rights to use such content. All rights not expressly granted are reserved by AndroMeta LLC.

### 5. NO WARRANTIES

AndroMeta LLC expressly disclaims any warranty for the SOFTWARE PRODUCT. The SOFTWARE PRODUCT is provided 'As Is' without any express or implied warranty of any kind, including but not limited to any warranties of merchantability, noninfringement, or fitness of a particular purpose. AndroMeta LLC does not warrant or assume responsibility for the accuracy or completeness of any information, text, graphics, links or other items contained within the SOFTWARE PRODUCT. AndroMeta LLC makes no warranties respecting any harm that may be caused by the transmission of a computer virus, worm, time bomb, logic bomb, or other such computer program. AndroMeta LLC further expressly disclaims any warranty or representation to Authorized Users or to any third party.

### 6. LIMITATION OF LIABILITY

In no event shall AndroMeta LLC be liable for any damages (including, without limitation, lost profits, business interruption, or lost information) rising out of 'Authorized Users' use of or inability to use the SOFTWARE PRODUCT, even if AndroMeta LLC has been advised of the possibility of such damages. In no event will AndroMeta LLC be liable for loss of data or for indirect, special, incidental, consequential (including lost profit), or other damages based in contract, tort or otherwise. AndroMeta LLC shall have no liability with respect to the content of the SOFTWARE PRODUCT or any part thereof, including but not limited to errors or omissions contained therein, libel, infringements of rights of publicity, privacy, trademark rights, business interruption, personal injury, loss of privacy, moral rights or the disclosure of confidential information.

----- EXTERNAL LIBRARIES LICENSE -----

The following copyrights and licenses apply to the external libraries and other resources that AndroMeta uses. AndroMeta libraries are covered by the license above.

----- OGRE LICENSE -----

OGRE is licensed under the GNU Lesser General Public License (LGPL), with the following exceptions / clarifications:

1. Making modifications to OGRE configuration files, build scripts and configuration headers such as OGREConfig.h in order to create a customised build setup of OGRE with the otherwise unmodified source code, does not constitute a derived work
2. Building against OGRE headers which have inlined code does not constitute a derived work
3. User code which subclasses OGRE classes does not form a derivative work
4. Statically linking the OGRE libraries into a user application does not make the user application a derived work, and therefore does not require the user to pass on the source code or object code to their own application. The OGRE source code with all modifications must still be passed on in the same way as using OGRE as a shared library.
5. Using source code obfuscation on the OGRE source code when distributing it is not permitted .

These exceptions / clarifications shall be deemed to amend the base LGPL text, as reproduced below.

GNU LESSER GENERAL PUBLIC LICENSE  
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.  
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA  
Everyone is permitted to copy and distribute verbatim copies  
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source

code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE  
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) The modified work must itself be a software library.
- b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
- d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has

a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not.

Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
- b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made

generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

#### NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

#### END OF TERMS AND CONDITIONS

#### How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the

ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>
```

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the
library 'Frob' (a library for tweaking knobs) written by James Random Hacker.
```

```
<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice
```

That's all there is to it!

----- FRICAS / AXIOM LICENSE

Copyright (c) 1991-2002, The Numerical Algorithms Group Ltd.  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of The Numerical Algorithms Group Ltd. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

----- GMP LICENSE

The GMP library is licensed as follows:

GNU GENERAL PUBLIC LICENSE  
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>>  
Everyone is permitted to copy and distribute verbatim copies  
of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

## TERMS AND CONDITIONS

### 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

### 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for

the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

## 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

## 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

## 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

## 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This license gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

## 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is

available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

## 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place

additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

## 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the

licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

#### 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

#### 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

#### 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a

publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

#### 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

#### 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

#### 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered

version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

#### 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

#### 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

#### 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

### END OF TERMS AND CONDITIONS

#### How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>  
Copyright (C) <year> <name of author>
```

```
This program is free software: you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation, either version 3 of the License, or  
(at your option) any later version.
```

```
This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.
```

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program> Copyright (C) <year> <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

----- MPFR LICENSE

MPFR is licensed under the LGPL version 2.1 which is included above.

----- LLVM LICENSE

=====  
LLVM Release License  
=====

University of Illinois/NCSA  
Open Source License

Copyright (c) 2003-2011 University of Illinois at Urbana-Champaign.  
All rights reserved.

Developed by:

LLVM Team

University of Illinois at Urbana-Champaign

<http://llvm.org>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal with the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

- \* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimers.
- \* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimers in the documentation and/or other materials provided with the distribution.
- \* Neither the names of the LLVM Team, University of Illinois at Urbana-Champaign, nor the names of its contributors may be used to endorse or promote products derived from this Software without specific prior written permission.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS WITH THE SOFTWARE.

=====  
Copyrights and Licenses for Third Party Software Distributed with LLVM:  
=====

The LLVM software contains code written by third parties. Such software will have its own individual LICENSE.TXT file in the directory in which it appears. This file will describe the copyrights, license, and restrictions which apply to that code.

The disclaimer of warranty in the University of Illinois Open Source License applies to all code in the LLVM Distribution, and nothing in any of the other licenses gives permission to use the names of the LLVM Team or the University of Illinois to endorse or promote products derived from this Software.

The following pieces of software have additional or alternate copyrights, licenses, and/or restrictions:

Program	Directory
Autoconf	llvm/autoconf llvm/projects/ModuleMaker/autoconf llvm/projects/sample/autoconf
CellSPU backend	llvm/lib/Target/CellSPU/README.txt
Google Test	llvm/utils/unittest/googletest
OpenBSD regex	llvm/lib/Support/{reg*, COPYRIGHT.regex}

----- CLANG LICENSE

=====  
LLVM Release License  
=====

University of Illinois/NCSA  
Open Source License

Copyright (c) 2007-2011 University of Illinois at Urbana-Champaign.  
All rights reserved.

Developed by:

LLVM Team

University of Illinois at Urbana-Champaign

<http://llvm.org>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal with the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

- \* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimers.
- \* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimers in the documentation and/or other materials provided with the distribution.
- \* Neither the names of the LLVM Team, University of Illinois at Urbana-Champaign, nor the names of its contributors may be used to endorse or promote products derived from this Software without specific prior written permission.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS WITH THE SOFTWARE.

=====  
The LLVM software contains code written by third parties. Such software will have its own individual LICENSE.TXT file in the directory in which it appears. This file will describe the copyrights, license, and restrictions which apply to that code.

The disclaimer of warranty in the University of Illinois Open Source License applies to all code in the LLVM Distribution, and nothing in any of the other licenses gives permission to use the names of the LLVM Team or the University of Illinois to endorse or promote products derived from this Software.

The following pieces of software have additional or alternate copyrights, licenses, and/or restrictions:

<u>Program</u>	<u>Directory</u>
<none yet>	

----- CORE PLOT LICENSE

Copyright (c) 2010, Drew McCormack, Brad Larson, Eric Skroch, Barry Wark, Dirkjan Krijnders, Rick Maddy, Vijay Kalusani, Caleb Cannon  
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the Core Plot Project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.