

---

# Distributed Applications Design on Mac OS X

Jean-Matthieu Schaffhauser  
MSc. Computer Science  
Student #03053404  
Oxford Brookes University  
*09<sup>th</sup> September 2004*

# Abstract

The need for distributed computing is growing exponentially everyday. The best examples nowadays are the Internet and even more so, the Grid. There are many ways distributed applications are used, with much success in scientific research, through large projects such as AIDS@home or Distributed.net where computer users can join forces in seemingly insurmountable computational challenges.

This dissertation identifies distributed applications development methods on Mac OS X and the context they should be used in.

Mac OS X's favourite development language is Objective-C. This programming language provides many ways for objects to interact with one another. Using protocols, notifications or distributed objects techniques offered by Objective-C, we will see that a standard application can easily be turned into a distributed application.

We will also see how the XML/RPC standards work and how we can access them using Objective-C.

Finally, I will present MPI standards and the Message Passing Interface API required in order to initiate an Objective-C language binding to the standard MPI C programming interface.

# Dissertation Proposal

## Distributed applications design on Mac OS X

Proposal approved by Professor *Chris Cox*.

*19<sup>th</sup> April 2004*

## Objectives

One of my objective is to present various techniques for designing distributed applications on Mac OS X and should inform the reader when one should be used or discarded.

Moreover, I wish to program a complete Objective-C framework conform to MPI standards Version 1 to provide Objective-C developers a familiar way to develop MPI-based applications.

This dissertation should inform the reader on the following :

- Objective-C Language facilities for developping distributed applications.
- XML/RPC standards
- XML/RPC development with Objective-C
- MPI Standards
- MPI development
- Creating an Objective-C Framework

My work should also convince him that :

- Objective-C makes it easy to develop distributed applications.
- XML/RPC standards will let him design distributed application in heterogeneous network.
- MPI, often though as a very complex library, can be efficiently used knowing just a few about it.

## Methods

### Developing distributed applications with Objective-C

I assume that the reader will have a minimum knowledge of the Objective-C paradigms and the language syntax.

- Notifications: Notifications are message that can be trasmit within an application or between to clearly identified applications. I will discuss the notifications centers as well and pointed out any security flaws that can result in their use.
- Distributed Objects: Then, I will present distributed objects concepts, how connections between instances of objects can be made and how they can be used in a simple distributed Agenda I will code to illustrate the above.

### XML/RPC through Objective-C

- Standards first: How does it work ? These sections should give the reader a background on XML/RPC technologies before he/she continues to the next sections dedicated to XML/RPC programming.
- XMLRPCObjC: XMLRPCObjC [PADL03] is an opensource Framework develop by PADL Software Pty Ltd to access XML/RPC technologies through Objective-C. I

will illustrate those techniques implementing a news feed reader that updates its content based on XML file retrieved by XML/RPC calls.

## Message Passing Interface

- A brief introduction to MPI will be given. I will illustrate various problems that can be solved with MPI as an introduction to MPI usage in development context.
- Objective-C Framework: I will then develop a framework to access MPI within Objective-C applications. This task requires a lot of effort and should be greatly guided by different implementations already existing in C++.

## Resources

Books, CPUs and caffeine ...

### Books

- Parallel Programming in C with MPI and OpenMP [QUIN03].
- Building Cocoa Applications, A Step-By-Step Guide [GAMA02]
- Introduction to Parallel Computing [GGKK03], Chapter 6: *Programming using the Message Passing paradigm*
- Parallel Programming using C++ [WILU96], Chapter 12: *MPI++*
- Sourcebook of Parallel Computing [DFFG03], Chapter 13: *Parallel object-oriented libraries*
- Parallel Programming [WIAL99], Chapter 2: *Message Passing programming*

## Internet Resources

- <http://developer.apple.com>: A great Objective-C knowledge base.
- <http://www.xmlrpc.org>: XML/RPC Homepage.
- <http://xmlrpc-c.sf.net>: XML/RPC-C Homepage.
- <http://www.mpi-forum.org>: MPI Forum Homepage.
- <http://www.erc.msstate.edu/mpi/mpi++>: MPI++ Homepage.

## Schedule

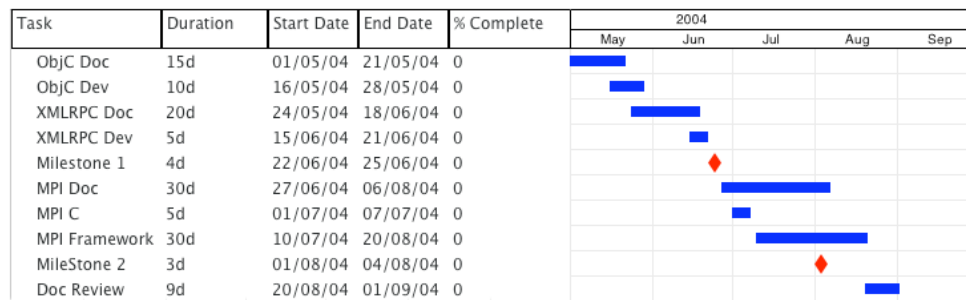


Figure 1: Gantt chart

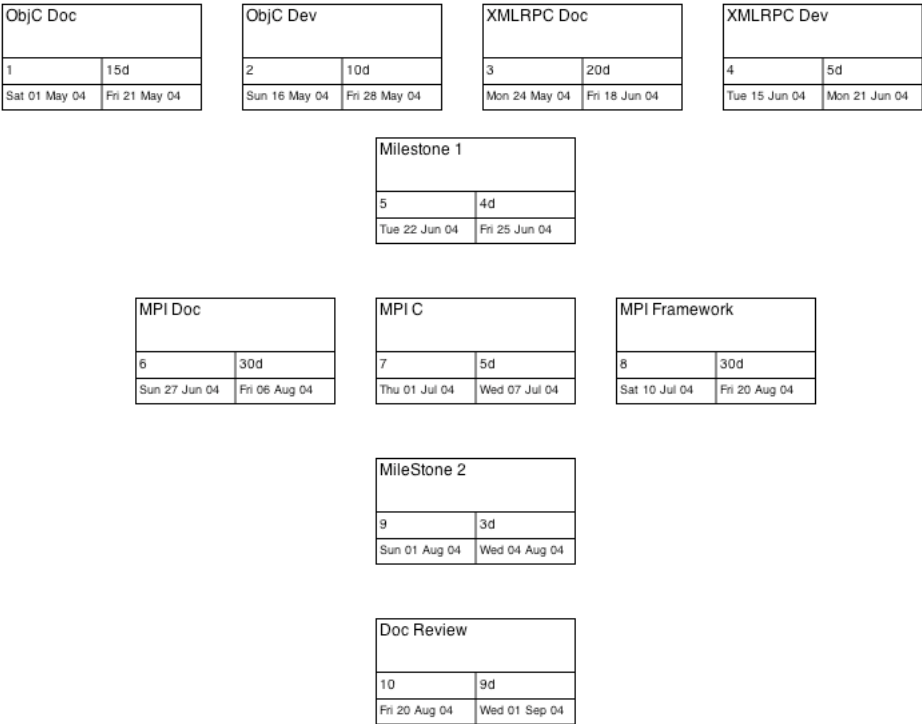


Figure 2: PERT chart



# Table of Contents

<b>Dissertation Proposal</b>	<b>ii</b>
Objectives . . . . .	ii
Methods . . . . .	iii
Developing distributed applications with Objective-C . . . . .	iii
XML/RPC Through Objective-C . . . . .	iii
Message Passing Interface . . . . .	iv
Resources . . . . .	iv
Schedule . . . . .	vi
Gantt Chart . . . . .	vi
PERT Chart . . . . .	vi
<b>1 Interapplication Communication Using Objective-C</b>	<b>1</b>
1.1 Distributed Objects Overview . . . . .	1
1.2 Connection Setup and Object Proxy . . . . .	7
1.3 Implementing an Objective-C Distributed Application . . . . .	13
<b>2 XML-RPC Programming</b>	<b>18</b>
2.1 Introduction to XML-RPC . . . . .	18
2.2 XML-RPC Specifications . . . . .	19
2.3 XMLRPCObjC: An XML-RPC Framework for Mac OS X . . . . .	22
2.3.1 API Overview . . . . .	23
2.4 Freshmint: A Client to Freshmeat.net XML-RPC Interface . . . . .	30

---

<b>3</b>	<b>Message Passing Programming with MPI</b>	<b>36</b>
3.1	Introduction to the Message Passing Interface . . . . .	37
3.2	MPI Operations . . . . .	43
3.2.1	Point-to-Point Communications . . . . .	46
3.2.2	Collective Communications . . . . .	52
<b>4</b>	<b>MPIObjC: A Language Binding to MPI</b>	<b>58</b>
4.1	MPIInstance Class Reference . . . . .	58
4.1.1	Detailed Description . . . . .	59
4.1.2	Member Function Documentation . . . . .	59
4.2	MPIComm Class Reference . . . . .	63
4.2.1	Detailed Description . . . . .	64
4.2.2	Member Function Documentation . . . . .	64
4.3	MPIRequest Class Reference . . . . .	75
4.3.1	Detailed Description . . . . .	75
4.3.2	Member Function Documentation . . . . .	75
4.4	Coding with MPIObjC . . . . .	77
	<b>Bibliography</b>	<b>86</b>
	<b>Source code repository</b>	<b>88</b>

# List of Tables

1.1	Objective-C Special Type Qualifiers . . . . .	7
2.1	XML-RPC Type Qualifiers . . . . .	22
3.1	Basic MPI Datatypes . . . . .	38
3.2	Send Modes Calling Routines . . . . .	52
3.3	MPI_Reduce Operators . . . . .	55

# List of Figures

1	Gantt chart . . . . .	vi
2	PERT chart . . . . .	vii
1.1	Sending a Message to a Vended Object . . . . .	3
1.2	Comminucation between proxies and objects . . . . .	12
1.3	PhotoAlbum Client Application Screenshot . . . . .	15
2.1	XML-RPC Transport . . . . .	19
2.2	Freshmint Screenshot . . . . .	31
3.1	MPI_Send Arguments . . . . .	46
3.2	MPI_Recv Arguments . . . . .	47
3.3	MPI - Broadcast Operation . . . . .	54
3.4	MPI - Reduce Operation . . . . .	54
3.5	MPI - Gather Operation . . . . .	56
3.6	MPI - Scatter Operation . . . . .	56

# List of Programs

1.2.1 Vending an Object with <code>NSConnection</code> . . . . .	8
1.2.2 Getting a Vended Object . . . . .	10
1.3.1 PhotoAlbum Daemon . . . . .	13
1.3.2 PhotoAlbum Client . . . . .	15
2.3.1 A Simple XML-RPC Telephone Directory Server . . . . .	24
2.3.2 A Simple XML-RPC Telephone Directory Client . . . . .	27
2.3.3 A Simple XML-RPC Telephone Directory Proxy . . . . .	29
2.4.1 Freshmint Implementation of Freshmeat.net API . . . . .	31
3.2.1 MPI - Hello, World . . . . .	45
3.2.2 MPI - Blocking Send and Receive . . . . .	48
3.2.3 MPI - Non-blocking Send and Receive . . . . .	50
4.4.1 A Simple <code>MPIObjC</code> Program. . . . .	77
4.4.2 Using the <code>MPIComm</code> Object. . . . .	77
4.4.3 More Fun with <code>MPIComm</code> . . . . .	79
4.4.4 The Sieve of Eratosthenes . . . . .	80

# Chapter 1

## Interapplication Communication Using Objective-C

### 1.1 Distributed Objects Overview

In this section we will introduce the concept of *Distributed Objects* through everyday life examples and give the fundamentals of distributed objects programming. Readers shall continue to the next section for usage of fundamentals classes and code examples.

#### Introduction to Distributed Objects

In order to enable applications to call an object in a different application (or running in a different thread in the same application or on a different computer on the network), the Objective-C runtime supports an interprocess messaging solution called *Distributed Objects*.

With distributed objects one can split a complex task into different segments that run independently while exchanging messages to ensure whole application consistency. Imagine for example an application that would render a three-dimensional representation of a human brain. One would have his brain scanned at the hospital and the pictures would

be transferred to a computer for processing on the 4<sup>th</sup> floor. Back in her second-floor office, a graphical front-end displays the processed results to the doctor. The front-end can accept all the user input and tell the back-end to perform various steps (like zooming to a particular region of the brain). The back-end will handle the user's actions and inform the front-end to redraw its display with updated data when it is computed. Because the front and back ends run independently, our doctor can still queue other requests through the front-end to be processed later on.

One can also use distributed objects to implement parallel processing. Given a large process, break it into smaller processes, distribute them on multiple machines across a network and get the combined computational power of a computer room to complete a job.

On Mac OS X, Cocoa<sup>1</sup> allows distributed objects to communicate on a single machine over Mach ports and message ports. It uses standard Unix sockets so that they can communicate on large networks, such as the Internet. Remote messages can be sent synchronously, forcing the sender to stop its execution and wait for a reply before continuing, or asynchronously allowing the sender to continue its execution without waiting for a reply and ignoring any response from the remote object.

## Distributed Objects Architecture

This section describes Cocoa classes used to send a message to a remote, or *vended*, object. In a distributed object architecture, a server process will *vend* an object to which clients processes can access. A client will initiate a connection to a server vended object and invoke the remote object's methods. The methods a remote object can respond to are usually declared in a formal protocol available to the client. We will see in the next section how to declare and implement a protocol for a vended object and how a client can, in turn, use this one to invoke methods defined remotely.

---

<sup>1</sup>Apple's Objective-C Framework

Figure 1.1 describes the main steps a client process takes to send a message to an object vended by a server process. It indicates Cocoa classes used by this process. For each

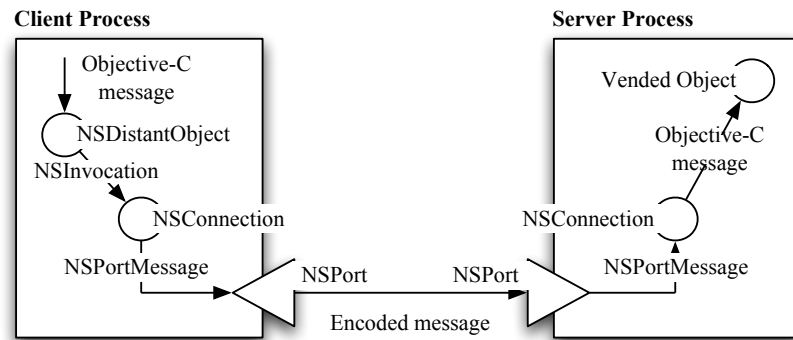


Figure 1.1: Sending a Message to a Vended Object. *Source: Apple Documentation - Distributed Objects*

object a server wants to vend, it will create an `NSConnection`<sup>2</sup> so that client processes can contact the object. The client process will gain access to the vended object connecting a `NSConnection` to the server's `NSPort` and requesting a *proxy* of the vended object. This *proxy* is referred to as a `NSDistantObject` and the client can send Objective-C messages to the object as it would usually do. If the distant object has no declaration on the client-side, it should conform to a specified protocol so that an `NSProtocolChecker` can filter out methods not implemented by the object's protocol (raising an exception caught by the client `NSConnection`) before forwarding any message to a distant object. The `NSConnection` is responsible for converting client's Objective-C message invocation (`NSInvocation`) into `NSPortMessage`, a message that could be encoded for transfer over an `NSPort` to a remote process or different thread. On the server-side, the encode data is converted back to an Objective-C message that the `NSConnection` forwards to the vended object which in turn can transparently return a value to the client.

It is important to know that the clients blocks until it receives a return value from the server or an exception has been raised.

<sup>2</sup>An `NSPort` is instantiated for every `NSConnection`.



## Objective-C Language Support for Distributed Object Architectures

### Protocols

The central concept of a protocol is that it declares methods that must be implemented by an object that wishes to conform to it. There are two kinds of protocols: informal and formal protocols.

#### Informal Protocols

An *informal protocol* is simply a category of an object, generally a category on `NSObject`, so that any class inheriting from `NSObject` can get the protocol's functionality by implementing the methods it declares. For example, the `NSTableDataSource` protocol is declared as:

```
@interface NSObject(NSTableDataSource)

- (int)numberOfRowsInTableView:(NSTableView *)aTableView;

- (BOOL)tableView:(NSTableView *)tableView
  acceptDrop:(id <NSDraggingInfo>)info
  row:(int)row
  dropOperation:(NSTableViewDropOperation)operation;

- (id)tableView:(NSTableView *)aTableView
  objectValueForTableColumn:(NSTableColumn *)aTableColumn
  row:(int)rowIndex;

- (void)tableView:(NSTableView *)aTableView
  setObjectValue:(id)anObject
  forTableColumn:(NSTableColumn *)aTableColumn
  row:(int)rowIndex;
```

```
- (void)tableView:(NSTableView *)tableView
    sortDescriptorsDidChange:(NSArray *)oldDescriptors;

- (NSDragOperation)tableView:(NSTableView *)tableView
    validateDrop:(id <NSDraggingInfo>)info
    proposedRow:(int)row
    proposedDropOperation:(NSTableViewDropOperation)operation;

- (BOOL)tableView:(NSTableView *)tableView
    writeRows:(NSArray *)rows
    toPasteboard:(NSPasteboard *)pboard;

@end
```

Any NSObject that becomes a datasource for a NSTableView should conform to this protocol and implement the method for which it wishes to override the default behaviour. Imagine for example a NSObject class called TableController and a NSTableView called tableView.

```
@interface ApplicationController : NSObject
    NSTableView *tableView
@end
```

```
@interface ApplicationController
-(id) init
{
    self = [super init];
    if(self)
    {
```

```
tableView = [[NSTableView alloc] init];
/* more code ...*/
[tableView setDataSource:self];
}
}

- (int)numberOfRowsInTableView:(NSTableView *)aTableView
{
    /* tableView should have 42 rows */
    if ([aTableView isEqualTo:tableView])
        return 42;
    else
        return 0;
}
@end
```

### Formal Procotols

Formal protocols are true protocols declared by the Objective-C language directive `@protocol`. For example, imagine a distributed application that would fetch pictures on a remote server to be displayed locally. A `PhotoAlbum` protocol for this purpose could be defined like this:

```
@protocol PhotoAlbum
- (bycopy NSImage *)showPicture:(in byref NSString *)pictureName;
@end
```

Any object willing to adopt this protocol states it in its class declaration:

```
@interface anObject: NSObject <PhotoAlbum>
{
}
}
```

@end

When an object adopts a formal protocol, it must implement all the methods declared in the protocol declaration or one will get compiler warnings.

### Type qualifiers

Objective-C defines six type qualifiers that can be used when declaring methods inside a formal protocol. There are listed in table 1.1.

Type	Description
<b>oneway</b>	Used for sending <b>asynchronous</b> messages, when one doesn't need to wait for a reply.
<b>in</b>	Information is being passed in a message.
<b>out</b>	Indicates that an argument is being used to return information by reference.
<b>inout</b>	Indicates that an argument is used both to provide information and to get information back. Default type for all pointer arguments except for those declared <b>const</b> , for which <b>in</b> is the default.
<b>bycopy</b>	Sends a copy of the object to the remote process so that the process can interact with the object directly in its own address space. (The application that receives the object must have the class of the object loaded in its address space.)
<b>byref</b>	Specifies that objects passed to a method or objects returned from a method should be passed or returned by reference.

Table 1.1: Objective-C Special Type Qualifiers. *Source: The Objective-C Programming Language - Remote Messaging*

## 1.2 Connection Setup and Object Proxy

### Communication Between Objects through the `NSConnection` Class

In a distributed objects architecture, the `NSConnection` class is the fundamental class for exchanging information between a server and its clients, a server and various threads of a client or between several threads inside the same application. `NSConnection` objects work on each communication end-point; these are instantiated explicitly on a server before an object we shall vend is attached to it. On the client side, it is used explicitly only for connecting to a remote server and setting the connection attributes (like the sending and

receiving ports, the connection timeout, the remote object we wish to gain access to, ...). Once one captures the vended object one wants to use, one will directly interact with it, conforming to a protocol priorly defined. Program 1.2.1 describes how a server vends an object.

Program 1.2.1: Vending an Object with `NSConnection`

```
1 #import "MyVendedObject.h"
2 #import <Foundation/Foundation.h>
3
4 int main (int argc, const char * argv[]) {
5     NSSocketPort *receivePort;
6     NSConnection *connection;
7
8     NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
9     NSRunLoop *runloop = [NSRunLoop currentRunLoop];
10    MyVendedObject *vendedObject = [[MyVendedObject alloc] init];
11
12    NS_DURING
13        // This server will wait for requests on port 4242
14        receivePort = [[NSSocketPort alloc] initWithTCPPort:4242];
15    NS_HANDLER
16        NSLog(@"Unable to get port 4242");
17        exit(-1);
18    NS_ENDHANDLER
19
20    // Create the NSConnection object
21    connection = [NSConnection connectionWithReceivePort:receivePort
22                                                         sendPort:nil];
23
24    // The port is retained by the connection
```

```
25     [receivePort release];
26
27     // When clients use this connection , they will
28     // talk to the vendedObject
29     [connection setRootObject:vendedObject];
30
31     // The chatter server is retained by the connection
32     [vendedObject release];
33
34
35     // Start the runloop
36     [runloop run];
37
38     // If the run loop exits (and I do not know why it would) , cleanup
39     [connection release];
40     [pool release];
41     return 0;
42 }
```

Program 1.2.2 describes how a client gets a vended object from a server. Note the `setRequestTimeout` and `setReplyTimeout` `NSConnection`'s methods, both set to 10 seconds, on lines 20 and 21. They will prevent us from waiting indefinitely if the link goes down. We could imagine another object on the client-side called `ConnectionStatus`, inheriting on `NSObject`, and delegate for our `connection`. It could handle `NSConnectionDidDieNotification` notification and clean the process when the link goes down.

Also note that on line 7 of this program, we define a variable `id proxy`. This proxy will be the remote object we access to and thus we set its protocol to `MyVendedObjectProtocol`. It is important to know that by telling the proxy about the protocol for the object it represents, we significantly reduce the network traffic involved in each invocation. Thus, we will define a protocol for every remote object we connect to in future projects.

## Program 1.2.2: Getting a Vended Object

```
1 #import "MyVendedObjectProtocol.h"
2 #import <Foundation/Foundation.h>
3
4 int main (int argc, const char * argv[]) {
5     NSSocketPort *sendPort;
6     NSConnection *connection;
7     id proxy;
8
9     NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
10
11     // Create the send port
12     sendPort = [[NSSocketPort alloc] initWithTCPPort:8081
13                                                         host:@"localhost"];
14
15     // Create an NSConnection
16     connection = [NSConnection connectionWithReceivePort:nil
17                                                         sendPort:sendPort];
18
19     // Set timeouts to something reasonable
20     [connection setRequestTimeout:10.0];
21     [connection setReplyTimeout:10.0];
22
23     // The send port is retained by the connection
24     [sendPort release];
25
26     // Get the proxy
27     proxy = [[connection rootProxy] retain];
28
```

```
29     // By telling the proxy about the protocol for the object
30     // it represents , we significantly reduce the network
31     // traffic involved in each invocation
32     [proxy setProtocolForProxy:@protocol(MyVendedObjectProtocol)];
33
34     // The rest of your program code goes here
35
36     // If the run loop exits (and I do not know why it would) , cleanup
37     [connection release];
38     [pool release];
39     return 0;
40 }
```

## Proxy

The Objective-C runtime refers to an instantiated object using a pointer to this one. This implies all threads run in a single address space limiting parallelism to shared-memory architectures. In order to exploit parallelism efficiently, a message call must return immediately, either a void value or an object such as **self** (a pointer to the object receiving messages). If we think about arguments passed by reference and modified by a routine, we don't want to use an argument before it has been modified by that routine running on a different application on a remote computer. But how long should we wait ? What could tell us when it is ready to be used again ?

Distributed Objects in Objective-C introduce the concept of *proxy*; a concept that can be summarized as a placeholder for a return value. The following paradigms apply to the use of proxies :

- One should be able to query a proxy to determine its state.
- Any subsequent use of a return value should block until the routine computing that value has finished.



Using proxies, one does not need to have a single address space for the objects to reside. Actually, a proxy can refer to an object residing in a different thread's address space; when it receives a message, it will forward it to the remote object the proxy was built for using the Objective-C **forward** mechanism<sup>3</sup>.

Figure 1.2 explains how various proxies and objects communicate with each other in a distributed grid object.

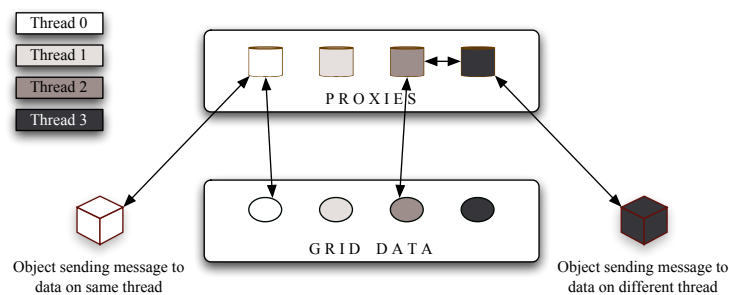


Figure 1.2: Communication between proxies and objects. *Source: Russell Standish - Distributed Object in Objective-C.*

### Cocoa's NSProxy class and its subclasses

In a distributed objects architecture, one often has the need to refer to objects that are not real objects for the client application. The **NSProxy** class is an abstract superclass for that kind of objects and sending a message to an **NSProxy** instance will result in this instance forwarding the message to the remote object it refers to. **NSProxy** has two concrete subclasses we are about to discuss: **NSDistantObject** and **NSProtocolChecker**.

#### NSDistantObject

A **NSDistantObject** is a proxy for an object in another thread or application. A client application object sends an Objective-C message to this class and the resulting **NSInvocation** is passed to a **NSConnection** responsible for converting and forwarding the message to destination as in Figure 1.1. It receives back a return value or if an exception

<sup>3</sup>If no method exists for a particular message, a **forward** message is sent to the object allowing the proxy to forward the message to a remote client.

is raised passes it so it can be caught.

The classic way to obtain a `NSDistantObject` is to call

`rootProxyForConnectionWithRegisteredName:host:`, an `NSConnection` message that returns the root proxy for a connection. Note that `NSDistantObject` adds a very useful method we use in Program 1.2.2 (`setProtocolForProxy:`) in order to set the methods the remote object responds to.

### **NSProtocolChecker**

Cocoa defines a special class `NSProtocolChecker` in the distributed objects system when one may want to vend only some methods of an object to an application cluster. This concrete subclass of `NSProxy` allows one to define which methods one wants to make remotely available, restricting the messages that can be sent to an object and raising an `NSInvalidArgumentException` when a message is not allowed.

## **1.3 Implementing an Objective-C Distributed Application**

To illustrate Cocoa distributed objects system, I coded a photo album application called *PhotoAlbum*. *PhotoAlbum* is composed of a server (Program 1.3.1) `photoalbumd` that exports pictures inside a user folder for clients (Program 1.3.2) to browse on a remote computer. This XCode project is available from <http://www.cocoanut.net/PhotoAlbum/>.

Program 1.3.1: PhotoAlbum Daemon

```

1 /*                                     9
2 * PhotoAlbumProtocol.h               10 #import <Foundation/Foundation.h>
3 * Protocol definition for exchanging photos 11
4 * Project: PhotoAlbum                12 // Objective-C messages the client will send to the
5 *                                     server
6 * Created by Jean-Matthieu.          13
7 *                                     14 @protocol PhotoAlbumServer
8 */                                   15 // Get the pictures list

```

```

17
16 - (bycopy NSArray *) getPicturesList;
18
19 // Retrieve a picture from the server
20 - (bycopy NSData*) getPicture:(in bycopy NSString
    *)pictureName;
21
22 @end
23
24
25 //
26 // PhotoAlbumServer.h
27 // PhotoAlbum
28 //
29 // Created by Jean-Matthieu.
30 //
31
32 #import "PhotoAlbumProtocol.h"
33 #import <Foundation/Foundation.h>
34
35
36 @interface PhotoAlbumServer : NSObject <
    PhotoAlbumServer> {
37     NSMutableArray *photos;
38 }
39
40 - (void)setup;
41
42 @end
43
44
45 //
46 // PhotoAlbumServer.m
47 // PhotoAlbum
48 //
49 // Created by Jean-Matthieu.
50 //
51
52 #import "PhotoAlbumServer.h"
53
54 static NSString *photoAlbumPath = nil;
55
56 @implementation PhotoAlbumServer
57
58 - (id)init
59 {
60     self = [super init];
61     [self setup];
62     return self;
63 }
64
65 - (void)dealloc
66 {
67     [photos release];
68     [super dealloc];
69 }
70
71 - (void)setup
72 {
73     if (photos)
74         [photos release];
75     photoAlbumPath = [NSString stringWithFormat:@"%@"
        @"/photos", NSHomeDirectory()];
76     photos = [[NSMutableArray alloc] initWithArray
        :[[NSFileManager defaultManager]
        directoryContentsAtPath:photoAlbumPath]];
77 }
78
79 #pragma mark Protocol Implementation
80 - (bycopy NSArray *) getPicturesList
81 {
82     return photos;
83 }
84
85 - (bycopy NSData *) getPicture:(in bycopy NSString
    *)pictureName
86 {
87     NSImage *anImage = [[NSImage alloc]
        initWithContentsOfFile:[photoAlbumPath
        stringByAppendingPathComponent:pictureName
        ] autorelease];
88     return [anImage TIFFRepresentation];
89 }
90
91
92 @end
93
94
95 /*
96 * photoalbumd.m
97 * PhotoAlbum
98 *
99 * Created by Jean-Matthieu.
100 *
101 */
102
103 #import "PhotoAlbumServer.h"
104 #import <Foundation/Foundation.h>
105
106 int main (int argc, const char * argv[]) {
107     NSSocketPort *receivePort;
108     NSAutoreleasePool * pool = [[NSAutoreleasePool
        alloc] init];
109     NSRunLoop *runloop = [NSRunLoop currentRunLoop
        ];
110     PhotoAlbumServer *photoAlbumD = [[
        PhotoAlbumServer alloc] init];
111
112     NS_DURING
113         // This server will wait for requests
114         // on port 4242
115         receivePort = [[NSSocketPort alloc]
        initWithTCPPort:4242];
116     NS_HANDLER

```

```

117     NSLog(@"Unable to get port 4242");
118     exit(1);
119     NS_ENDHANDLER
120
121     NSConnection *connection = [NSConnection
122         connectionWithReceivePort:receivePort
123
124         // The port is retained by the connection
125         [receivePort release];
126
127         // Set the responding server object as the root
128         object for this connection.
129         [connection setRootObject:photoAlbumD];
130
131         // The photo album is retained by the
132         connection
133         [photoAlbumD release];
134
135         sendPort
136         // Start the runloop
137         [runloop run];
138         // If the run loop exits (and I do not know why
139         it would), cleanup
140         [connection release];
141         [pool release];
142         return 0;
143     }

```



Figure 1.3: PhotoAlbum Client Application Screenshot

## Program 1.3.2: PhotoAlbum Client

```

1  /*
2  *   PhotoAlbumController.h
3  *   PhotoAlbum
4  *
5  *   Created by Jean-Matthieu.
6  *
7  */
8
9  #import <Cocoa/Cocoa.h>
10
11  @interface PhotoAlbumController : NSObject
12  {
13      IBOutlet NSButton *connectButton;
14      IBOutlet NSTextField *hostnameField;
15      IBOutlet NSTableView *photoTable;
16      IBOutlet NSImageView *photoViewer;
17      id proxy;
18      NSArray *myPhotos;
19  }
20  - (IBAction)connect:(id)sender;
21
22  - (void)doConnect;
23  - (void)doDisconnect;
24  @end
25
26
27  /*
28  *   PhotoAlbumController.m
29  *   PhotoAlbum
30  *
31  *   Created by Jean-Matthieu.
32  *
33  */
34

```

```

36 #import "PhotoAlbumController.h"
37
38 #
39     38 import "PhotoAlbumProtocol.h"
40 @implementation PhotoAlbumController
41
42 -(void) dealloc
43 {
44     if(proxy)
45         [self doDisconnect];
46     [super dealloc];
47 }
48
49 -(void) awakeFromNib
50 {
51     proxy = nil;
52     myPhotos = nil;
53 }
54
55
56 -(void) doConnect
57 {
58     NSConnection *connection;
59     NSSocketPort *sendPort;
60
61     // Create the send port
62     sendPort = [[NSSocketPort alloc]
63                 initWithTCPPort:4242
64                 host:[hostnameField
65                     stringValue]];
66
67     // Create an NSConnection
68     connection = [NSConnection
69                 connectionWithReceivePort:nil
70                 sendPort:
71                 sendPort];
72
73     // Set timeouts to something reasonable
74     [connection setRequestTimeout:10.0];
75     [connection setReplyTimeout:10.0];
76     // The send port is retained by the connection
77     [sendPort release];
78
79     NS_DURING
80         // Get the proxy
81         proxy = [[connection rootProxy] retain];
82         // By telling the proxy about the protocol
83         // for the object
84         // it represents, we significantly reduce the
85         // network
86         // traffic involved in each invocation
87
88     [proxy setProtocolForProxy:@protocol(
89         PhotoAlbumServer)];
90
91     NS_HANDLER
92         // If the server does not respond in 10
93         // seconds,
94         // this handler will get called
95         [self doDisconnect];
96     NS_ENDHANDLER
97 }
98
99 -(void) doDisconnect
100 {
101     NSConnection *connection = [proxy
102     connectionForProxy];
103     [connection invalidate];
104     [proxy release];
105     proxy = nil;
106     [myPhotos release];
107     myPhotos = nil;
108     [connectButton setTitle:@"Connect"];
109 }
110
111 -(IBAction) connect:(id) sender
112 {
113     if (!proxy){
114         [self doConnect];
115         if (!proxy)
116             return;
117
118         myPhotos = [[NSArray alloc] initWithArray:[
119             proxy getPicturesList]];
120         [connectButton setTitle:@"Disconnect"];
121         [photoTable reloadData];
122     } else {
123         [self doDisconnect];
124         [photoTable reloadData];
125     }
126 }
127
128 #pragma mark TableView delegate and datasource
129 -(int) numberOfRowsInTableView:(NSTableView *)
130     aTableView
131 {
132     if (nil != myPhotos){
133         return [myPhotos count];
134     } else
135         return 0;
136 }
137
138 -(id) tableView:(NSTableView *) aTableView
139     objectValueForTableColumn:(NSTableColumn *)
140     aTableColumn row:(int) rowIndex
141 {
142     if (nil != myPhotos){
143         return [myPhotos objectAtIndex:rowIndex];
144     } else
145         return nil;
146 }

```

```
136 }
137
138 - (void)tableViewSelectionDidChange:(NSNotification *)notification{
139     if ([photoTable selectedRow] != -1){
140         NSData *theData = [proxy getPicture:[
141             myPhotos objectAtIndex:[photoTable
142                 selectedRow]]];
143         UIImage *anImg = [[UIImage alloc]
144             initWithData:theData] autorelease];
145         [photoViewer setImage:anImg];
146     } else {
147         [photoViewer setImage:nil];
148     }
149 }
150
151 @end
152
153 //
154 // main.m
155 // PhotoAlbum
156 //
157 // Created by Jean-Matthieu on Tue Jun 15 2004.
158 // Copyright (c) 2004 --MyCompanyName--. All
159 // rights reserved.
160
161 #import <Cocoa/Cocoa.h>
162
163 int main(int argc, char *argv[])
164 {
165     return NSApplicationMain(argc, argv);
166 }
```

# Chapter 2

## XML-RPC Programming

*Simple cross-platform distributed computing, based on the standards of the Internet.*

### 2.1 Introduction to XML-RPC

*XML-RPC* is a W3C standard designed by Dave Winer for *UserLand*. Winer defines XML-RPC as

a specification and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet. It is remote procedure calling using HTTP as the transport and XML as the encoding. It is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned.

*RPC* stands for *Remote Procedure Call*, a specification that allows two end-points to communicate. Basically it is a common language understood and spoken by both parties. Figure 2.1 details how data is transported accross a network.

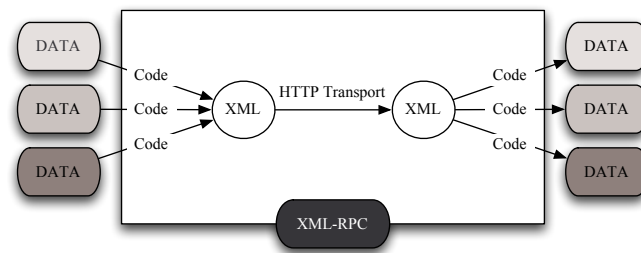


Figure 2.1: XML-RPC Transport. *Source: JY Stervinou*

## 2.2 XML-RPC Specifications

XML-RPC messages are exchanged between a client and a remote server using the XML format. When a client calls a remote procedure on a server, it posts an HTTP-POST request encapsulating XML inside the request's body. A procedure can carry parameters to the method it calls. To make it clearer, let's look at the basic **Hello, World !** example proposed by Winer.

### Request example

```
POST /RPC2 HTTP/1.0
```

```
User-Agent: Safari/1.2.2 (OSX)
```

```
Host: betty.userland.com
```

```
Content-Type: text/xml
```

```
Content-length: 181
```

```
<?xml version="1.0"?>
```

```
<methodCall>
```

```
  <methodName>examples.getStateName</methodName>
```

```
  <params>
```

```
    <param>
```

```
      <value><i4>41</i4></value>
```

```
    </param>
```



```
</params>
</methodCall>
```

**Header Requirements** The first line of the header defines the request responder to a procedure call. It can be omitted if the server only handles XML-RPC calls but allowing a URI<sup>1</sup> will help routing a request to the code designed for an XML-RPC request. **User-Agent** and **Host** are mandatory. The **Content-Type** is **text-xml** and the **Content-length** must be specified and accurate for the request to be handled.

**Request Details** The method called by this request is `examples.getStateName`. It takes an integer between 1 and 50 as argument and returns the corresponding state of the United States of America. Lovely, isn't it ? Now, let's see in details how the method is composed.

- **methodCall** is the *root* element of an XML-RPC request.
- **methodName** is the procedure we call. It is usually composed of a service name (mail, yellowpages, ...) and the procedure name we call, using Java-Style formatting.
- **params**, the parameters list we pass along to the request. We will see later on the various type that can be used. Remember that there is no restriction on the number of parameters; the list can be null or huge.

In reply to this XML-RPC call, we expect a return value, the name of the 41<sup>st</sup> state. I save you the response's header; it contains the **POST** request's return code (200 OK) and data similar to the post request's header.

```
<?xml version="1.0"?>
<methodResponse>
  <params>
```

---

<sup>1</sup>Uniform Resource Identifier

```
<param>
  <value><string>South Dakota</string></value>
</param>
</params>
</methodResponse>
```

Note the *root* element `methodResponse`. It contains a list of parameters `params` and is a common return value for any XML-RPC requests. If an error occurred when executing the remote procedure, the client should get informed by another `methodResponse` shown below :

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>4</int></value>
        </member>
        <member>
          <name>faultString</name>
          <value><string>Too many parameters.</string></value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

### XML-RPC Type Qualifiers

Table 2.1 list all types available for requests or reply parameters.

Type	Description
i4 or int	Integer value
boolean	Boolean value
string	String value
double	Double value
dateTime.iso8601	Date and hour conform to ISO8601
base64	64bits-coded binary
array	An array, such as NSArray
struct	Data structure such as NSDictionary

Table 2.1: XML-RPC Type Qualifiers

## 2.3 XMLRPCObjC: An XML-RPC Framework for Mac OS X

*XMLRPCObjC* binds the Objective-C language to XML-RPC specifications. It has been developed by Luke Howard for *PADL Software Pty Ltd* for almost 3 years. The framework has been designed regarding to the distributed objects system we mentioned previously. It enables XML-RPC methods invocation through proxy objects, translates Objective-C objects into XML-RPC valid parameters, and can automatically register Objective-C to be vended by an XML-RPC server.

There are few requirements that need to be satisfied before one can start using the API.

- A copy of the xmlrpc-c library. Source code is available from <http://xmlrpc-c.sf.net> or one can use compiled binaries for Mac OS X from my website (<http://cocoanut.net/xmlrpc>).
- A copy of w3c-libwww library from <http://www.w3c.org/Library/>. Again, compiled binaries are available on <http://cocoanut.net/xmlrpc>.

*XMLRPCObjC* is available from <http://www.padl.com/Research/XMLRPCObjC.html>. For convinience, I packaged it for Mac OS X and one will find it on my website as well.

In this section we will describe the framework's API and give examples to create both an XML-RPC server and client using the Objective-C framework.

### 2.3.1 API Overview

#### XMLRPCServer

XMLRPCServer is an abstract class inheriting on `NSObject` acting as an XML-RPC server. Objects that one wishes to distribute to remote clients are cache by a XMLRPCServer instance. See Program 2.3.1 for a detailed example.

#### Methods

- `+(XMLRPCServer *)server`: Factory method for XMLRPCServer class. Returns a XMLRPCServer object.
- `-(void)run`: Method to run the server. This method never returns; there is currently no support for runloop as in real distributed objects system.
- `-(void)setObject:(id)object forKey:(NSString *)target`: Sets an object to a target name. The target name is usually the prefix of an XML-RPC method, a service name for example.
- `-(id)objectForKey:(NSString *)target`: Retrieves an object from the server's object cache for a particular key.
- `-(void)removeObjectForKey:(id)aKey`: Removes an objects for a specified key from the XMLRPCServer object cache.
- `-(void)setObjectAutoCreation:(BOOL)yorn`: Automatically instantiates objects upon users' requests and adds it to the server's object cache.

## Program 2.3.1: A Simple XML-RPC Telephone Directory Server

```
1  /*
2  *   xmlrpcserver.m
3  *   A simple XML-RPC server
4  *   Compile with: gcc xmlrpcserver.m -framework Foundation -framework
        XMLRPCObjC -o xmlrpcserver
5  *   Run with ./xmlrpcserver
6  *
7  *   Created by Jean-Matthieu.
8  *
9  */
10
11 #import <Foundation/Foundation.h>
12 #import <XMLRPCObjC/XMLRPCObjC.h>
13
14 @interface telephoneDirectory : NSObject
15 - (NSDictionary *)cardForUser:(NSString *)aUser;
16 @end
17
18 @implementation telephoneDirectory
19 - (NSDictionary *)cardForUser:(NSString *)aUser;
20 {
21     /* Telephone Directory
22      * A dictionary where each entry is a username
23      * representing user's personal information stored as a
        dictionary
24      */
25     NSDictionary *telephoneDict = [NSDictionary
        dictionaryWithContentsOfFile:@"~/Users/jms/telephone.plist
```

```

        ”];
26     NSDictionary *result;
27     if (nil != (result = [telephoneDict objectForKey:aUser])){
28         return result;
29     } else {
30         return nil;
31     }
32 }
33 @end
34
35 int main (int argc, const char *argv[]) {
36     NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
37     XMLRPCServer *server = [XMLRPCServer server];
38     telephoneDirectory *td = [[telephoneDirectory alloc] init];
39
40     [server setObject:td forKey:@"telephoneDirectory"];
41
42     // The telephone directory is retained by the server
43     [td release];
44     /*
45      * run the server (never exits)
46      */
47     [server run];
48     [pool release];
49
50     exit(0);
51 }

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://
   www.apple.com/DTDs/PropertyList-1.0.dtd">

```

```

3 <plist version="1.0">
4 <dict>
5     <key>JMS</key>
6     <dict>
7         <key>Name</key>
8         <string>Jean-Matthieu Schaffhauser</string>
9         <key>School</key>
10        <string>Oxford Brookes University</string>
11        <key>Telephone</key>
12        <string>+44 1865 765 535</string>
13        <key>address</key>
14        <string>27 York Avenue Headington OX38NS UK</string>
15        <key>email</key>
16        <string>jean-matthieu@users.sourceforge.net</string>
17        <key>website</key>
18        <string>http://cocoanut.net</string>
19    </dict>
20 </dict>
21 </plist>

```

## XMLRPCClient

`XMLRPCClient` is an abstract class inheriting on `NSObject` acting as a XML-RPC client. Program 2.3.1 gives an example of a client to our telephone directory server.

### Methods

- `+(XMLRPCClient *)client:(NSURL *)url`: Returns an client to the specified *url* ready for remote method invocation.
- `-(id)invoke:(NSString *)method withArguments:(NSArray *)args`: This is the method used by clients to invoke an XML-RPC method.

- - (XMLRPCProxy \*)rootProxy: Returns a proxy object for a client session and forwards method invocations to the remote XMLRPC server.
- - (XMLRPCProxy \*)proxyForTarget:(NSString \*)name: Returns a proxy object for a specified service.

#### Program 2.3.2: A Simple XML-RPC Telephone Directory Client

```

1  /*
2   *  xmlrpcclient.m
3   *  A simple XML-RPC Client
4   *  Compile with: gcc xmlrpcclient.m -framework Foundation -framework
      XMLRPCObjC -o xmlrpcclient
5   *  Run with ./xmlrpcclient
6   *
7   *  Created by Jean-Matthieu.
8   *
9  */
10
11 #import <Foundation/Foundation.h>
12 #import <XMLRPCObjC/XMLRPCObjC.h>
13
14 int main (int argc, const char *argv[]) {
15     NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
16     XMLRPCClient *client;
17     id object;
18     NSString *username = [NSString stringWithString:@"JMS"];
19     NSArray *args = [NSArray arrayWithObject:username];
20
21     client = [XMLRPCClient client:[NSURL URLWithString:@"http://
      localhost:8000/RPC2"]];

```



```
22         object = [client invoke:@"telephoneDirectory.cardForUser"
23                     withArguments:args];
24
25         NSLog(@"%@ ", [object description]);
26
27         [pool release];
28         exit(0);
29     }
```

The result is displayed below. It conforms to a **struct** XML-RPC type as we expected since the method returns an `NSDictionary`.

```
{
    Name = "Jean-Matthieu Schaffhauser";
    School = "Oxford Brookes University";
    Telephone = "+44 1865 765 535";
    address = "27 York Avenue Headington OX38NS UK";
    email = "jean-matthieu@users.sourceforge.net";
    website = "http://cocoanut.net";
}
```

## XMLRPCProxy

`XMLRPCProxy` inherits on `NSProxy`. It allows a remote XML-RPC services to be accessed as if it were a local Objective-C object, just as we saw before. Please refer to Program 2.3.1 for an example.

### Methods

- `+(XMLRPCProxy *)proxyWithTarget:(NSString *)target client:(XMLRPCClient *)client`: Factory method. Instantiate a new `XMLRPCProxy` where *target* is an XML-RPC method's prefix available to *client*.

- - (XMLRPCProxy \*)proxyForTarget:(NSString \*)name: Creates a proxy with a target concatenated with the current target, a period, and the supplied argument. It also retains the current proxy's protocol.
- - (void)setProtocolForProxy:(Protocol \*)proto: Sets a protocol *proto* for a proxy object.
- - (XMLRPCClient \*)clientForProxy: Returns a XMLRPCClient *client* for an instantiated proxy object.

Program 2.3.3: A Simple XML-RPC Telephone Directory Proxy

```

1 /*
2  *  xmlrpcproxy.m
3  *  A simple XML-RPC Client
4  *  Compile with: gcc xmlrpcproxy.m -framework Foundation -framework
      XMLRPCObjC -o xmlrpcproxy
5  *  Run with ./xmlrpcclient
6  *
7  *  Created by Jean-Matthieu.
8  *
9  */
10
11 #import <Foundation/Foundation.h>
12 #import <XMLRPCObjC/XMLRPCObjC.h>
13
14 @protocol Bell
15 - (NSDictionary *)cardForUser:(NSString *)aUser;
16 @end
17
18 int main (int argc, const char *argv[]) {

```

```
19     NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
20     XMLRPCClient * client;
21     XMLRPCProxy <Bell> *telDirectory;
22
23     NSDictionary *result;
24
25     client = [XMLRPCClient client:[NSURL URLWithString:@"http://
        localhost:8000/RPC2"]];
26     telDirectory = (id <Bell>)[client proxyForTarget:@"
        telephoneDirectory"];
27     [telDirectory setProtocolForProxy:@protocol(Bell)];
28
29     result = [NSDictionary dictionaryWithDictionary:[telDirectory
        cardForUser:@"JMS"]];
30
31     NSLog(@"%@ ", [result description]);
32
33     [pool release];
34     exit(0);
35 }
```

## 2.4 Freshmint: A Client to Freshmeat.net XML-RPC Interface

Freshmeat.net, a popular software publication website, offers to subscribers to manage their projects through an XML-RPC interface. *Freshmint* is a freshmeat.net client for Mac OS X. It has the following features :

- Quickly view all the projects one owns.

- Browse the project's branches and view detailed information about the last updates.
- Add a new release for a project branch.
- Withdraw a release from public access.

**Freshmeat** class (Program 2.4.1) fully implements Freshmeat.net API. It handles all the remote procedure calls between Freshmint and freshmeat RPC's server. The full project is available on my website ( <http://www.cocoanut.net/freshmint/>).



Figure 2.2: Freshmint Screenshot

#### Program 2.4.1: Freshmint Implementation of Freshmeat.net API

```

1 //
2 //  Freshmeat.h
3 //  Freshmint
4 //
5 //  Created by Jean-Matthieu
6 //
7
8 #import <Foundation/Foundation.h>
9 #include <XMLRPCObjC/XMLRPCObjC.h>
10
11 @interface Freshmeat : NSObject
12 {
13     XMLRPCClient *client;
14     NSMutableDictionary *sessionDictionary;
15
16     BOOL isConnected;
17 }
18
19 /* [ sessionDictionary method ]
20     * Parameters:
21     * None
22     * Returns:
23     * NSDictionary with session informations
24     *
25     * Description:
26     * Returns a dictionary containing SID, API
27         Version, Lifetime, logintime
28 */
29 - (NSMutableDictionary *)sessionDictionary;
30
31
32
33 /* [ autoLogout method ]

```

```

36      *
37
38      * Parameters:
39      * None
40
41      * Description:
42      * Automatically logs out from Freshmeat.net
43 */
44 - (void)autoLogout;
45
46
47
48 /* [ isConnected method ]
49 * Parameters:
50 * None
51 *
52 * Returns:
53 * None
54 *
55 * Description:
56 * Informs whether a session is active or not.
57 */
58 - (BOOL)isConnected;
59
60
61
62 /* [ fetch_available_licenses method ]
63 * Parameters:
64 * None
65 *
66 * Returns:
67 * Array of available licenses
68 */
69 - (NSArray *)fetch_available_licenses;
70
71 /* [ fetch_available_release_foci method ]
72 * Parameters:
73 * None
74 *
75 * Returns:
76 * Struct of available release focus types and
    associated ID
77 */
78 - (NSDictionary *)fetch_available_release_foci;
79
80 /* [ fetch_project_list method ]
81 * Parameters (passed in struct form):
82 * SID                      - Session ID to work
    with
83
84 * Returns:
85 * Struct consisting of "projectname_full",
    86 * "projectname_short", "project_status", and "
    project_version"
87 */
88
89 - (NSDictionary *)fetch_project_list;
90
91
92 /* [ fetch_branch_list method ]
93 * Parameters (passed in struct form):
94 * SID                      - Session ID to work
    with
95 * project_name             - Project name to
    fetch branches for
96 *
97 * Returns:
98 * Array of branch name strings.
99 */
100 - (NSArray *)fetch_branch_list_for_project:(
    NSString *)project_name;
101
102
103 /* [ fetch_release method ]
104 * Parameters (passed in struct form):
105 * SID                      - Session ID
106 * project_name             - Project name
107 * branch_name              - Branch name
108 * version                  - Release version
    string
109 *
110 * Returns:
111 * Struct consisting of "version", "changes", "
    release_focus", and "hide_from_frontpage"
112 */
113 - (NSDictionary *)fetch_release_for_project:(
    NSString *)project_name branch:(NSString *)
    branch_name version:(NSString *)version;
114
115
116 /* [ login method ]
117 * Parameters (passed in struct form):
118 * username                 - Regular freshmeat
    username
119 * password                 - Regular freshmeat
    password
120 *
121 * Returns:
122 * Struct of SID, lifetime, and API Version
123 * SID: Session ID to be used in subsequent
    requests to the XML-RPC service
124 * Lifetime: Lifetime of the session ID in
    seconds
125 * API Version: API Version currently in use
126 */
127 - (void)login:(NSString *)username password:(
    NSString *)password;
128
129
130 /* [ logout method ]
131 * Parameters (passed in struct form):

```

```

132  * SID                      - Session ID to work with
133      terminate
134  * Returns:
135  * Struct of "OK" => "Logout successful." if
136      logout was successful
137  - (void)logout:(NSString *)SID;
138
139  /* [ publish_release method ]
140  * Parameters (passed in struct form):
141  * SID                      - Session ID to work with
142  * project_name             - Project name to submit a release for
143  * branch_name              - Branch name to submit a release for
144  * version                  - Version string of new release
145  * changes                  - Changes list, no HTML, character limit 600 chars
146  * release_focus            - Release focus ID of new release (see Appendix A)
147  * hide_from_frontpage      - Set to 'Y' if release is to be hidden from
148  * frontpage, everything else does not hide it
149  * license                  - Branch license
150  * url_homepage             - Homepage
151  * url_tgz                  - Tar/GZ
152  * url_bz2                  - Tar/BZ2
153  * url_zip                  - Zip
154  * url_changelog            - Changelog
155  * url_rpm                  - RPM package
156  * url_deb                  - Debian package
157  * url_osx                  - OS X package
158  * url_bsdport              - BSD Ports URL
159  * url_purchase             - Purchase
160  * url_cvs                  - CVS tree (cvsweb)
161  * url_list                  - Mailing list archive
162  * url_mirror               - Mirror site
163  * url_demo                 - Demo site
164  *
165  * Returns:
166  * Struct of "OK" => "submission successful"
167  *
168  * Notes:
169  * The "license" and "url_*" fields are optional and will be taken from the branch record if they are omitted from the submission. The 'hide_from_frontpage' option can be omitted an defaults to 'do not hide'.
170  *
171  * For convinience, we pass a dictionary to this method
172  */
173
174  - (void)publish_release:(NSDictionary *)newReleaseInfo;
175
176  /* [ withdraw_release method ]
177  * Parameters (passed in struct form):
178  * SID                      - Session ID
179  * project_name             - Project name
180  * branch_name              - Branch name
181  * version                  - Release version
182  *
183  * Returns:
184  * Struct of "OK" => "Withdraw successful."
185  */
186  - (void)withdraw_release_for_project:(NSString *)project_name
187      branch:(NSString *)branch_name
188      version:(NSString *)version;
189
190  @end
191
192  // Freshmeat.m
193  // Freshmint
194  // Created by Jean-Matthieu
195  //
196  #import "Freshmeat.h"
197
198  @implementation Freshmeat
199  - (id)init
200  {
201      self = [super init];
202      if (self) {
203          client = [[XMLRPCClient client:[NSURL URLWithString:@"http://freshmeat.net/xmlrpc"]] retain];
204          sessionDictionary = [[NSMutableDictionary alloc] init];
205          isConnected = NO;
206      }
207      return self;
208  }
209
210  - (void)dealloc
211  {
212      [sessionDictionary release];
213      [client release];
214      [super dealloc];
215  }
216
217  - (NSDictionary *)sessionDictionary
218  {
219      return sessionDictionary;
220  }
221
222  }
223
224  - (void)publish_release:(NSDictionary *)newReleaseInfo
225  {
226      return sessionDictionary;
227  }

```

```

229 - (void)autoLogout
228
230
231 {
232     [self logout:[sessionDictionary objectForKey:@"
233         SID "]];
234 }
235 - (BOOL)isConnected
236 {
237     return isConnected;
238 }
239
240 // Freshmeat methods invocation
241 - (NSArray *)fetch_available_licenses
242 {
243     NSArray *object;
244     object = [client invoke:@"
245         fetch_available_licenses" withArguments:[
246         NSArray arrayWithObject:@" "]];
247     return object;
248 }
249 - (NSDictionary *)fetch_available_release_foci
250 {
251     NSDictionary *object;
252     object = [client invoke:@"
253         fetch_available_release_foci"
254         withArguments:[NSArray arrayWithObject:@"
255         "]];
256     NSLog([object description]);
257     return object;
258 }
259 - (NSDictionary *)fetch_project_list
260 {
261     id object;
262     NSDictionary *myStruct = [NSDictionary
263         dictionaryWithObjects:[NSArray
264         arrayWithObjects:[sessionDictionary
265         objectForKey:@"SID"], nil] forKeys:[
266         NSArray arrayWithObjects:@"SID", nil]];
267     NSArray *args = [NSArray arrayWithObject:
268         myStruct];
269     object = [client invoke:@"fetch_project_list"
270         withArguments:args];
271
272     // Order projects
273    NSEnumerator *objEnumerator = [object
274         objectEnumerator];
275     id entry;
276     NSMutableDictionary *projectDictionary = [[[
277         NSMutableDictionary alloc] init]
278         autorelease];
279     while(entry = [objEnumerator nextObject]){
280         NSMutableDictionary *projectDetails;
281         if (nil == (projectDetails = [
282             projectDictionary objectForKey:[entry
283             objectForKey:@"projectname_full"]])){
284             projectDetails = [[[NSMutableDictionary
285                 alloc] init] autorelease];
286             NSArray *branches = [NSArray
287                 arrayWithArray:[self
288                 fetch_branch_list_for_project:[
289                 entry objectForKey:@"
290                 projectname_short "]]];
291             [projectDetails setObject:branches
292                 forKey:@"project.branches"];
293         }
294     }
295     NSArray *objects = [NSArray
296         arrayWithObjects:[entry objectForKey:@"
297         projectname_full"], [entry
298         objectForKey:@"project_version"],[
299         entry objectForKey:@"projectname_short
300         "], nil];
301     NSArray *keys = [NSArray arrayWithObjects:@"
302         project.name", @"project.version", @"
303         project.shortname", nil];
304     NSDictionary *projectInfo = [NSDictionary
305         dictionaryWithObjects:objects forKeys:
306         keys];
307     [projectDetails setObject:projectInfo
308         forKey:@"project.info"];
309     [projectDictionary setObject:projectDetails
310         forKey:[entry objectForKey:@"
311         projectname_full "]];
312     NSLog(@"%@", [projectDictionary description]);
313     [sessionDictionary setObject:projectDictionary
314         forKey:@"MyProjects"];
315     return nil;
316 }
317 - (NSArray *)fetch_branch_list_for_project:(
318     NSString *)project_name
319 {
320     id object;
321     NSDictionary *myStruct = [NSDictionary
322         dictionaryWithObjects:
323         [NSArray arrayWithObjects:[sessionDictionary
324         objectForKey:@"SID"], project_name, nil]
325         forKeys:[NSArray arrayWithObjects:@"SID", @"
326         project_name", nil]];

```

```

302                                     @selector(autoLogout) userInfo:nil repeats
303     NSArray *args = [NSArray arrayWithObject:                                     :NO];
        myStruct];                                     333
304     object = [client invoke:@"fetch_branch_list" 334     [self fetch_project_list];
        withArguments:args];                                     335
305     return object;                                     336 }
306 }                                     337
307                                     338 - (void)logout:(NSString *)SID
308 - (NSDictionary *)fetch_release_for_project:( 339 {
        NSString *)project_name branch:(NSString *) 340
        branch_name version:(NSString *)version 341     NSDictionary *myStruct = [NSDictionary
309 {                                     dictionaryWithObjects:[NSArray
310     id object;                                     arrayWithObject:SID] forKeys:[NSArray
311     NSDictionary *myStruct = [NSDictionary                                     arrayWithObject:@"SID"]];
        dictionaryWithObjects:[NSArray                                     342     NSArray *args = [NSArray arrayWithObject:
        arrayWithObjects:[sessionDictionary                                     myStruct];
        objectForKey:@"SID"], project_name,                                     343     [client invoke:@"logout" withArguments:args];
        branch_name,version , nil] forKeys:[NSArra344
        arrayWithObjects:@"SID" , @"project_name 345     [sessionDictionary removeAllObjects];
        , @"branch_name" , @"version" , nil]]; 346     isConnected = NO;
312     NSArray *args = [NSArray arrayWithObject: 347     NSLog(@"Freshmeat session terminated");
        myStruct];                                     348 }
313     object = [client invoke:@"fetch_release" 349
        withArguments:args];                                     350 - (void)publish_release:(NSDictionary *)
314     //NSLog(@"%@", [object description]);                                     newReleaseInfo
315     return object;                                     351 {
316 }                                     352     NSLog([newReleaseInfo description]);
317                                     353     NSArray *args = [NSArray arrayWithObject:
318 - (void)login:(NSString *)username password:( 354     newReleaseInfo];
        NSString *)password                                     355     [client invoke:@"publish_release" withArguments
319 {                                     :args];
320     id object = nil;                                     356 }
321     [sessionDictionary removeAllObjects];                                     357 - (void)withdraw_release_for_project:(NSString *)
322     NSDictionary *myStruct = [NSDictionary                                     project_name branch:(NSString *)branch_name
        dictionaryWithObjects:[NSArray                                     version:(NSString *)version
        arrayWithObjects:username , password , nil] 358 {
        forKeys:[NSArray arrayWithObjects:@" 359     NSDictionary *myStruct = [NSDictionary
        username" , @"password" , nil]];                                     dictionaryWithObjects:[NSArray
323     NSArray *args = [NSArray arrayWithObject:                                     arrayWithObjects:[sessionDictionary
        myStruct];                                     objectForKey:@"SID"], project_name ,
324     object = [client invoke:@"login" withArguments:                                     branch_name , version , nil] forKeys:[
        args];                                     NSArray arrayWithObjects:@"SID" , @"
325                                     project_name" , @"branch_name" , @"version
326                                     "]];
327     [sessionDictionary addEntriesFromDictionary: 360     NSArray *args = [NSArray arrayWithObject:
        object];                                     myStruct];
328     NSDate *date = [NSDate date];                                     361     [client invoke:@"withdraw_release"
329     [sessionDictionary setObject:date forKey:@"dat362
        "];                                     withArguments:args];
330     isConnected = YES;                                     363 }
331     // Autologout 5 sec before session ends 364
332     [NSTimer scheduledTimerWithTimeInterval:[[ 365     sessionDictionary objectForKey:@"Lifetime 366 @end
        sessionDictionary objectForKey:@"Lifetime 367
        "] intValue] - 5 target:self selector: 368

```



## Chapter 3

# Message Passing Programming with MPI

This chapter will present the basic concepts of message passing programming and discuss a design and an implementation of an Objective-C language binding for accessing some of MPI features from yet another popular programming language.

MPI stands for *Message Passing Interface*. It is a library of functions to be inserted in some source code to perform data communication between processes to implement some kind of parallel computing:

- A parallel computation consists of a number of processes, each working on some local data. A given process only accesses its local variables and cannot perform a direct access to the memory of another.
- Processes share their variables sending and receiving data through a network, a mechanism known as **message passing**.

This model is extremely general; any type of parallel computation can be cast in the message passing form allowing a programmer to distribute his tasks on a wide variety of platforms, should it be a multiprocessors computer or a network of single-processor

machines. In addition, explicit message passing provides more control over flow and data location within a parallel application than in the shared-memory model while improving its scalability and, by extension, its performance.

## 3.1 Introduction to the Message Passing Interface

### MPI History

It took about two years to define the *Message Passing Interface* standards. These were developed by sixty engineers from different organizations grouped as the MPI Forum. MPI-1 standard was completed in Spring of 1994, specifying the names, calling sequences, and results of subroutines and functions to be called from Fortran 77 and C, respectively. To ensure code portability, all implementations (even the partial ones) must conform to these rules in order to compile and run MPI programs on any platform that supports MPI standards. The detailed implementation of the library, in other words, what one puts inside each subroutines and functions, was left to the individual implementors who were thus free to produce optimized version of MPI for their machines.

An MPI-2 standard has also been defined, providing additional features to MPI-1, including tools for parallel Input/Output, C++ and Fortran 90 bindings and dynamic process management. Nowadays, some implementations of MPI have some of the MPI-2 standard, but the full MPI-2 is not available yet.

MPI-1 standard offers a large amount of features such as source code portability in order to compile and run MPI programs on a wide range of platforms and operating system, different types of communications, special routines for collective computations and the ability to handle user-defined data types and topologies. But some features are out of its scope. For example, there is no precise sequence defining the launch sequence of MPI programs; this generally depends on the implementation one is using. Moreover, there is no dynamic process management in MPI-1 meaning that the number of process is constant when the code is running. Finally, there is no special support neither for debugging nor for

Parallel-I/O, even though some of these missing features are addressed by MPI-2 standard.

## MPI Components and Architecture

MPI-1 offers about 150 functions for processes to communicate. Communications can be point-to-point or collectives, in a blocking or non-blocking way. It also provides a mechanism to gather processes inside a group and to realize communications within these groups of processes (Intra-communication). Another major feature of MPI is to identify communication contexts in order to isolate communications between specific groups (Inter-communication). The next paragraphs detail the various MPI-1 concepts.

## MPI Messages

An MPI Message wraps a collection of data to be sent or received. In order to send or receive a message one must specify the data memory address, the number of elements contained in the message and the message type. Table 3.1 lists common datatypes supported natively by MPI:

MPI Datatype	C type
MPI_BYTE	(none)
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_PACKED	(none)

Table 3.1: Basic MPI Datatypes

An envelope contains information on the process rank in the communication context, a message tag to identify the message, and the context where the message is to be sent.

## Processes and Groups

MPI-1 supposes that all processes were started *at-once* when the program started. There is no way to start a new process in MPI-1 and the way one starts a program generally depends on the MPI implementation one is using, as we said earlier. MPI-1 also suppose a Single Program Multiple Data programming style; each process manage its own memory access and controls its data flow by accepting incoming data or sending completed computation to a remote process. Moreover, one can create as many groups of processes as one wants when one initializes MPI, but these are statics and a process in a group cannot be moved to another.

## Communication Context and Communicators

Communicators are a major concept of MPI. A communicator *object* bundles a group of processes (processes that know each other) and a context of communication (an isolated communication line). It can also contain a virtual topology and other hidden arguments. Every MPI function has a communicator in its argument list. Inside a communicator, each process has a rank (from 0 to P-1 for P processes) and a process' rank can change from one communicator to another. Moreover, a context of communication can be used to isolate messages and it can be considered as a system tag. If a message is sent inside a communicator X, it cannot be received by a process inside a communicator Y.

Communicators can be modified in different ways. They can be duplicated (to obtain a new communicator with the same processes) or split in sub-communicators, and one can realize many operations on those groups of processes.

## Virtual Topologies

MPI supports the virtual topology concept. Many libraries, like BLACS, have had this feature that allows communications to take place in grids and sub-grids. MPI extends this notion allowing the use of cartesian topologies and irregular graphs, defined on-the-fly, by a user.

## Structured Data Types

In most message passing libraries prior to MPI, messages were only composed of data and data types defined by the implementation. Some libraries allow some extensions to that concept, like PVM *pack* and *unpack* functions but one could still not send a user-defined data structure. With MPI, one can build high-level data types using constructing functions bundled in the library. For example, one can define a *triangular-matrix* type and send a table of such type just as one would send any other variable.

## Point-to-point Communications

MPI has two subsets of point-to-point communications: the blocking one and the non-blocking (when sending and receiving data) one. Concerning the non-blocking communications, MPI offers an impressive set of waiting routines and a completion test. The non-blocking routines return a request number to be passed as an argument to the waiting and test routines.

## Collective Communications

MPI also offers a large set of collective communication functions. Indeed, one can synchronize data, broadcast variables among processes, scatter, or gather them. one can also realize some all-to-all operations, as well as global reduction (with various operators like *MPLSUM*, *MPLXOR*, ...), or scans. When one calls one of these routines, it executes with **all** communicator's processes; they are all blocking locally except the synchronization routine.

Those are the major features offered by the MPI-1 standard. As we stated previously, this standard has been updated a year after MPI-1 was finalized to add the next extensions to complete the library. In the next paragraphs, we will introduce the major MPI-2 features.

## Process Spawning

One of the first requests made by MPI-1 users was to be able to start processes inside

an MPI program, thanks to some kind of *spawning* function. Much like PVM *pvm\_spawn* routine syntax, this function allows the launch of other MPI processes from an MPI parent process. Thus, the newly created MPI processes have their own `MPI_COMM_WORLD`<sup>1</sup> and receive an inter-communicator to establish contact with their parent. Note that MPI is not dynamic like PVM and one obtains better results if one starts all processes at boot rather than one after the other.

### One-sided Communication

MPI-1 specifies that a two-process communication implies that those two agree to exchange data and that one process sends a message while the other is ready to receive it. MPI-2 one-sided communication extends this model allowing to *put* and *get* data to or from a distant process' memory. Processes must agree as well to exchange data with each other, since a memory access should be opened by one of the processes for the other to read or write to, but this is done on one side only. There are three types of one-sided communication in MPI:

1. **put** to write data inside a distant process' memory.
2. **get** to retrieve data from a distant process' memory.
3. **accumulate** to accumulate source and destination processes' data (using the same operators one would use for a reduction.)

These operations are non-blocking and a transfer ends when processes synchronize with each other. Several synchronization mechanisms, more or less collective, are also new and available to MPI-2.

### Parallel I/O

Parallel input-output is crucial to a large number of parallel applications, such as a distributed file system for example. Unfortunately, few software enable a useful and portable

---

<sup>1</sup>Default communicator globing all processes.

way to deal with it. Enters MPI I/O delivering a concurrent access to files from a set of processes. It can be considered as Unix input-output with extra features to deal with parallelism.

MPI-I/O offers equivalent functions to Unix routines `open`, `close`, `read`, `write` and `lseek`, and these functions use MPI to gain access to the file. MPI-I/O provides access to a distant file or memory: one can seek in a distant file just as one would do locally, create an individual or shared file pointer, execute non-blocking and/or collective operations on a file, adjust its settings to the distant file systems properties and have user-separated representation of a shared file.

MPI-2 has more capabilities such as extended collective communications, a way to create non-blocking routines, partial management of processes, C++, and Fortran 90 bindings, etc. A real-time interface to MPI (MPIRT<sup>2</sup>) has also been defined but no implementations are available yet. For reference, its main goals are:

1. Create integrated messaging, scheduling, and parallel programming API together with syntax and semantics to support the emerging computational hierarchies of node architectures and gigabit/s networks efficiently.
2. Expand the horizons of performance-portable real-time programming.
3. Support multiple real-time paradigms.
4. Enhance the performance of Messaging over MPI-1 and MPI-2.
5. Catalyze a new generation of portable parallel applications that require or benefit from the explicit use of time.

---

<sup>2</sup><http://www.mpirt.org>

## 3.2 MPI Operations

The next section will present the fundamental MPI operation. It is based and borrows parts of PACS' MPI Course [PACS01].

All MPI programs have the following general structure:

1. Include MPI header file
2. Variable declarations
3. Initialize the MPI environment
4. Do computation and MPI communication calls
5. Close MPI communications

The MPI header file contains MPI-specific definitions and function prototypes. Then, following the variable declarations, each process calls an MPI routine that initializes the message passing environment. All calls to MPI communication routines must come after this initialization. Finally, before the program ends, each process must call a routine that terminates MPI. No MPI routines may be called after the termination routine is called. Note that if any process does not reach this point during execution, the program will appear to hang.

### Initialization

The first MPI routine one must call in a program is `MPI_Init`. It establishes the MPI environment, returning an error code if the initialization failed. `MPI_Init` must be called only **once** in a program. The arguments to `MPI_Init` are the addresses of `argc` and `argv`, the variables that contain the command-line arguments for the program.

```
#include <mpi.h>

int main(int argc, char **argv)
{
    int error_code;
    error_code = MPI_Init(&argc, &argv);
```



```
...  
    return 0;  
}
```

### Communicators

A communicator is a handle representing a group of processes that can communicate with one another. Its name is required as an argument to all point-to-point or collective operations. For two processes to communicate, they must share a common communicator. A process is identified by a communicator's **rank**, a number from 0 to number of processes minus one in that communicator and it can belong to more than one communicator. The default communicator provided by MPI is `MPI_COMM_WORLD`. It is a communicator consisting of all processes. Using that communicator, every process can communicate with everyone and additional communicators, subsets of the available processes, can be defined.

**Getting a process rank:** `int MPI_Comm_rank(MPI_Comm comm, int *rank);`

**Getting the communicator size:** `int MPI_Comm_size(MPI_Comm comm, int *size);`

### Termination

The last MPI function to be called is `MPI_Finalize`. It is designed to clean up the MPI environment, freeing all MPI data structures, cancelling uncompleted operations, etc. It **must** be called by all processes or the program will appear to hang.

### Code Example

Program 3.2.1 is a simple first example of how to use MPI and a communicator to determine the rank of a process and the size of the default `MPI_COMM_WORLD` communicator. It outputs this information on each process' `stdout`.

```
1 #include <stdio.h>
```

## Program 3.2.1: MPI - Hello, World

```
2 #include <mpi.h>
3
4 int main(int argc, char **argv)
5 {
6     int error_code, prank, size;
7
8     // Initialize MPI
9     error_code = MPI_Init(&argc, &argv);
10
11    // Get the rank
12    MPI_Comm_rank(MPLCOMM_WORLD, &prank);
13    // Get nbr of processes
14    MPI_Comm_size(MPLCOMM_WORLD, &size);
15
16    printf("Process %d of %d: Hello MPI!\n", prank, size);
17
18    // Terminate MPI
19    MPI_Finalize();
20
21    return 0;
22 }
23
24 // Output
25 // Machine 1 : Process 0 of 3: Hello MPI!
26 // Machine 2 : Process 1 of 3: Hello MPI!
27 // Machine 3 : Process 2 of 3: Hello MPI!
```

### 3.2.1 Point-to-Point Communications

MPI provides facilities for processes to communicate with each other by sending and receiving messages. They fall into two categories: blocking communication that hangs the process until the communication is completed (creating a possibility of deadlock) and non-blocking communication, a two-step method to avoid deadlocks.

#### Blocking Communications

##### Sending a Message

MPI offers `MPI_Send` to send a message from one process to another. The message body

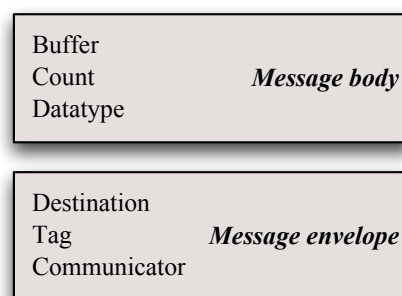


Figure 3.1: `MPI_Send` Arguments.

contains the data to be sent: `count` items of type `datatype`. The message envelope tells where to send it. In addition, an error code is returned.

`MPI_Send` C binding:

```
int MPI_Send(void *buf, int count, MPI_Datatype dtype,
             int dest, int tag, MPI_Comm comm);
```

##### Receiving a Message

`MPI_Recv` takes a set of arguments similar to `MPI_Send`: The message envelope defines

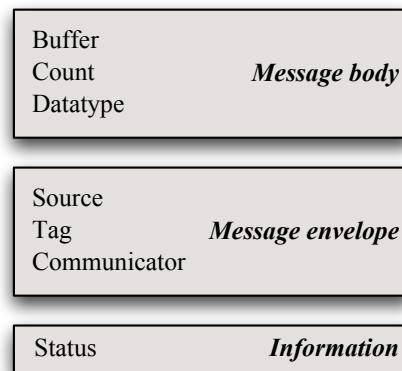


Figure 3.2: MPI\_Recv Arguments.

which message can be received. The source, tag, and communicator must match to a pending message in order for the message to be received. Note that one can use wildcard values to receive message from any source (`MPI_ANY_SOURCE`) or with any tag (`MPI_ANY_TAG`).

The message body arguments specify what type of message is to be received, what length it is expected to be and where to store it.

This routine returns an error code along with an `MPI_Status` status structure to inform of the operation's success.

`MPI_Recv` C binding:

```
int MPI_Recv(void *buf, int count, MPI_Datatype dtype,  
             int source, int tag, MPI_Comm comm,  
             MPI_Status *status);
```

### Code Example

Program 3.2.2 illustrates a simple MPI program that sends and receives data between processes in `MPI_COMM_WORLD`.

```
1 #include <stdio.h>
```

## Program 3.2.2: Blocking Send and Receive

```
2 #include <mpi.h>
3 int main (int argc , char **argv)
4 {
5     int myrank;
6     MPI_Status status;
7     double a[100];
8
9     /* Initialize MPI */
10    MPI_Init(&argc , &argv);
11
12    /* Get rank */
13    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
14
15
16    if (myrank == 0) {
17        /* Send a message */
18        MPI_Send(a, 100, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD);
19    } else if (myrank == 1) {
20        /* Receive a message */
21        MPI_Recv(a, 100, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD, &status);
22    }
23
24    MPI_Finalize(); /* Terminate MPI */
25    return 0;
26 }
```

## Non-blocking Communications

The non-blocking interface to send and receive requires two calls per communication operation: one call to initiate the operation, and a second call to complete it. Initiating a send operation is called posting a send. Initiating a receive operation is called posting a receive. Once a send or receive operation has been posted, MPI provides two distinct ways of completing it. A process can test to see if the operation has completed, without blocking on the completion. Alternately, a process can wait for the operation to complete.

Non-blocking send and receive routines all return request handles, which are used to identify the operation posted by the call.

### Posting a Send

The C binding to post a send is very close to `MPI_Send`. It has one more output argument: a request handle to test the post.

`MPI_Isend` C binding:

```
int MPI_Isend(void *buf, int count, MPI_Datatype dtype,
              int dest, int tag, MPI_Comm comm,
              MPI_Request *request);
```

**Note:** Another call to MPI is required to complete the send operation posted by this routine.

### Posting a Receive

The C binding to post a receive is very close to `MPI_Recv`. The last argument is change from `MPI_Status` to `MPI_Request`, a request handle to test the operation.

`MPI_Irecv` C binding:

```
int MPI_Irecv(void *buf, int count, MPI_Datatype dtype,
              int source, int tag, MPI_Comm comm,
```

```
MPI_Request *request);
```

**Note:** Another call to MPI is required to complete the receive operation posted by this routine.

### Completing a Non-blocking Operation

There are two ways to complete a non-blocking operation. one can either wait for the operation to complete with `MPI_Wait`, a blocking routine, or test the operation with `MPI_Test`, a non-blocking routine.

`MPI_Wait` C binding, returns a status after completion:

```
int MPI_Wait( MPI_Request *request, MPI_Status *status);
```

`MPI_Test` C binding

```
int MPI_Test( MPI_Request *request, int *completed, MPI_Status *status);
```

The output parameter `completed` is **true** if the send or the receive has completed. `status` is undefined if `completed` is equal to **false**, otherwise, it returns the operation status just like `MPI_Wait`.

**Code Example** Program 3.2.3 illustrates a simple MPI program that sends and receives data between processes in `MPI_COMM_WORLD` in a non-blocking way.

Program 3.2.3: Non-blocking Send and Receive

```
1 #include <stdio.h>
2 #include <mpi.h>
3 int main (int argc , char **argv)
4 {
5     int myrank;
6     MPI_Request request;
```

```
7  MPI_Status status;
8  double a[100], b[100];
9
10 /*Initialize MPI*/
11 MPI_Init(&argc, &argv);
12
13 /*Get rank*/
14 MPI_Comm_rank(MPLCOMM_WORLD, &myrank);
15
16
17 if (myrank == 0){
18     /*Post a receive, send a message, then wait*/
19     MPI_Irecv(b, 100, MPLDOUBLE, 1, 19, MPLCOMM_WORLD, &request);
20     MPI_Send(a, 100, MPLDOUBLE, 1, 17, MPLCOMM_WORLD);
21     MPI_Wait(&request, &status);
22 } else if (myrank == 1) {
23     /*Receive a message*/
24     MPI_Irecv(b, 100, MPLDOUBLE, 0, 17, MPLCOMM_WORLD, &request);
25     ;
26     MPI_Send(a, 100, MPLDOUBLE, 0, 19, MPLCOMM_WORLD);
27     MPI_Wait(&request, &status);
28 }
29 MPI_Finalize(); /*Terminate MPI*/
30 return 0;
31 }
```

### MPI Send Modes

MPI provides four send modes:



### 1. Standard Mode Send

When MPI executes a standard mode send, one of two things happens. Either the message is copied into an MPI internal buffer and transferred asynchronously to the destination process, or the source and destination processes synchronize on the message. The MPI implementation is free to choose (on a case-by-case basis) between buffering and synchronizing, depending on message size, resource availability, etc.

### 2. Synchronous Mode Send

When a synchronous mode send operation is completed, the sending process may assume the destination process has begun receiving the message. The destination process need not be done having finished receiving the message, but it must have begun receiving the message.

### 3. Ready Mode Send

It requires that a matching receive has already been posted at the destination process before ready mode send is called.

### 4. Buffer Mode Send

It requires MPI to use buffering.

Table 3.2 list the calling sequences for the various send modes.

Send Mode	Blocking Function	Non-blocking Function
Standard	MPI_Send	MPI_Isend
Synchronous	MPI_Ssend	MPI_Issend
Ready	MPI_Rsend	MPI_Irsend
Buffer	MPI_Bsend	MPI_Ibsend

Table 3.2: Send Modes Calling Routines

## 3.2.2 Collective Communications

Collective communication involves the sending and receiving of data among processes. Collective communication routines transmit data among all processes in a group. It is

important to note that collective communication calls do not use the tag mechanism of send/receive for associating calls. Rather they are associated by order of program execution. Thus, the user must ensure that all processors execute the same collective communication calls and execute them in the same order.

MPI provides the following collective communication routines:

- Barrier synchronization.
- Broadcast from one process to all other process.
- Global reduction operation.
- Gather data from all processes to one process.
- Scatter data from one process to all processes.

### **Barrier Synchronization**

Imagine a root process reads data before sending the complete data set to other processes. The root process cannot proceed to the send operation before all data has been received and the other processes must wait until the I/O operation is completed and the data are moved.

`MPI_Barrier` routine blocks the calling process until all group processes have called the function. When it returns, all processes are synchronized at the barrier.

`MPI_Barrier` C binding:

```
int MPI_Barrier (MPI_Comm comm);
```

### **Broadcast Operation**

The `MPI_Bcast` routine enables one to copy data from the memory of the root process to the same memory locations for other processes in the communicator. `MPI_Bcast` C binding:

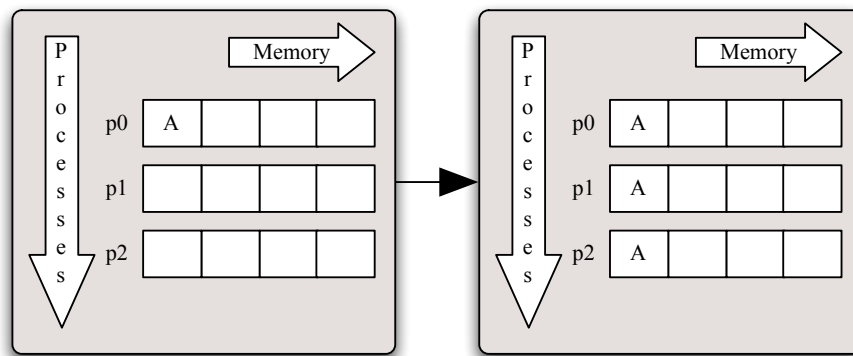


Figure 3.3: A Simple Broadcast Operation

```
int MPI_Bcast (void* buffer, int count, MPI_Datatype datatype,
               int rank, MPI_Comm comm) ;
```

### Reduce Operation

The `MPI_Reduce` routine enables one to collect data from each process, reduce this data to a single value and store the reduced result on the root process. Figure 3.4 shows a reduction example that sums the value of **A** on each process and stores the result in **X** in the root process.

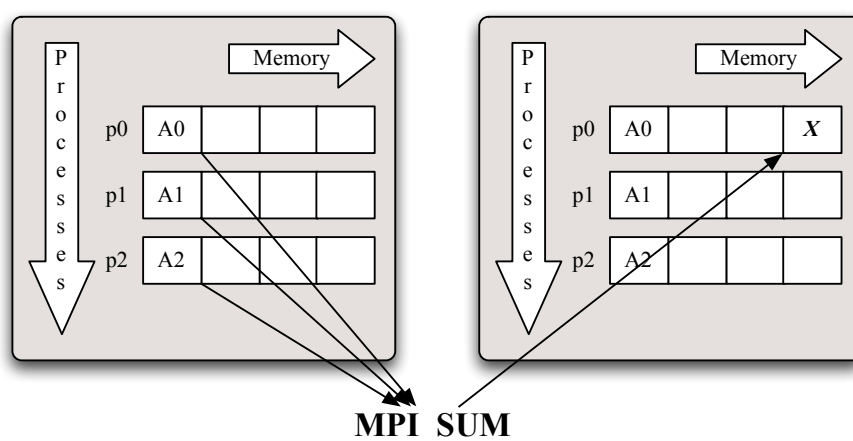


Figure 3.4: A Simple Reduce Operation

MPI\_Reduce C binding:

```
int MPI_Reduce (void* send_buffer, void *recv_buffer, int count,
               MPI_Datatype datatype, MPI_Op operation,
               int rank, MPI_Comm comm);
```

MPI offers predefined operations, listed in table 3.3, available to MPI\_Reduce.

Operation	Description
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical AND
MPI_BAND	Bit-wise AND
MPI_LOR	Logical OR
MPI_BOR	Bit-wise OR
MPI_LXOR	Logical XOR
MPI_BXOR	Bit-wise XOR
MPI_MINLOC	Rank of the process containing the minimum value.
MPI_MAXLOC	Rank of the process containing the maximum value

Table 3.3: MPI\_Reduce Operators

## Gather Operation

The MPI\_Gather routine is an *all-to-one* communication. MPI\_Gather has the same arguments as the matching scatter routines. The receive arguments are only meaningful to the root process.

When MPI\_Gather is called, each process (including the root process) sends the contents of its send buffer to the root process. The root process receives the messages and stores them in rank order.

MPI\_Gather C binding:

```
int MPI_Gather (void *send_buffer, int send_count, MPI_datatype send_type,
               void *recv_buffer, int recv_count, MPI_Datatype recv_type,
               int rank, MPI_Comm comm);
```

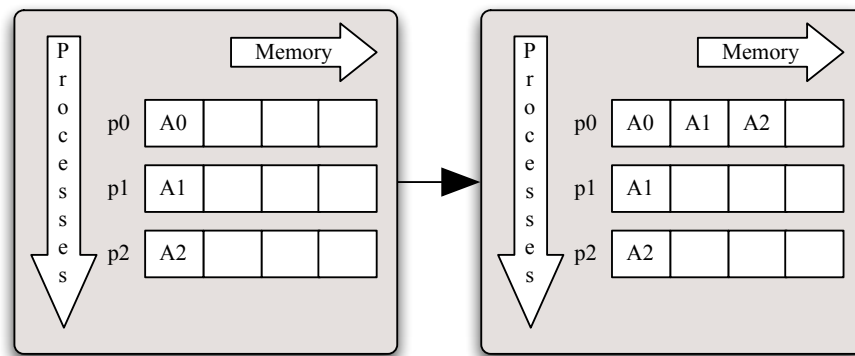


Figure 3.5: A Simple Gather Operation

In the previous example, after the data are gathered into processor 0, one could then `MPI_Bcast` the gathered data to all of the other processors. It is more convenient and efficient to gather and broadcast with the single `MPI_Allgather` operation.

### Scatter Operation

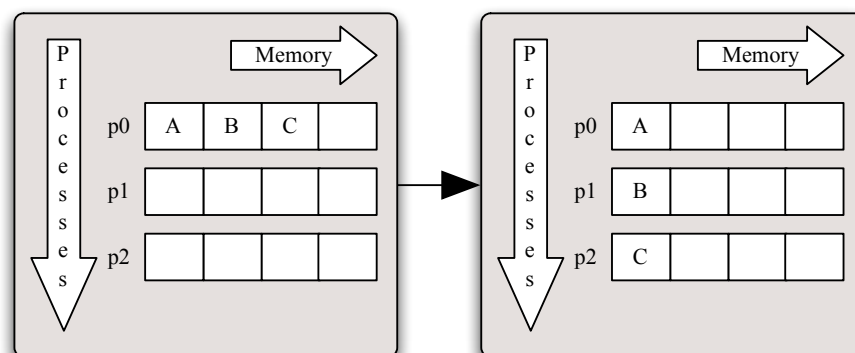


Figure 3.6: A Simple Scatter Operation

The `MPI_Scatter` routine is a *one-to-all* communication. Different data are sent from the root process to each process (in rank order).

When `MPI_Scatter` is called, the root process breaks up a set of contiguous memory loca-

tions into equal chunks and sends one chunk to each processor.

MPI\_Scatter C binding:

```
int MPI_Scatter(void *send_buffer, int send_count, MPI_datatype send_type,  
               void *recv_buffer, int recv_count, MPI_Datatype recv_type,  
               int rank, MPI_Comm comm);
```

# Chapter 4

## MPIObjC: A Language Binding to MPI

We covered in the previous section the fundamentals of MPI. We now know how to instantiate the environment and how to deal with message passing. The last topic for this paper suggests an MPI language binding for Objective-C based on MacMPI [MMPI04]. MPIObjC is intended to wrap the common MPI C calls inside a framework composed of three central objects: *MPIInstance*, *MPIComm* and *MPIRequest*. Following the class description, one will find examples that illustrate the use of MPIObjC.

### 4.1 MPIInstance Class Reference

```
#import <MPIInstance.h>
```

Collaboration diagram for MPIInstance:

#### Public Member Functions

- (void) MPIFinalize
- (void) MPIAbort
- (BOOL) MPIInitialized

- (NSNumber \*) MPIWTime
- (NSNumber \*) MPIWTick
- (NSString \*) MPIGetProcessorName
- (MPIComm \*) commWorld
- (void) setCommWorld:
- (MPI\_Status) status
- (void) setStatus:
- (NSMutableDictionary \*) requestDictionary
- (void) fixMacMPI

## Static Public Member Functions

- (id) mpiWith:argc:
- (MPIInstance \*) getInstance

### 4.1.1 Detailed Description

MPIInstance. An instance class for MPI operation.

### 4.1.2 Member Function Documentation

#### - (MPIComm \*) commWorld

##### Returns:

Returns MPI World Communicator.

#### - (void) fixMacMPI

A fix for Cocoa MacMPI applications. Because MacMPI expects to read the nodelist.ip file using fopen, and this file is generally placed in the same directory where the Cocoa bundle application resides, it is necessary to set the default directory to the directory of the application very early in the code (before calling MPI\_Init).



**+ (MPIInstance \*) getInstance**

Get MPI Instance. A utility function to access the MPI environment. It is better to initialize the MPI environment before calling this function.

**Returns:**

Returns a previously instantiated MPIInstance

**- (void) MPIAbort**

Abort MPI. Terminates MPI execution environment.

**- (void) MPIFinalize**

Finalize MPI. Terminates MPI execution environment. All processes must call this routine before exiting. One does not need to call this method, MPI\_Finalize() is called automatically when one releases MPIInstance.

**- (NSString \*) MPIGetProcessorName**

MPI Processor name. Gets the name of the processor.

**Returns:**

Returns the name of the processor as a NSString.

**- (BOOL) MPIInitialized**

Check MPI state. Indicates whether MPI\_Init has been called.

**Returns:**

Returns TRUE if MPI\_Init has been called, FALSE otherwise.

**+ (id) mpiWith: (int \*) (char \*\*\*) argv**

Instantiate MPI. Initialize the MPI environment. Always call this function !

**Parameters:**

*argc* : from the command line.

*argv* : from the command line.

**Returns:**

Returns an MPI Instance

- (NSNumber \*) **MPIWTick**

Returns the resolution of MPI\_Wtime.

**Returns:**

Time in seconds of resolution of MPI\_Wtime

- (NSNumber \*) **MPIWTime**

MPI Time. Returns an elapsed time on the calling processor.

**Returns:**

Time in seconds since an arbitrary time in the past.

- (NSMutableDictionary \*) **requestDictionary**

Dictionary of MPI non blocking request. Keys are message tags.

- (void) **setCommWorld:** (MPIComm \*) *aComm*

Set MPIInstance communicator.

**Parameters:**

*An* MPIComm object.

- (void) **setStatus:** (MPI\_Status) *status*

Set MPI environment status.

**Parameters:**

*status*: A MPI\_Status tag.

**- (MPI\_Status) status**

MPI Environment status. Get MPI environment status.

**Returns:**

Returns MPI Status.

## 4.2 MPIComm Class Reference

```
#import <MPIComm.h>
```

### Public Member Functions

- (id) initWithCommunicator:
- (NSNumber \*) MPICommSize
- (NSNumber \*) MPICommRank
- (MPIComm \*) MPICommDup
- (MPIComm \*) MPICommSplit:andKey:
- (void) MPICommFree
- (void) MPISend:ofSize:ofType:toProcess:withTag:
- (void) MPIRecv:ofSize:type:from:withTag:
- (void) MPISendRecv:ofSize:ofType:toProcess:withTag:outMessage:outSize:type:from:withTag:
- (NSNumber \*) MPIGetCount:
- (void) MPIISend:ofSize:ofType:toProcess:withTag:
- (void) MPIIRecv:ofSize:type:from:withTag:
- (void) MPIBarrier
- (void) MPIBcast:ofSize:ofType:rank:
- (void) MPIGather:ofSize:ofType:outMessage:outSize:outType:rank:
- (void) MPIGatherv:ofSize:ofType:outMessage:outSize:displacement:outType:rank:
- (void) MPIAllGather:ofSize:ofType:outMessage:outSize:outType:
- (void) MPIScatter:ofSize:ofType:outMessage:outSize:outType:rank:
- (void) MPIScatterv:ofSize:displacement:ofType:outMessage:outSize:outType:rank:
- (void) MPIReduce:outMessage:ofSize:ofType:withOp:rank:
- (void) MPIAllReduce:outMessage:ofSize:ofType:withOp:

- (void) MPIReduceScatter:outMessage:outSize:ofType:withOp:
- (void) MPIScan:outMessage:ofSize:ofType:withOp:
- (void) MPIAllToAll:ofSize:ofType:outMessage:outSize:outType:
- (void) MPIAllToAllv:ofSize:inDispls:ofType:outMessage:outSize:outDispls:outType:

### 4.2.1 Detailed Description

MPIComm - MPI Communication object.

### 4.2.2 Member Function Documentation

- (id) initWithCommunicator: (MPI\_Comm) *aComm*

Initiate a MPIComm object.

**Parameters:**

***aComm:*** A communicator, such as MPI\_COMM\_WORLD, MPI\_COMM\_SELF, MPI\_COMM\_NULL ...

**Returns:**

Returns an MPIComm instance; a communicator for your MPI environment.

- (void) MPIAllGather: (void \*) *message*(int) *count*(MPI\_Datatype) *type*(void \*) *outBuffer*(int) *outSize*(MPI\_Datatype) *outType*

All-gather operation. Gather messages from all process in all proccess in the communicator.

**Parameters:**

***message:*** Message to send.

***count:*** Number of elements in the sent message.

***type:*** The sent message type (ie: MPI\_CHAR, MPI\_INT ...).

***outBuffer:*** A buffer to store the received message.

***outSize:*** Number of elements in the received buffer.

***outType:*** The received message type (ie: MPI\_CHAR, MPI\_INT ...).

**Returns:**

- (void) MPIAllReduce: (void \*) *message*(void \*) *outBuffer*(int)  
*size*(MPI\_Datatype) *type*(MPI\_Op) *operation*

Reduction computation. Combines values from all processes and distributes the result back to all processes.

**Parameters:**

***message:*** Message to send.

***outBuffer:*** A buffer to store the received message.

***size:*** Number of elements in the received message.

***type:*** The message datatype.

***operation:*** The reduction operation.

**Returns:**

- (void) MPIAllToAll: (void \*) *message*(int) *inSize*(MPI\_Datatype)  
*inType*(void \*) *outBuffer*(int) *outSize*(MPI\_Datatype) *outType*

Sends data from all to all processes.

**Parameters:**

***message:*** Message to send.

***inSize:*** Number of elements in the sent message.

***inType:*** Type of sent message.

***outBuffer:*** A buffer to store the received message.

***outSize:*** Size of the received message.

***outType:*** Type of received message.

**Returns:**

- (void) MPIAllToAllv: (void \*) *message*(int \*) *inSize*(int \*)  
*inDispls*(MPI\_Datatype) *inType*(void \*) *outBuffer*(int \*) *outSize*(int \*)  
*outDispls*(MPI\_Datatype) *outType*

Sends data from all to all processes, with a displacement.

**Parameters:**

***message:*** Message to send.

***inSize:*** Number of elements in the sent message.

***inDispls:*** Displacement in sent message.

***inType:*** Type of sent message.

***outBuffer:*** A buffer to store the received message.

***outSize:*** Size of the received message.

***outDispls:*** Displacement in received message.

***outType:*** Type of received message.

**Returns:**

- (void) MPIBarrier

Process Synchronization performs a barrier synchronization among all processes in the communicator.

- (void) MPIBcast: (void \*) *message*(int) *size*(MPI\_Datatype) *type*(int)  
*rootProcess*

Message Broadcast. Broadcast a message to all process in the communicator world.

**Parameters:**

- message:*** Message to send.
- size:*** Number of elements in the buffer.
- type:*** The message type (ie: MPI\_CHAR, MPI\_INT ...).
- rootProcess:*** Rank of process with message to broadcast.

**- (MPIComm \*) MPICommDup**

Communicator duplicator. Duplicates an existing communicator with all its cached information.

**Returns:**

A duplicated MPIComm object.

**- (void) MPICommFree**

Free a communicator. Marks the communicator object for deallocation

**- (NSNumber \*) MPICommRank**

Process rank. Determines the rank of the calling process in the communicator.

**Returns:**

The rank of the calling process as a NSNumber.

**- (NSNumber \*) MPICommSize**

Communicator size. Determines the size of the group associated with a communicator.

**Returns:**

The size of the group as a NSNumber.

**- (MPIComm \*) MPICommSplit: (int) *color*(int) *aKey***

Communicator splitter Creates new communicators based on colors and keys.



**Parameters:**

**color:** An integer to specify the color, control of subset assignment. The color must be non-negative or MPI\_UNDEFINED.

**aKey:** An integer to specify the key, control of rank assignment.

**Returns:**

A new MPIComm instance.

- (void) MPIGather: (void \*) *message*(int) *count*(MPI\_Datatype) *type*(void \*)  
*outBuffer*(int) *outSize*(MPI\_Datatype) *outType*(int) *rootProcess*

Basic Message gathering. Gather messages from all process in the communicator.

**Parameters:**

**message:** Message to send.

**count:** Number of elements in the sent message.

**type:** The sent message type (ie: MPI\_CHAR, MPI\_INT ...).

**outBuffer:** A buffer to store the received message.

**outSize:** Number of elements in the received buffer.

**outType:** The received message type (ie: MPI\_CHAR, MPI\_INT ...).

**rootProcess:** Rank of gathering process.

**Returns:**

- (void) MPIGatherv: (void \*) *message*(int) *count*(MPI\_Datatype) *type*(void \*)  
*outBuffer*(int \*) *outSize*(int \*) *displs*(MPI\_Datatype) *outType*(int)  
*rootProcess*

More complex message gathering. Gather message with variable length from all process in the communicator.

**Parameters:**

**message:** Message to send.

***count:*** Number of elements in the sent message.

***type:*** The sent message type (ie: MPI\_CHAR, MPI\_INT ...).

***outBuffer:*** A buffer to store the received message.

***outSize:*** Number of elements in the received buffer.

***displs:*** Displacement in received message of elements gathered from all processes.

***outType:*** The received message type (ie: MPI\_CHAR, MPI\_INT ...).

***rootProcess:*** Rank of gathering process.

**Returns:**

- (NSNumber \*) **MPIGetCount:** (MPI\_Datatype) *forType*

Get the number of received elements.

**Parameters:**

***forType:*** The message type (ie: MPI\_CHAR, MPI\_INT ...)

**Returns:**

The number of received elements as a NSNumber.

- (void) **MPIRecv:** (void \*) *outBuffer*(int) *messageSize*(MPI\_Datatype)  
*type*(int) *src*(int) *tag*

Receiving data. Basic receive from a process.

**Parameters:**

***outBuffer:*** A buffer to store the received message.

***messageSize:*** The expected message size.

***type:*** The message type (ie: MPI\_CHAR, MPI\_INT ...).

***src:*** Rank of the sending process.

***tag:*** Message tag.

**Returns:**

- (void) MPIISend: (void \*) *message*(int) *messageSize*(MPI\_Datatype)  
*type*(int) *dest*(int) *tag*

Sending data. Performs a basic send.

**Parameters:**

*message*: Message to send.

*messageSize*: Number of elements in the sent message.

*type*: The message type (ie: MPI\_CHAR, MPI\_INT ...).

*dest*: Rank of process to send the data to (integer).

*tag*: Message tag.

- (void) MPIRecv: (void \*) *outBuffer*(int) *messageSize*(MPI\_Datatype)  
*type*(int) *src*(int) *tag*

Receiving data. Basic receive from a process.

**Parameters:**

*outBuffer*: A buffer to store the received message.

*messageSize*: The expected message size.

*type*: The message type (ie: MPI\_CHAR, MPI\_INT ...).

*src*: Rank of the sending process.

*tag*: Message tag.

**Returns:**

- (void) MPIReduce: (void \*) *message*(void \*) *outBuffer*(int)  
*size*(MPI\_Datatype) *type*(MPI\_Op) *operation*(int) *rank*

Reduction computation. Reduces values on all processes to a single value.

**Parameters:**

*message*: Message to send.

***outBuffer:*** A buffer to store the received message.

***size:*** Number of elements in the received message.

***type:*** The message datatype.

***operation:*** The reduction operation.

***rank:*** Rank of the reducing process.

**Returns:**

- (void) MPIReduceScatter: (void \*) *sMessage*(void \*) *outBuffer*(int \*)  
*rSize*(MPI\_Datatype) *type*(MPI\_Op) *operation*

Reduces and scatters a message. Combines values and scatters the results.

**Parameters:**

***message:*** Message to send.

***outBuffer:*** A buffer to store the received message.

***rSize:*** Number of elements in the received message.

***type:*** The message datatype.

***operation:*** *The* reduction operation.

**Returns:**

- (void) MPIScan: (void \*) *message*(void \*) *outBuffer*(int)  
*size*(MPI\_Datatype) *type*(MPI\_Op) *operation*

Partial reduction computation Computes the scan (partial reductions) of data on a collection of processes.

**Parameters:**

***message:*** Message to send.

***outBuffer:*** A buffer to store the received message.

**size:** Number of elements in the received message.

**type:** The message datatype.

**operation:** *The* reduction operation.

**Returns:**

- (void) MPIScatter: (void \*) *message*(int) *count*(MPI\_Datatype) *type*(void \*)  
*outBuffer*(int) *outSize*(MPI\_Datatype) *outType*(int) *rootProcess*

Basic Message scattering. Distribute individual messages from root to each process in the communicator.

**Parameters:**

**message:** Message to send.

**count:** Number of elements in the sent message.

**type:** The sent message type (ie: MPI\_CHAR, MPI\_INT ...).

**outBuffer:** A buffer to store the received message.

**outSize:** Number of elements in the received buffer.

**outType:** The received message type (ie: MPI\_CHAR, MPI\_INT ...).

**rootProcess:** Rank of scattering process.

**Returns:**

- (void) MPIScatterv: (void \*) *message*(int \*) *count*(int \*)  
*displs*(MPI\_Datatype) *type*(void \*) *outBuffer*(int) *outSize*(MPI\_Datatype)  
*outType*(int) *rootProcess*

Complex message scattering. Distributes individual messages from root to each process in the communicator. Messages can have different sizes and displacements.

**Parameters:**

- message***: Message to send.
- count***: Number of elements in the sent message.
- displs***: Displacement in sent message.
- type***: The sent message type (ie: MPI\_CHAR, MPI\_INT ...).
- outBuffer***: A buffer to store the received message.
- outSize***: Number of elements in the received buffer.
- outType***: The received message type (ie: MPI\_CHAR, MPI\_INT ...).
- rootProcess***: Rank of scattering process.

**Returns:**

- (void) MPISend: (void \*) *message*(int) *messageSize*(MPI\_Datatype)  
*type*(int) *dest*(int) *tag*

Sending data. Performs a basic send.

**Parameters:**

- message***: Message to send.
- messageSize***: Number of elements in the sent message.
- type***: The message type (ie: MPI\_CHAR, MPI\_INT ...).
- dest***: Rank of process to send the data to (integer).
- tag***: Message tag.

- (void) MPISendRecv: (void \*) *message*(int) *sMessageSize*(MPI\_Datatype)  
*sType*(int) *dest*(int) *sTag*(void \*) *outBuffer*(int) *rMessageSize*(MPI\_Datatype)  
*rType*(int) *src*(int) *rTag*

Send and receive a message. This method sends and receive a message.

**Parameters:**

***message:*** Message to send.

***sMessageSize:*** Number of elements in the sent message.

***sType:*** The message type (ie: MPI\_CHAR, MPI\_INT ...)

***dest:*** Rank of process to send the data to (integer).

***sTag:*** Message tag.

***outBuffer:*** A buffer to store the received message.

***rMessageSize:*** The expected message size.

***rType:*** The message type (ie: MPI\_CHAR, MPI\_INT ...)

***src:*** Rank of the sending process.

***rtag:*** Message tag.

**Returns:**

## 4.3 MPIRequest Class Reference

```
#import <MPIRequest.h>
```

### Public Member Functions

- (id) initWithRequest:
- (void) MPIWait
- (int) MPITest
- (void) MPIRequestFree

#### 4.3.1 Detailed Description

MPIRequest - MPI Request object.

#### 4.3.2 Member Function Documentation

##### - (id) initWithRequest: (MPI\_Request) *aRequest*

MPIRequest Constructor. Initiate a MPIRequest object with request aRequest.

**Parameters:**

***aRequest:*** A MPI\_Request handle.

**Returns:**

Returns an instantiated MPIRequest object.

##### - (void) MPIRequestFree

Frees a request. Frees a communication request object.

##### - (int) MPITest

Tests a non-blocking operation. Tests for the completion of a send or receive.

**Returns:**

True if operation competed. Sets the status member of the MPIRequest object.



**- (void) MPIWait**

Completes a non-blocking operation. MPIWait waits for a MPI send or receive to complete.

## 4.4 Coding with MPIObjC

Program 4.4.1 is a basic example of MPIObjC use. It initiates the MPI environment and gets the processor name before printing it on each process.

Program 4.4.1: A Simple MPIObjC Program.

```
1
2 #import <Foundation/Foundation.h>
3 #import <MPIObjC/MPIObjC.h>
4
5 int main (int argc , char **argv)
6 {
7     NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
8     int numprocs , myrank;
9
10    MPIInstance *mpi = [MPIInstance mpiWith:&argc :&argv];
11
12    NSString *pname = [mpi MPIGetProcessorName];
13    NSLog(pname);
14
15    [mpi release];
16
17    [pool release];
18    return 0;
19
20 }
```

Program 4.4.2 is a simple communicator example. It uses the default MPI communicator, gets its size and the rank of the calling process.

Program 4.4.2: Using the MPIComm Object.

```
1  /*
2   *  MPICommTest.c
3   *  MPIObjC
4   *
5   *  Test for the MPIComm object
6   *
7   */
8
9  #import <Foundation/Foundation.h>
10 #import <MPIObjC/MPIObjC.h>
11
12 int main (int argc , char **argv)
13 {
14     NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
15
16     MPIInstance *mpi = [MPIInstance mpiWith:&argc :&argv];
17
18     MPIComm *mCommWorld = [mpi commWorld];
19     NSNumber *size = [mCommWorld MPICommSize];
20     NSNumber *rank = [mCommWorld MPICommRank];
21
22     NSLog(@" Process %@ of %@." , rank+1, size);
23
24
25     [mpi release];
26
27     [pool release];
28     return 0;
29 }
```

Program 4.4.3 receives on the root process the rank number of every other process in the ring. This is yet another example of MPIComm capabilities.

Program 4.4.3: More Fun with MPIComm

```
1 #import <Foundation/Foundation.h>
2 #import <MPIObjC/MPIObjC.h>
3
4 int main (int argc, char **argv)
5 {
6     NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
7
8     MPIInstance *mpi = [MPIInstance mpiWith:&argc :&argv];
9
10    MPIComm *mCommWorld = [mpi commWorld];
11    NSNumber *size = [mCommWorld MPICommSize];
12    int rank = [[mCommWorld MPICommRank] intValue];
13    NSNumber *aNumber = [NSNumber numberWithInt:rank+1];
14    int i=1;
15    if (rank == 0) {
16        for (i = 1; i < [size intValue]; i++)
17        {
18            int recv = 0;
19            [mCommWorld MPIRecv:&recv ofSize:1 type:MPLINT from:i
20             withTag:1];
21            NSLog(@"%d", recv);
22        }
23    } else {
24        int rank = [aNumber intValue];
25        [mCommWorld MPISend:&rank ofSize:1 ofType:MPLINT toProcess:0
26         withTag:1];
```

```

25     }
26
27
28     [mpi release];
29
30     [pool release];
31     return 0;
32 }

```

Program 4.4.4 is a parallel implementation of the sieve of Eratosthenes counting the number of prime numbers between 2 and 60. This example is based on Michael J. Quinn's sieve in *Parallel Programming in C with MPI and OpenMP*.

Program 4.4.4: The Sieve of Eratosthenes

```

1  /* Sieve of Eratosthenes */
2
3  #import <Foundation/Foundation.h>
4  #import <MPIObjC/MPIObjC.h>
5
6  #import <math.h>
7  #define BLOCKLOW(id,p,n) ((id)*(n)/(p))
8  #define BLOCKHIGH(id,p,n) (BLOCKLOW((id)+1,p,n) - 1)
9  #define BLOCK_SIZE(id,p,n) ((BLOCKHIGH(id,p,n) - BLOCKLOW(id,p,n))
    +1)
10
11
12 int main (int argc, char **argv)
13 {
14     NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
15
16     int count;                // Local prime count

```

```
17     double elapsed_time;           // Parallel execution time
18     int first;                     //index of first multiple
19     int global_count;              // Global prime count
20     int high_value;                // Highest value on this proc.
21     int i;
22     int rank;                       // Process id number
23     int index=0;                    // Index of current prime
24     int low_value;                 // Lowest value on this proc.
25     char *marked;                  // Portion of 2,...n
26     int n = 60;                    // Sieving from 2 to n=60
27     int p;                          // Number of processes
28     int proc0_size;                 // Size of proc0's subarray
29     int prime;                      // Current prime
30     int size;                       // Elements in 'marked'
31
32
33
34     MPIInstance *mpi = [MPIInstance mpiWith:&argc :&argv];
35
36     MPIComm *mCommWorld = [mpi commWorld];
37
38     // Start the timer
39     [mCommWorld MPIBarrier];
40     elapsed_time = -[[mpi MPIWTick] doubleValue];
41
42
43     p = [[mCommWorld MPICommSize] intValue];
44     rank = [[mCommWorld MPICommRank] intValue];
45
```

```
46     // Figure out this process's share of the array, as well as the
        integers
47     // represented by the first and last array elements.
48     low_value = 2 + BLOCKLOW(rank,p,n-1);
49     high_value = 2 + BLOCKHIGH(rank,p,n-1);
50
51     size =BLOCK_SIZE(rank,p, n-1 );
52
53     // Bail out if all the primes used for the sieving are not all
        held by process 0
54
55     proc0_size = (n-1)/p;
56     if ((2 + proc0_size) < (int) sqrt((double) n))
57     {
58         if (!rank)
59             NSLog(@"Too many processes");
60         [mpi release];
61         exit(1);
62     }
63
64     // Allocate this process share of the array
65     marked = (char *) malloc(size);
66
67     if (marked == NULL)
68     {
69         NSLog(@"Cannot allocate enough memory");
70         [mpi release];
71         exit(1);
72     }
73
```

```
74     for (i = 0; i < size; i++)
75         marked[i] = 0;
76     if (!rank)
77         index = 0;
78
79     prime = 2;
80
81     do {
82         if (prime * prime > low_value) {
83             first = prime * prime - low_value;
84         } else {
85             if (!(low_value % prime)) {
86                 first = 0;
87             } else {
88                 first = prime - (low_value % prime);
89             }
90         }
91
92         for (i = first; i < size; i+=prime)
93             marked[i]=1;
94
95         if (!rank){
96             while(marked[++index]);
97             prime = index + 2;
98         }
99
100         [mCommWorld MPIBcast:&prime ofSize:1 ofType:MPLINT rank:0];
101     } while (prime * prime <= n);
102
103
```



```
104
105     count = 0;
106     for (i = 0; i < size; i++)
107         if (!marked[i]) count++;
108
109     [mCommWorld MPIReduce:&count outMessage:&global_count ofSize:1
110        ofType:MPI_INT withOp:MPLSUM rank:0];
111
112     // Stop the timer
113     elapsed_time += [[mpi MPIWTime] doubleValue];
114
115     // Prints the results
116
117     if (!rank)
118     {
119         NSLog(@"%d primes are less than or equal to %d", global_count
120            , n);
121         NSLog(@"Total elapsed time : %f", elapsed_time);
122     }
123
124     free(marked);
125     [mpi release];
126
127     [pool release];
128     return 0;
129 }
```

# Conclusion

In this paper we discussed three possible ways to implement a distributed application on Mac OS X. We illustrated how powerful the Objective-C's distributed objects architecture was and saw through examples how easy it was to implement. If one ever needs to extend the distributed capabilities of an existing Objective-C application, this is probably the way to go. It provides everything one needs to build strong distributed applications and it is probably the most intuitive way for an Objective-C programmer to start computing on a grid.

With the XMLRPCObjC framework, one can extend the application with distributed operations based on web standards and provide access to any client. Even a Perl client could query a server one coded with that framework. This is probably the most interoperable distributing system available to Mac OS X.

With MPIObjC, I intend to provide a familiar way for Objective-C programmers to access a wide and popular message passing library. Although it is not completed yet, since one cannot do operations on virtual topology or groups with pure Objective-C calls, it is handy for using distributed resources on a local network through a standard and continuously evolving library approved by a wide community of researchers. With MPI-2 standards finalized and MPI implementations getting more and more up to date to its new features, we are witnessing what could be the definitive standard in distributed application programming for the next decade.

# Bibliography

- [PADL03] XMLRPCObjC Framework, *PADL Software*  
<http://www.padl.com/Research/XMLRPCObjC.html>, 2003
- [QUIN03] Michael J. Quinn, *Parallel Programming in C with MPI and OpenMP*, McGraw-Hill Editions, 2003
- [PACS01] Introduction to MPI, *PACS Training group*, 2001
- [GAMA02] Simon Garfinkel & Michael K. Mahoney, *Building Cocoa Applications, A Step-By-Step Guide*, O'Reilly Editions, 2002
- [GGKK03] Ananth Grama, Anshul Gupta, George Karypis, Vipi Kumar, *Introduction to Parallel Computing, 2<sup>nd</sup> edition*, Pearson Editions, 2003
- [WILU96] George V. Wilson, Paul Lu, *Parallel Programming using C++*, The MIT Press, 1996
- [DFFG03] Jack Dongara, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon, Andy White, *Sourcebook of parallel computing*, Morgan Kauffman Publishers, 2003
- [WIAL99] Barry Wilkinson, Michael Allen, *Parallel Programming*, Prentice Hall, 1999
- [APPL04] Introduction to Distributed Object, *Apple Developer Connection*  
<http://developer.apple.com/documentation/Cocoa/Conceptual/DistrObjects/>, 2004
- [STAN03] Distributed Objects in Objective-C, *Russel Standish*,  
<http://parallel.hpc.unsw.edu.au/rks/docs/ecolab4/node6.html>, 2003

[MMPI04] MacMPI, *Viktor Decyk, Dean Dauger, Pieter Kokelaar*

<http://exodus.physics.ucla.edu/appleseed/dev/Developer.html>, 2004

# Source Code Repository

## Freshmint source code

```
1 //
2 //  DownloadFile.h
3 //  Freshmint
4 //
5 //  Created by Jean-Matthieu
6 //
7
8 #import <Foundation/Foundation.h>
9
10
11 @interface DownloadFile : NSObject {
12     NSURLDownload *m_download;
13     NSURLRequest *m_request;
14     NSString *m_filepath;
15     id m_delegate;
16 }
17
18 - (DownloadFile *)initWithURL:(NSURL *)url;
19 - (NSData *)contentsData;
20 - (NSString *)contentsPath;
21 - (id)delegate;
22 - (void)setDelegate:(id)delegate;
23
24 @end
25
26 @interface NSObject (DownloadFileDelegate)
27
28 - (void)downloadFileDidFinish:(DownloadFile *)
29     download;
30
31 - (void)downloadFile:(DownloadFile *)download
32     didFinishWithError:(NSError *)error;
33
34 @end
35 //
36 //  DownloadFile.m
37 //  Freshmint
38 //
39
40 #import "DownloadFile.h"
41 #import <WebKit/WebKit.h>
42 #import <Foundation/NSURLResponse.h>
43 #import <Foundation/NSError.h>
44
45 @implementation DownloadFile
46
47 - (DownloadFile *)initWithURL:(NSURL *)url
48 {
49     self = [super init];
50     if (self) {
51         m_request = [NSURLRequest requestWithURL:
52             url cachePolicy:
53                 NSURLRequestUseProtocolCachePolicy
54                 timeoutInterval: 30 ];
55         m_download = [ [NSURLDownload alloc]
56             initWithRequest: m_request delegate:
57                 self ];
58     }
59     return self;
60 }
61
62 - (void)dealloc
63 {
64     NSFileManager *fm = [NSFileManager
65         defaultManager];
66     [fm changeCurrentDirectoryPath: [m_filepath
67         stringByDeletingLastPathComponent] ];
68     [fm removeFileAtPath: m_filepath handler: self
69         ];
70     [m_download release];
71     [m_filepath release];
72     [super dealloc];
73 }
74
75 - (NSData *)contentsData
76 {
77     NSData *reply = [NSData
```

```

72     return reply;
73
74     dataWithContentsOfFile: m_filepath ];
75
76 - (NSString *)contentsPath
77 {
78     return m_filepath;
79 }
80
81 - (id)delegate
82 {
83     return m_delegate;
84 }
85
86 - (void)setDelegate:(id)delegate
87 {
88     m_delegate = delegate;
89 }
90
91 #pragma mark -
92 #pragma mark == NSDownload Delegate ==
93 - (void)download:(NSURLDownload*)download
94     decideDestinationWithSuggestedFilename:(
95         NSString*)filename
96     {
97         NSString *path = [ @"/tmp"
98             stringByAppendingPathComponent: filename
99             ];
100         [ download setDestination: path allowOverwrite
101             : YES ];
102     }
103
104 - (void)download:(NSURLDownload *)download
105     didCreateDestination:(NSString *)path
106     {
107         m_filepath = [ path copy ];
108     }
109
110 - (void)download:(NSURLDownload *)download
111     didFailWithError:(NSError *)error
112     {
113         [ m_delegate downloadFileDidFinish: self ];
114     }
115
116 - (void)download:(NSURLDownload *)download
117     didFailWithError:(NSError *)error
118     {
119         [ m_delegate downloadFile: self
120             didFailWithError: error ];
121     }
122
123 @end//
124 // Freshmeat.h
125 // Freshmint
126
127 //
128 // Created by Jean-Matthieu
129 //
130
131 #import <Foundation/Foundation.h>
132 #include <XMLRPCObjC/XMLRPCObjC.h>
133
134 @interface Freshmeat : NSObject
135 {
136     XMLRPCClient *client;
137     NSMutableDictionary *sessionDictionary;
138
139     BOOL isConnected;
140 }
141
142 /* [ sessionDictionary method ]
143 * Parameters:
144 * None
145 *
146 * Returns:
147 * NSDictionary with session informations
148 *
149 * Description:
150 * Returns a dictionary containing SID, API
151     Version, Lifetime, logintime
152 */
153 - (NSDictionary *)sessionDictionary;
154
155 /* [ autoLogout method ]
156 * Parameters:
157 * None
158 *
159 * Returns:
160 * None
161 *
162 * Description:
163 * Automatically logs out from Freshmeat.net
164 */
165 - (void)autoLogout;
166
167 /* [ isConnected method ]
168 * Parameters:
169 * None
170 *
171 * Returns:
172 * None
173 *
174 * Description:
175 * Informs whether a session is active or not.
176 */
177 - (BOOL)isConnected;
178
179 /* [ fetch_available_licenses method ]

```

```

175     * Parameters:                                NSString *)project_name branch:(NSString *)
176     * None                                         branch_name version:(NSString *)version;
177     *                                             225
178     * Returns:                                     226
179     * Array of available licenses                 227 /* [ login method ]
180 */                                             228 * Parameters (passed in struct form):
181 - (NSArray *)fetch_available_licenses;          229 * username                - Regular freshmeat
182                                             username
183 /* [ fetch_available_release_foci method ]      230 * password                - Regular freshmeat
184     * Parameters:                                password
185     * None                                       231 *
186     *                                             232 * Returns:
187     * Returns:                                   233 * Struct of SID, lifetime, and API Version
188     * Struct of available release focus types and 234 * SID: Session ID to be used in subsequent
        associated ID                               requests to the XML-RPC service
189 */                                             235 * Lifetime: Lifetime of the session ID in
190 - (NSDictionary *)fetch_available_release_foci; seconds
191                                             236 * API Version: API Version currently in use
192 /* [ fetch_project_list method ]               237 */
193 * Parameters (passed in struct form):          238 - (void)login:(NSString *)username password:(
194 * SID                - Session ID to work      NSString *)password;
195     with                                             239
196     with                                             240
197 * Returns:                                       241 /* [ logout method ]
198 * Struct consisting of "projectname_full", "    242 * Parameters (passed in struct form):
199     projectname_short", "project-status", and " 243 * SID                - Session ID to
200     project_version"                             terminate
201                                             244 *
202                                             245 * Returns:
203 /* [ fetch_branch_list method ]               246 * Struct of "OK" => "Logout successful." if
204     * Parameters (passed in struct form):        logout was successful
205     * SID                - Session ID to work    247 */
206     with                                             248 - (void)logout:(NSString *)SID;
207     * project_name            - Project name to   249
208     * Returns:                                       250
209     * Array of branch name strings.              251 /* [ publish_release method ]
210 */                                             252 * Parameters (passed in struct form):
211 - (NSArray *)fetch_branch_list_for_project:(    253 * SID                - Session ID to work
        NSString *)project_name;                with
212                                             254 * project_name            - Project name to
213                                             submit a release for
214 /* [ fetch_release method ]                   255 * branch_name            - Branch name to
215     * Parameters (passed in struct form):        submit a release for
216     * SID                - Session ID            256 * version                - Version string of
217     * project_name            - Project name      new release
218     * branch_name            - Branch name        257 * changes                - Changes list, no
219     * version                - Release version    HTML, character limit 600 chars
220     string                                             258 * release_focus          - Release focus ID of
221     * Returns:                                       new release (see Appendix A)
222     * Struct consisting of "version", "changes", " 259 * hide_from_frontpage    - Set to 'Y' if
        release_focus", and "hide_from_frontpage" release is to be hidden from
223 */                                             260 * frontpage, everything else does not hide it
224 - (NSDictionary *)fetch_release_for_project:(  261 * license                - Branch license
        NSString *)project_name;                262 * url_homepage           - Homepage
263 * url_tgz                - Tar/GZ
264 * url_bz2                - Tar/BZ2
265 * url_zip                - Zip
266 * url_changelog          - Changelog
267 * url_rpm                - RPM package

```

```

268 * url_deb          - Debian package      318 {
269 * url_osx          - OS X package         319     self = [super init];
270 * url_bsdport      - BSD Ports URL        320     if (self) {
271 * url_purchase     - Purchase             321         client = [[XMLRPCClient client:[NSURL
272 * url_cvs           - CVS tree (cvsweb)    322             URLWithString:@"http://freshmeat.net/
273 * url_list         - Mailing list archive  323             xmlrpc"]] retain];
274 * url_mirror       - Mirror site          324     sessionDictionary = [[NSMutableDictionary
275 * url_demo         - Demo site            325         alloc] init];
276 *                 326     isConnected = NO;
277 * Returns:         327     }
278 * Struct of "OK" => "submission successful" 328     return self;
279 *                 329 }
280 * Notes:           330
281 * The "license" and "url_*" fields are optional 331 -(void) dealloc
282 * and will be taken from the branch record iB29 {
283 * they                 332     [sessionDictionary release];
284 * are omitted from the submission. The '       333     [client release];
285 *   hide_from_frontpage' option can be omitted 334     [super dealloc];
286 *   an defaults to                 335 }
287 * 'do not hide'.                 336
288 *                               337 -(NSMutableDictionary *)sessionDictionary
289 * For convinience, we pass a dictionary to this 338 {
290 *   method                 339     return sessionDictionary;
291 */                               340 }
292 - (void)publish_release:(NSDictionary *)        341     newReleaseInfo;
293 *                               342     -(void)autoLogout
294 *                               343 {
295 *                               344     [self logout:[sessionDictionary objectForKey:@"
296 *                               345     SID"]];
297 * Parameters (passed in struct form):          346 }
298 * SID                - Session ID              347
299 * project_name       - Project name             348 - (BOOL)isConnected
300 * branch_name       - Branch name              349 {
301 * version            - Release version          350     return isConnected;
302 * string                 351 }
303 *                               352
304 * Returns:           353 // Freshmeat methods invocation
305 * Struct of "OK" => "Withdraw successful.".    354 - (NSArray *)fetch_available_licenses
306 */                               355 {
307 - (void)withdraw_release_for_project:(NSString *) 356     NSArray *object;
308 *   project_name branch:(NSString *)branch_name  357     object = [client invoke:@"
309 *   version:(NSString *)version;                 358     fetch_available_licenses" withArguments:[
310 *                               359     NSArray arrayWithObject:@""];
311 *                               360     return object;
312 @end                               361 }
313 *                               362 - (NSMutableDictionary *)fetch_available_release_foci
314 *                               363 {
315 *                               364     NSDictionary *object;
316 *                               365     object = [client invoke:@"
317 *                               366     fetch_available_release_foci"
318 *                               367     withArguments:[NSArray arrayWithObject:@"
319 *                               370     "]];
320 *                               371     NSLog([object description]);
321 *                               372     return object;
322 *                               373 }
323 * @implementation Freshmeat           374
324 *                               375
325 *                               376
326 *                               377
327 *                               378
328 *                               379
329 *                               380
330 *                               381
331 *                               382
332 *                               383
333 *                               384
334 *                               385
335 *                               386
336 *                               387
337 *                               388
338 *                               389
339 *                               390
340 *                               391
341 *                               392
342 *                               393
343 *                               394
344 *                               395
345 *                               396
346 *                               397
347 *                               398
348 *                               399
349 *                               400
350 *                               401
351 *                               402
352 *                               403
353 *                               404
354 *                               405
355 *                               406
356 *                               407
357 *                               408
358 *                               409
359 *                               410
360 *                               411
361 *                               412
362 *                               413
363 *                               414
364 *                               415
365 *                               416
366 *                               417
367 *                               418
368 *                               419
369 *                               420
370 *                               421
371 *                               422
372 *                               423
373 *                               424
374 *                               425
375 *                               426
376 *                               427
377 *                               428
378 *                               429
379 *                               430
380 *                               431
381 *                               432
382 *                               433
383 *                               434
384 *                               435
385 *                               436
386 *                               437
387 *                               438
388 *                               439
389 *                               440
390 *                               441
391 *                               442
392 *                               443
393 *                               444
394 *                               445
395 *                               446
396 *                               447
397 *                               448
398 *                               449
399 *                               450
400 *                               451
401 *                               452
402 *                               453
403 *                               454
404 *                               455
405 *                               456
406 *                               457
407 *                               458
408 *                               459
409 *                               460
410 *                               461
411 *                               462
412 *                               463
413 *                               464
414 *                               465
415 *                               466
416 *                               467
417 *                               468
418 *                               469
419 *                               470
420 *                               471
421 *                               472
422 *                               473
423 *                               474
424 *                               475
425 *                               476
426 *                               477
427 *                               478
428 *                               479
429 *                               480
430 *                               481
431 *                               482
432 *                               483
433 *                               484
434 *                               485
435 *                               486
436 *                               487
437 *                               488
438 *                               489
439 *                               490
440 *                               491
441 *                               492
442 *                               493
443 *                               494
444 *                               495
445 *                               496
446 *                               497
447 *                               498
448 *                               499
449 *                               500
450 *                               501
451 *                               502
452 *                               503
453 *                               504
454 *                               505
455 *                               506
456 *                               507
457 *                               508
458 *                               509
459 *                               510
460 *                               511
461 *                               512
462 *                               513
463 *                               514
464 *                               515
465 *                               516
466 *                               517
467 *                               518
468 *                               519
469 *                               520
470 *                               521
471 *                               522
472 *                               523
473 *                               524
474 *                               525
475 *                               526
476 *                               527
477 *                               528
478 *                               529
479 *                               530
480 *                               531
481 *                               532
482 *                               533
483 *                               534
484 *                               535
485 *                               536
486 *                               537
487 *                               538
488 *                               539
489 *                               540
490 *                               541
491 *                               542
492 *                               543
493 *                               544
494 *                               545
495 *                               546
496 *                               547
497 *                               548
498 *                               549
499 *                               550
500 *                               551
501 *                               552
502 *                               553
503 *                               554
504 *                               555
505 *                               556
506 *                               557
507 *                               558
508 *                               559
509 *                               560
510 *                               561
511 *                               562
512 *                               563
513 *                               564
514 *                               565
515 *                               566
516 *                               567
517 *                               568
518 *                               569
519 *                               570
520 *                               571
521 *                               572
522 *                               573
523 *                               574
524 *                               575
525 *                               576
526 *                               577
527 *                               578
528 *                               579
529 *                               580
530 *                               581
531 *                               582
532 *                               583
533 *                               584
534 *                               585
535 *                               586
536 *                               587
537 *                               588
538 *                               589
539 *                               590
540 *                               591
541 *                               592
542 *                               593
543 *                               594
544 *                               595
545 *                               596
546 *                               597
547 *                               598
548 *                               599
549 *                               600
550 *                               601
551 *                               602
552 *                               603
553 *                               604
554 *                               605
555 *                               606
556 *                               607
557 *                               608
558 *                               609
559 *                               610
560 *                               611
561 *                               612
562 *                               613
563 *                               614
564 *                               615
565 *                               616
566 *                               617
567 *                               618
568 *                               619
569 *                               620
570 *                               621
571 *                               622
572 *                               623
573 *                               624
574 *                               625
575 *                               626
576 *                               627
577 *                               628
578 *                               629
579 *                               630
580 *                               631
581 *                               632
582 *                               633
583 *                               634
584 *                               635
585 *                               636
586 *                               637
587 *                               638
588 *                               639
589 *                               640
590 *                               641
591 *                               642
592 *                               643
593 *                               644
594 *                               645
595 *                               646
596 *                               647
597 *                               648
598 *                               649
599 *                               650
600 *                               651
601 *                               652
602 *                               653
603 *                               654
604 *                               655
605 *                               656
606 *                               657
607 *                               658
608 *                               659
609 *                               660
610 *                               661
611 *                               662
612 *                               663
613 *                               664
614 *                               665
615 *                               666
616 *                               667
617 *                               668
618 *                               669
619 *                               670
620 *                               671
621 *                               672
622 *                               673
623 *                               674
624 *                               675
625 *                               676
626 *                               677
627 *                               678
628 *                               679
629 *                               680
630 *                               681
631 *                               682
632 *                               683
633 *                               684
634 *                               685
635 *                               686
636 *                               687
637 *                               688
638 *                               689
639 *                               690
640 *                               691
641 *                               692
642 *                               693
643 *                               694
644 *                               695
645 *                               696
646 *                               697
647 *                               698
648 *                               699
649 *                               700
650 *                               701
651 *                               702
652 *                               703
653 *                               704
654 *                               705
655 *                               706
656 *                               707
657 *                               708
658 *                               709
659 *                               710
660 *                               711
661 *                               712
662 *                               713
663 *                               714
664 *                               715
665 *                               716
666 *                               717
667 *                               718
668 *                               719
669 *                               720
670 *                               721
671 *                               722
672 *                               723
673 *                               724
674 *                               725
675 *                               726
676 *                               727
677 *                               728
678 *                               729
679 *                               730
680 *                               731
681 *                               732
682 *                               733
683 *                               734
684 *                               735
685 *                               736
686 *                               737
687 *                               738
688 *                               739
689 *                               740
690 *                               741
691 *                               742
692 *                               743
693 *                               744
694 *                               745
695 *                               746
696 *                               747
697 *                               748
698 *                               749
699 *                               750
700 *                               751
701 *                               752
702 *                               753
703 *                               754
704 *                               755
705 *                               756
706 *                               757
707 *                               758
708 *                               759
709 *                               760
710 *                               761
711 *                               762
712 *                               763
713 *                               764
714 *                               765
715 *                               766
716 *                               767
717 *                               768
718 *                               769
719 *                               770
720 *                               771
721 *                               772
722 *                               773
723 *                               774
724 *                               775
725 *                               776
726 *                               777
727 *                               778
728 *                               779
729 *                               780
730 *                               781
731 *                               782
732 *                               783
733 *                               784
734 *                               785
735 *                               786
736 *                               787
737 *                               788
738 *                               789
739 *                               790
740 *                               791
741 *                               792
742 *                               793
743 *                               794
744 *                               795
745 *                               796
746 *                               797
747 *                               798
748 *                               799
749 *                               800
750 *                               801
751 *                               802
752 *                               803
753 *                               804
754 *                               805
755 *                               806
756 *                               807
757 *                               808
758 *                               809
759 *                               810
760 *                               811
761 *                               812
762 *                               813
763 *                               814
764 *                               815
765 *                               816
766 *                               817
767 *                               818
768 *                               819
769 *                               820
770 *                               821
771 *                               822
772 *                               823
773 *                               824
774 *                               825
775 *                               826
776 *                               827
777 *                               828
778 *                               829
779 *                               830
780 *                               831
781 *                               832
782 *                               833
783 *                               834
784 *                               835
785 *                               836
786 *                               837
787 *                               838
788 *                               839
789 *                               840
790 *                               841
791 *                               842
792 *                               843
793 *                               844
794 *                               845
795 *                               846
796 *                               847
797 *                               848
798 *                               849
799 *                               850
800 *                               851
801 *                               852
802 *                               853
803 *                               854
804 *                               855
805 *                               856
806 *                               857
807 *                               858
808 *                               859
809 *                               860
810 *                               861
811 *                               862
812 *                               863
813 *                               864
814 *                               865
815 *                               866
816 *                               867
817 *                               868
818 *                               869
819 *                               870
820 *                               871
821 *                               872
822 *                               873
823 *                               874
824 *                               875
825 *                               876
826 *                               877
827 *                               878
828 *                               879
829 *                               880
830 *                               881
831 *                               882
832 *                               883
833 *                               884
834 *                               885
835 *                               886
836 *                               887
837 *                               888
838 *                               889
839 *                               890
840 *                               891
841 *                               892
842 *                               893
843 *                               894
844 *                               895
845 *                               896
846 *                               897
847 *                               898
848 *                               899
849 *                               900
850 *                               901
851 *                               902
852 *                               903
853 *                               904
854 *                               905
855 *                               906
856 *                               907
857 *                               908
858 *                               909
859 *                               910
860 *                               911
861 *                               912
862 *                               913
863 *                               914
864 *                               915
865 *                               916
866 *                               917
867 *                               918
868 *                               919
869 *                               920
870 *                               921
871 *                               922
872 *                               923
873 *                               924
874 *                               925
875 *                               926
876 *                               927
877 *                               928
878 *                               929
879 *                               930
880 *                               931
881 *                               932
882 *                               933
883 *                               934
884 *                               935
885 *                               936
886 *                               937
887 *                               938
888 *                               939
889 *                               940
890 *                               941
891 *                               942
892 *                               943
893 *                               944
894 *                               945
895 *                               946
896 *                               947
897 *                               948
898 *                               949
899 *                               950
900 *                               951
901 *                               952
902 *                               953
903 *                               954
904 *                               955
905 *                               956
906 *                               957
907 *                               958
908 *                               959
909 *                               960
910 *                               961
911 *                               962
912 *                               963
913 *                               964
914 *                               965
915 *                               966
916 *                               967
917 *                               968
918 *                               969
919 *                               970
920 *                               971
921 *                               972
922 *                               973
923 *                               974
924 *                               975
925 *                               976
926 *                               977
927 *                               978
928 *                               979
929 *                               980
930 *                               981
931 *                               982
932 *                               983
933 *                               984
934 *                               985
935 *                               986
936 *                               987
937 *                               988
938 *                               989
939 *                               990
940 *                               991
941 *                               992
942 *                               993
943 *                               994
944 *                               995
945 *                               996
946 *                               997
947 *                               998
948 *                               999
949 *                               1000
950 *                               1001
951 *                               1002
952 *                               1003
953 *                               1004
954 *                               1005
955 *                               1006
956 *                               1007
957 *                               1008
958 *                               1009
959 *                               1010
960 *                               1011
961 *                               1012
962 *                               1013
963 *                               1014
964 *                               1015
965 *                               1016
966 *                               1017
967 *                               1018
968 *                               1019
969 *                               1020
970 *                               1021
971 *                               1022
972 *                               1023
973 *                               1024
974 *                               1025
975 *                               1026
976 *                               1027
977 *                               1028
978 *                               1029
979 *                               1030
980 *                               1031
981 *                               1032
982 *                               1033
983 *                               1034
984 *                               1035
985 *                               1036
986 *                               1037
987 *                               1038
988 *                               1039
989 *                               1040
990 *                               1041
991 *                               1042
992 *                               1043
993 *                               1044
994 *                               1045
995 *                               1046
996 *                               1047
997 *                               1048
998 *                               1049
999 *                               1050
1000 *                               1051
1001 *                               1052
1002 *                               1053
1003 *                               1054
1004 *                               1055
1005 *                               1056
1006 *                               1057
1007 *                               1058
1008 *                               1059
1009 *                               1060
1010 *                               1061
1011 *                               1062
1012 *                               1063
1013 *                               1064
1014 *                               1065
1015 *                               1066
1016 *                               1067
1017 *                               1068
1018 *                               1069
1019 *                               1070
1020 *                               1071
1021 *                               1072
1022 *                               1073
1023 *                               1074
1024 *                               1075
1025 *                               1076
1026 *                               1077
1027 *                               1078
1028 *                               1079
1029 *                               1080
1030 *                               1081
1031 *                               1082
1032 *                               1083
1033 *                               1084
1034 *                               1085
1035 *                               1086
1036 *                               1087
1037 *                               1088
1038 *                               1089
1039 *                               1090
1040 *                               1091
1041 *                               1092
1042 *                               1093
1043 *                               1094
1044 *                               1095
1045 *                               1096
1046 *                               1097
1047 *                               1098
1048 *                               1099
1049 *                               1100
1050 *                               1101
1051 *                               1102
1052 *                               1103
1053 *                               1104
1054 *                               1105
1055 *                               1106
1056 *                               1107
1057 *                               1108
1058 *                               1109
1059 *                               1110
1060 *                               1111
1061 *                               1112
1062 *                               1113
1063 *                               1114
1064 *                               1115
1065 *                               1116
1066 *                               1117
1067 *                               1118
1068 *                               1119
1069 *                               1120
1070 *                               1121
1071 *                               1122
1072 *                               1123
1073 *                               1124
1074 *                               1125
1075 *                               1126
1076 *                               1127
1077 *                               1128
1078 *                               1129
1079 *                               1130
1080 *                               1131
1081 *                               1132
1082 *                               1133
1083 *                               1134
1084 *                               1135
1085 *                               1136
1086 *                               1137
1087 *                               1138
1088 *                               1139
1089 *                               1140
1090 *                               1141
1091 *                               1142
1092 *                               1143
1093 *                               1144
1094 *                               1145
1095 *                               1146
1096 *                               1147
1097 *                               1148
1098 *                               1149
1099 *                               1150
1100 *                               1151
1101 *                               1152
1102 *                               1153
1103 *                               1154
1104 *                               1155
1105 *                               1156
1106 *                               1157
1107 *                               1158
1108 *                               1159
1109 *                               1160
1110 *                               1161
1111 *                               1162
1112 *                               1163
1113 *                               1164
1114 *                               1165
1115 *                               1166
1116 *                               1167
1117 *                               1168
1118 *                               1169
1119 *                               1170
1120 *                               1171
1121 *                               1172
1122 *                               1173
1123 *                               1174
1124 *                               1175
1125 *                               1176
1126 *                               1177
1127 *                               1178
1128 *                               1179
1129 *                               1180
1130 *                               1181
1131 *                               1182
1132 *                               1183
1133 *                               1184
1134 *                               1185
1135 *                               1186
1136 *                               1187
1137 *                               1188
1138 *                               1189
1139 *                               1190
1140 *                               1191
1141 *                               1192
1142 *                               1193
1143 *                               1194
1144 *                               1195
1145 *                               1196
1146 *                               1197
1147 *                               1198
1148 *                               1199
1149 *                               1200
1150 *                               1201
1151 *                               1202
1152 *                               1203
1153 *                               1204
1154 *                               1205
1155 *                               1206
1156 *                               1207
1157 *                               1208
1158 *                               1209
1159 *                               1210
1160 *                               1211
1161 *                               1212
1162 *                               1213
1163 *                               1214
1164 *                               1215
1165 *                               1216
1166 *                               1217
1167 *                               1218
1168 *                               1219
1169 *                               1220
1170 *                               1221
1171 *                               1222
1172 *                               1223
1173 *                               1224
1174 *                               1225
1175 *                               1226
1176 *                               1227
1177 *                               1228
1178 *                               1229
1179 *                               1230
1180 *                               1231
1181 *                               1232
1182 *                               1233
1183 *                               1234
1184 *                               1235
1185 *                               1236
1186 *                               1237
1187 *                               1238
1188 *                               1239
1189 *                               1240
1190 *                               1241
1191 *                               1242
1192 *                               1243
1193 *                               1244
1194 *                               1245
1195 *                               1246
1196 *                               1247
1197 *                               1248
1198 *                               1249
1199 *                               1250
1200 *                               1251
1201 *                               1252
1202 *                               1253
1203 *                               1254
1204 *                               1255
1205 *                               1256
1206 *                               1257
1207 *                               1258
1208 *                               1259
1209 *                               1260
1210 *                               1261
1211 *                               1262
1212 *                               1263
1213 *                               1264
1214 *                               1265
1215 *                               1266
1216 *                               1267
1217 *                               1268
1218 *                               1269
1219 *                               1270
1220 *                               1271
1221 *                               1272
1222 *                               1273
1223 *                               1274
1224 *                               1275
1225 *                               1276
1226 *                               1277
1227 *                               1278
1228 *                               1279
1229 *                               1280
1230 *                               1281
1231 *                               1282
1232 *                               1283
1233 *                               1284
1234 *                               1285
1235 *                               1286
1236 *                               1287
1237 *                               1288
1238 *                               1289
1239 *                               1290
1240 *                               1291
1241 *                               1292
1242 *                               1293
1243 *                               1294
1244 *                               1295
1245 *                               1296
1246 *                               1297
1247 *                               1298
1248 *                               1299
1249 *                               1300
1250 *                               1301
1251 *                               1302
1252 *                               1303
1253 *                               1304
1254 *                               1305
1255 *                               1306
1256 *                               1307
1257 *                               1308
1258 *                               1309
1259 *                               1310
1260 *                               1311
1261 *                               1312
1262 *                               1313
1263 *                               1314
1264 *                               1315
1265 *                               1316
1266 *                               1317
1267 *                               1318
1268 *                               1319
1269 *                               1320
1270 *                               1321
1271 *                               1322
1272 *                               1323
1273 *                               1324
1274 *                               1325
1275 *                               1326
1276 *                               1327
1277 *                               1328
1278 *                               1329
1279 *                               1330
1280 *                               1331
1281 *                               1332
1282 *                               1333
1283 *                               1334
1284 *                               1335
1285 *                               1336
1286 *                               1337
1287 *                               1338
1288 *                               1339
1289 *                               1340
1290 *                               1341
1291 *                               1342
1292 *                               1343
1293 *                               1344
1294 *                               1345
1295 *                               1346
1296 *                               1347
1297 *                               1348
1298 *                               1349
1299 *                               1350
1300 *                               1351
1301 *                               1352
1302 *                               1353
1303 *                               1354
1304 *                               1355
1305 *                               1356
1306 *                               1357
1307 *                               1358
1308 *                               1359
1309 *                               1360
1310 *                               1361
1311 *                               1362
1312 *                               1363
1313 *                               1364
1314 *                               1365
1315 *                               1366
1316 *                               1367
1317 *                               1368
1318 *                               1369
1319 *                               1370
1320 *                               1371
1321 *                               1372
1322 *                               1373
1323 *                               1374
1324 *                               1375
1325 *                               1376
1326 *                               1377
1327 *                               1378
1328 *                               1379
1329 *                               1380
1330 *                               1381
1331 *                               1382
1332 *                               1383
1333 *                               1384
1334 *                               1385
1335 *                               1386
1336 *                               1387
1337 *                               1388
1338 *                               1389
1339 *                               1390
1340 *                               1391
1341 *                               1392
1342 *                               1393
1343 *                               1394
1344 *                               1395
1345 *                               1396
1346 *                               1397
1347 *                               1398
1348 *                               1399
1349 *                               1400
1350 *                               1401
1351 *                               1402
1352 *                               1403
1353 *                               1404
1354 *                               1405
1355 *                               1406
1356 *                               1407
1357 *                               1408
1358 *                               1409
1359 *                               1410
1360 *                               1411
1361 *                               1412
1362 *                               1413
1363 *                               1414
1364 *                               1415
1365 *                               1416
1366 *                               1417
1367 *                               1418
1368 *                               1419
1369 *                               1420
1370 *                               1421
1371 *                               1422
1372
```



Oxford Brookes University

```

421     NSArray *args = [NSArray arrayWithObject:
422         myStruct];
423     object = [client invoke:@"fetch_release"
424         withArguments:args];
425     //NSLog(@"%@", [object description]);
426     return object;
427 }
428 - (void)login:(NSString *)username password:(
429     NSString *)password
430 {
431     id object = nil;
432     [sessionDictionary removeAllObjects];
433     NSDictionary *myStruct = [NSDictionary
434         dictionaryWithObjects:[NSArray
435             arrayWithObjects:username, password, nil]
436         forKeys:[NSArray arrayWithObjects:@"
437             username", @"password", nil]];
438     NSArray *args = [NSArray arrayWithObject:
439         myStruct];
440     object = [client invoke:@"login" withArguments:
441         args];
442     [sessionDictionary addEntriesFromDictionary:
443         object];
444     NSDate *date = [NSDate date];
445     [sessionDictionary setObject:date forKey:@"date
446         "];
447     isConnected = YES;
448     // Autologout 5 sec before session ends
449     [NSTimer scheduledTimerWithTimeInterval:[
450         [sessionDictionary objectForKey:@"Lifetime
451         "] intValue] - 5
452         target:
453             self
454         selector:
455             @selector(
456                 (
457                     autoLogout
458                 )
459                 userInfo:nil
460                 repeats:NO
461             )];
462     [self fetch_project_list];
463 }
464 - (void)logout:(NSString *)SID
465 {
466     NSDictionary *myStruct = [NSDictionary
467         dictionaryWithObjects:[NSArray
468             arrayWithObject:SID] forKeys:[NSArray
469             arrayWithObject:@"SID"]];
470     NSArray *args = [NSArray arrayWithObject:
471         myStruct];
472     [client invoke:@"logout" withArguments:args];
473     [sessionDictionary removeAllObjects];
474     isConnected = NO;
475     NSLog(@"Freshmeat session terminated");
476 }
477 - (void)publish_release:(NSDictionary *)
478     newReleaseInfo
479 {
480     NSLog([newReleaseInfo description]);
481     NSArray *args = [NSArray arrayWithObject:
482         newReleaseInfo];
483     [client invoke:@"publish_release" withArguments:
484         args];
485 }
486 - (void)withdraw_release_for_project:(NSString *)
487     project_name branch:(NSString *)branch_name
488     version:(NSString *)version
489 {
490     NSDictionary *myStruct = [NSDictionary
491         dictionaryWithObjects:[NSArray
492             arrayWithObjects:[sessionDictionary
493                 objectForKey:@"SID"], project_name,
494                 branch_name, version, nil]
495         forKeys:
496             @[
497                 "SID",
498                 "project_name",
499                 "branch_name",
500                 "version"
501             ]];
502     NSArray *args = [NSArray arrayWithObject:
503         myStruct];
504     [client invoke:@"withdraw_release"
505         withArguments:args];
506 }

```

```

480 @end
481 //
482 // NSStringEx.h
483 // Freshmint
484 //
485 // Created by Jean-Matthieu
486 //
487
488 #import <Foundation/Foundation.h>
489
490
491 @interface NSString (NSStringEx)
492
493 - (NSString *)trimWhitespace;
494 - (NSString *)trimHTML;
495
496 @end
497
498 //
499 // NSStringEx.m
500 // Freshmint
501 //
502 // Created by Jean-Matthieu
503 //
504
505 #import "NSStringEx.h"
506
507
508 @implementation NSString (NSStringEx)
509
510 - (NSString *)trimWhitespace
511 {
512     NSMutableString *str = [ [ self mutableCopy ]
513                             autorelease ];
514     CFStringTrimWhitespace( ( CFMutableStringRef ) str );
515     return (NSString *) ( [ [ str copy ] autorelease
516                             ] );
517 }
518
519 - (NSString *)trimHTML
520 {
521     int len = [ self length ];
522     NSMutableString *str = [ NSMutableString
523                             stringWithCapacity: len ];
524     int i = 0, level = 0;
525     for ( i = 0; i < len; i++ ) {
526         NSString *check = [ self
527                             substringWithRange: NSMakeRange( i
528                                                             , 1 ) ];
529         if ( [ check isEqualToString: @"<" ] ) {
530             level++;
531         } else if ( [ check isEqualToString: @">" ] ) {
532             level--;
533             if ( level == 0 ) {
534                 [ str appendString: @" " ];
535             }
536         } else if ( level == 0 ) {
537             [ str appendString: check ];
538         }
539     }
540
541 @end
542 //
543 // SessionController.h
544 // Freshmint
545 //
546 // Created by Jean-Matthieu
547 //
548
549 #import <Cocoa/Cocoa.h>
550 #import <WebKit/WebKit.h>
551
552 #import "Freshmeat.h"
553
554 @interface SessionController : NSObject
555 {
556     IBOutlet NSTextField *loginField;
557     IBOutlet NSWindow *loginWindow;
558     IBOutlet NSButton *closeLoginBtn;
559     IBOutlet NSButton *newReleaseBtn;
560     IBOutlet NSTableView *allProjectsTable;
561     IBOutlet NSTextField *nrBranchName;
562     IBOutlet NSFormCell *nrBSDField;
563     IBOutlet NSFormCell *nrBZ2Field;
564     IBOutlet NSFormCell *nrChangelogField;
565     IBOutlet NSTextView *nrChangesTV;
566     IBOutlet NSFormCell *nrCVSField;
567     IBOutlet NSFormCell *nrDEBField;
568     IBOutlet NSFormCell *nrDemoField;
569     IBOutlet NSPopUpButton *nrFocusPopUp;
570     IBOutlet NSButton *nrHideBtn;
571     IBOutlet NSFormCell *nrHomeField;
572     IBOutlet NSPopUpButton *nrLicensePopUp;
573     IBOutlet NSFormCell *nrMirrorField;
574     IBOutlet NSFormCell *nrMLField;
575     IBOutlet NSFormCell *nrOSXField;
576     IBOutlet NSTextField *nrProjectName;
577     IBOutlet NSFormCell *nrPurchaseField;
578     IBOutlet NSFormCell *nrRPMField;
579     IBOutlet NSFormCell *nrTGZField;
580     IBOutlet NSTextField *nrVersionField;
581     IBOutlet NSWindow *nrWindow;
582     IBOutlet NSFormCell *nrZipField;
583     IBOutlet NSSecureTextField *passwordField;
584     IBOutlet NSTableView *projectBranchTable;
585     IBOutlet NSWindow *projectWindow;
586     IBOutlet NSButton *withdrawBtn;
587     IBOutlet WebView *releaseWebView;
588     IBOutlet NSProgressIndicator *spinWheel;
589
590     Freshmeat *freshmeat;

```

```

591     NSMutableDictionary *myProjects;
592     //NSDictionary *releaseFocus;
593     NSDictionary *projectDictionary;
594     NSArray *licenseArray;
595     NSString *currentProject;
596     NSString *currentBranch;
597     BOOL closeAndQuit;
598 }
599
600 + (id) getInstance;
601 - (void) sortMyProjects;
602 - (void) closeNow: (NSWindow *) sheet;
603 - (void) showLoginSheet;
604
605 - (void) downloadXMLFileForProject: (NSString *)
        project;
606 - (void) displayXMLData: (NSData *) theData;
607 - (void) renderHTMLWithDictionary: (NSDictionary *)
        aDict;
608 - (void) setReleaseAndLicenseMenus;
609
610 - (IBAction) doLogin: (id) sender;
611 - (IBAction) doNewRelease: (id) sender;
612 - (IBAction) doPublish: (id) sender;
613 - (IBAction) doWithdraw: (id) sender;
614 - (IBAction) closeModal: (id) sender;
615
616 @end
617 //
618 // SessionController.m
619 // Freshmint
620 //
621 // Created by Jean-Matthieu
622 //
623
624 #import "SessionController.h"
625 #import "XMLFeeder.h"
626 #import "DownloadFile.h"
627
628 static SessionController *theSessionController =
        nil;
629 @implementation SessionController
630
631 +(id) getInstance
632 {
633     // TODO: Mutex Begin
634     if (theSessionController == nil) {
635         theSessionController = [[SessionController
        alloc] init];
636     }
637     // TODO: Mutex End
638     return theSessionController;
639 }
640
641
642 -(id) init
643 {
644     self = [super init];
645     if (self)
646     {
647         theSessionController = self;
648         freshmeat = [[Freshmeat alloc] init];
649         currentProject = nil;
650         currentBranch = nil;
651         closeAndQuit = YES;
652     }
653     return self;
654 }
655
656 -(void) dealloc
657 {
658     [freshmeat release];
659     //[releaseFocus release];
660     if (projectDictionary)
661         [projectDictionary release];
662     [licenseArray release];
663     [myProjects release];
664     [super dealloc];
665 }
666
667
668
669 - (void) applicationWillTerminate: (NSNotification *)
        aNotification
670 {
671     [[NSFileManager defaultManager]
        removeFileAtPath:@" /tmp/project.html"
        handler: nil];
672     NSString *sid = nil;
673     if ((sid = [[freshmeat sessionDictionary]
        objectForKey:@"SID"]) != nil)
674         [freshmeat logout:sid];
675 }
676
677 - (void) applicationDidFinishLaunching: (
        NSNotification *) aNotification
678 {
679     NSString *bgPath = [NSBundle pathForResource:@"
        bg" ofType:@"tif" inDirectory:[NSBundle
        mainBundle] bundlePath];
680     NSString *projectHTML = [NSString
        stringWithFormat:@"% <HTML><HEAD></HEAD><
        BODY MARGINHEIGHT=0 MARGINWIDTH=0
        BACKGROUND=%@ BGCOLOR=#FFFFFF></BODY></
        HTML>", bgPath];
681     [projectHTML writeToFile:@" /tmp/project.html"
        atomically:NO];
682     [[releaseWebView mainFrame] loadRequest:
        [NSURLRequest requestWithURL:[NSURL
        URLWithString:[NSString
        stringWithString:@" file:///tmp/project
        .html" ]]]];
683
684
685     if (![freshmeat isConnected])
686     {
687         [self showLoginSheet];
688     }
689

```

```

690 }
691
692 - (void) sortMyProjects
693 {
694     // [prefsProjectTable reloadData];
695     [allProjectsTable reloadData];
696 }
697
698 - (void)closeNow:(NSWindow *)sheet
699 {
700     if(closeAndQuit)
701         [NSApp terminate:nil];
702 }
703
704 - (void)showLoginSheet
705 {
706     [NSApp beginSheet: loginWindow
707         modalForWindow: projectWindow
708         modalDelegate: self
709         didEndSelector: @selector(closeNow:)
710         contextInfo: nil];
711     [NSApp runModalForWindow: loginWindow];
712     // Sheet is up here.
713     [NSApp endSheet: loginWindow];
714     [loginWindow orderOut: self];
715 }
716
717 - (IBAction)doLogin:(id)sender
718 {
719
720     NSString *login = [loginField stringValue];
721     NSString *password = [passwordField stringValue];
722
723     /* Handle exception that may raise on login
       failure... */
724 NS_DURING
725     [freshmeat login:login password:password];
726 NS_HANDLER
727     NSRunCriticalAlertPanel(NSLocalizedString(@"
       Login Incorrect",@"Login Incorrect"),
728                             NSLocalizedString(@"The
       username and
       password you
       entered could not
       be authenticated.
       Please try again
       .",@"Please try
       again"),
729                             nil, nil, nil);
730     return;
731 NS_ENDHANDLER
732
733     // or continue
734     [NSApp stopModal];
735     closeAndQuit = NO;
736     myProjects = [[NSMutableDictionary alloc]
       initWithDictionary:[ [freshmeat
       sessionDictionary] objectForKey:@"
       MyProjects" ]];
737     [self setReleaseAndLicenseMenus];
738     [allProjectsTable reloadData];
739 }
740
741 - (void)setReleaseAndLicenseMenus
742 {
743     licenseArray = [[NSArray alloc] initWithArray:[
       freshmeat fetch_available_licenses ]];
744
745     NSEnumerator *licenseEnum = [licenseArray
       objectEnumerator];
746     NSString *licenseKind;
747     NSMenu *licenseMenu = [[[NSMenu alloc] init]
       autorelease];
748     while(licenseKind = [licenseEnum nextObject]){
749         NSMenuItem *anItem = [[[NSMenuItem alloc]
       initWithTitle:licenseKind action:nil
       keyEquivalent:@""] retain];
750         [licenseMenu addItem:anItem];
751         [anItem release];
752     }
753     [nrLicensePopUp setMenu:licenseMenu];
754
755     /*releaseFocus = [[NSDictionary alloc]
       initWithDictionary:[freshmeat
       fetch_available_release_foci]];
756     NSMenu *releaseMenu = [[[NSMenu alloc] init]
       autorelease];
757     int i=0;
758     for (i=0; i<[releaseFocus count]; i++){
759         NSMenuItem *anItem = [[[NSMenuItem alloc]
       initWithTitle:[NSString
       stringWithFormat:@"%d - %@", i, [
       releaseFocus objectForKey:[NSString
       stringWithFormat:@"%d", i]]] action:
       nil keyEquivalent:@""] retain];
760         [releaseMenu addItem:anItem];
761         [anItem release];
762     }
763     [nrFocusPopUp setMenu:releaseMenu];*/
764 }
765
766 - (IBAction)doNewRelease:(id)sender
767 {
768     [nrProjectName setStringValue:currentProject];
769     [nrBranchName setStringValue:currentBranch];
770     [nrHideBtn setState:NSOffState];
771     [nrChangesTV setString:@""];
772     [nrVersionField setStringValue:@""];
773     [nrHomeField setStringValue:@""];
774     [nrTGZField setStringValue:@""];
775     [nrZipField setStringValue:@""];
776     [nrBZ2Field setStringValue:@""];
777     [nrChangelogField setStringValue:@""];
778     [nrRPMField setStringValue:@""];
779     [nrDEBField setStringValue:@""];
780     [nrOSXField setStringValue:@""];

```

```

782     [nrBSDField setStringValue:@""];
783     [nrCVSField setStringValue:@""];
784     [nrMLField setStringValue:@""];
785     [nrMirrorField setStringValue:@""];
786     [nrDemoField setStringValue:@""];
787
788
789
790     [nrLicensePopUp selectItemWithTitle:[
791         projectDictionary objectForKey:@"license
792         "]];
793
794     [NSApp beginSheet: nrWindow
795         modalForWindow: projectWindow
796         modalDelegate: nil
797         didEndSelector: nil
798         contextInfo: nil];
799     [NSApp runModalForWindow: nrWindow];
800     // Sheet is up here.
801     [NSApp endSheet: nrWindow];
802     [nrWindow orderOut: self];
803 }
804
805 - (IBAction)closeModal:(id)sender
806 {
807     [NSApp stopModal];
808 }
809
810 - (IBAction)doPublish:(id)sender
811 {
812     //
813     NSMutableDictionary *releaseInfo = [[
814         NSMutableDictionary alloc] init];
815     [releaseInfo setObject:[freshmeat
816         sessionDictionary] forKey:@"SID"];
817     [releaseInfo setObject:currentProject forKey:@"
818         project_name"];
819     [releaseInfo setObject:currentBranch forKey:@"
820         branch_name"];
821     [releaseInfo setObject:[nrVersionField
822         stringValue] forKey:@"version"];
823     [releaseInfo setObject:[nrChangesTV string]
824         forKey:@"changes"];
825     [releaseInfo setObject:[NSString
826         stringWithFormat:@"%d", [nrFocusPopUp
827         indexOfSelectedItem]] forKey:@"
828         release_focus"];
829     if ([nrHideBtn state])
830         [releaseInfo setObject:@"Y" forKey:@"
831         hide_from_frontpage"];
832     else
833         [releaseInfo setObject:@"N" forKey:@"
834         hide_from_frontpage"];
835     [releaseInfo setObject:[nrLicensePopUp
836         titleOfSelectedItem] forKey:@"license"];
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861

```

```

//url fields
NSString *optString = nil;
if (!([optString = [nrHomeField stringValue])
    isEqualToString:@""])
    [releaseInfo setObject:optString forKey:@"
    url_homepage"];
if (!([optString = [nrTGZField stringValue])
    isEqualToString:@""])
    [releaseInfo setObject:optString forKey:@"
    url_tgz"];
if (!([optString = [nrBZ2Field stringValue])
    isEqualToString:@""])
    [releaseInfo setObject:optString forKey:@"
    url_bz2"];
if (!([optString = [nrZipField stringValue])
    isEqualToString:@""])
    [releaseInfo setObject:optString forKey:@"
    url_zip"];
if (!([optString = [nrChangelogField stringValue]
    ]) isEqualToString:@""])
    [releaseInfo setObject:optString forKey:@"
    url_changelog"];
if (!([optString = [nrRPMField stringValue])
    isEqualToString:@""])
    [releaseInfo setObject:optString forKey:@"
    url_rpm"];
if (!([optString = [nrDEBField stringValue])
    isEqualToString:@""])
    [releaseInfo setObject:optString forKey:@"
    url_deb"];
if (!([optString = [nrOSXField stringValue])
    isEqualToString:@""])
    [releaseInfo setObject:optString forKey:@"
    url_osx"];
if (!([optString = [nrBSDField stringValue])
    isEqualToString:@""])
    [releaseInfo setObject:optString forKey:@"
    url_bsdprots"];
if (!([optString = [nrPurchaseField stringValue]
    ]) isEqualToString:@""])
    [releaseInfo setObject:optString forKey:@"
    url_purchase"];
if (!([optString = [nrCVSField stringValue])
    isEqualToString:@""])
    [releaseInfo setObject:optString forKey:@"
    url_cvs"];
if (!([optString = [nrMLField stringValue])
    isEqualToString:@""])

```

```

862         [releaseInfo setObject:optString forKey:@"
            url_list"];
863
864         if (![optString = [nrMirrorField stringValue])
            isEqualToString:@""])
865         [releaseInfo setObject:optString forKey:@"
            url_mirror"];
866
867         if (![optString = [nrDemoField stringValue])
            isEqualToString:@""])
868         [releaseInfo setObject:optString forKey:@"
            url_demo"];
869
870     NS_DURING
871         [freshmeat publish_release:releaseInfo];
872     NS_HANDLER
873         NSRunCriticalAlertPanel(NSLocalizedString(@"
            "Publish error",@"Publish error"),
874                                 NSLocalizedString(@"
            "Could not
            publish new
            release !" ,@"
            Could not
            publish new
            release !" ),
875                                 nil, nil, nil);
876     return;
877     NS_ENDHANDLER
878
879     [releaseInfo release];
880
881     [NSApp stopModal];
882 }
883
884 - (IBAction)doWithdraw:(id)sender
885 {
886     if (NSRunCriticalAlertPanel(NSLocalizedString(@"
            "Confirm withdraw",@"Withdraw confirmation
            " ),
887                                 NSLocalizedString(@"
            "Are you sure
            you want to
            withdraw this
            release ?" ,@"
            Are you sure
            you want to
            withdraw this
            release ?" ),
888                                 nil, NSLocalizedString(@"
            "Abort",@"
            Abort" ), nil)
            != NSOKButton)
889     return;
890     NS_DURING
891         [freshmeat withdraw_release_for_project:
            currentProject branch:currentBranch
            version:[projectDictionary
            objectForKey:@"latest_release_version
            "]];
892     NS_HANDLER
893         NSRunCriticalAlertPanel(NSLocalizedString(@"
            "Withdraw error",@"Withdraw error"),
894                                 NSLocalizedString(@"
            "Could not
            withdraw
            latestst
            release for
            the selected
            project",@"
            "Could not
            withdraw
            latestst
            release for
            the selected
            project"),
895                                 nil, nil, nil);
896     return;
897     NS_ENDHANDLER
898
899     - (void)downloadXMLFileForProject:(NSString *)
        project
900     {
901         [spinWheel startAnimation: self];
902         NSString *urlString = [NSString
            stringWithFormat:@"http://freshmeat.net/
            projects-xml/%@/%@.xml", project, project
            ];
903         NSURL *theURL = [NSURL URLWithString:
            urlString];
904         DownloadFile *download = [ [ DownloadFile
            alloc ] initWithURL: theURL ] autorelease
            ];
905         [download setDelegate: self];
906
907         - (void)displayXMLData:(NSData *)theData
908         {
909             XMLFeeder *parser = [ [ XMLFeeder alloc ]
            initWithData: theData ] autorelease ];
910             BOOL result = NO;
911             if ( parser ) {
912                 result = [ parser parse ];
913                 if ( result == NO ) {
914                     NSError *theErr = [ parser
            parserError ];
915                     if ( [ theErr code ] != 0 ) {
916                         NSLog( [ theErr
            localizedDescription ] );
917                     }
918                 } else { /* Render HTML display */
919                     if (projectDictionary)
920                         [projectDictionary release];
921                     projectDictionary = [[NSDictionary
            alloc] initWithDictionary:[parser

```

	projectDictionary]];	myProjects	
926	[self renderHTMLWithDictionary:		
	projectDictionary];	objectForKey	
927	//NSLog( [ [ parser projectDictionary	:	
	] description ] );	currentProject	
928	//NSLog( [ [ parser projectItems ]	]	
	description ] );	objectForKey	
929	}	:@"	
930	}	project	
931	}	.info	
932		"]	
933	-(void)renderHTMLWithDictionary:( NSDictionary *)	objectForKey	
	aDict	:@"	
934	{	project	
935	NSString *percent=@"%";	.	
936	NSString *bgPath = [NSBundle pathForResource:@"	version	
	bg" ofType:@"tif" inDirectory:[ [NSBundle	"]];	
	mainBundle] bundlePath]];		946
937	if ( nil == aDict){	NSString *changelog = [NSString	
938	NSString *projectHTML = [NSString	stringWithString:[releaseInfo	
	stringWithFormat:@"<HTML><HEAD></HEAD>	objectForKey:@"changes"]];	
	<BODY MARGINHEIGHT=0 MARGINWIDTH=0	NSString *projectHTML = [NSString	
	BACKGROUND=%@ BGCOLOR=#FFFFFF></BODY>	stringWithFormat:@"\	
	></HTML>", bgPath];	<HTML>\n\	
939	[projectHTML writeToFile:@" /tmp/project.	<HEAD>\n\	
	html" atomically:NO];	</HEAD>\n\	
940	[[releaseWebView mainFrame] loadRequest:	<BODY MARGINHEIGHT=0 MARGINWIDTH=0	
941	[NSURLRequest requestWithURL:[NSURL	BACKGROUND=%@ BGCOLOR=#FFFFFF>\n\	
	URLWithString:[ NSString	<TABLE BORDER=0 WIDTH=100%@ CELLSPACING=0>\	
	stringWithString:@" file:///tmp/	n\	
	project.html" ] ] ] ];	<TR><TD VALIGN=TOP WIDTH=50%>\n\	
942	} else {	<H2>%@</H2>\n\	
943	NSDictionary *releaseInfo = [freshmeat	<H5>Entry Date: %@ \n\	
	fetch_release_for_project:[ [	Last Update: %@ \n\	
	myProjects objectForKey:currentProject	Latest release version: %@</H5>\n\	
	] objectForKey:@"project.info"]	<FONT SIZE=2 COLOR=#000000>\n\	
	objectForKey:@"project.shortname"]	License: %@ \n\	
944	branch	Subscribers: %@ \n\	
	: [	</FONT>\n\	
	myProjects	</TD>\n\	
		<TD VALIGN=TOP WIDTH=50%@ ALIGN=RIGHT>\n\	
	objectForKey	<IMG SRC=%@ WIDTH=150>\n\	
	:	</TD>\n\	
	currentProject	</TR>\n\	
	]	<TR><TD COLSPAN=2> \n\	
	objectForKey	<FONT SIZE=3 COLOR=#000000><B>Description	
	:@"	:</B></FONT> \n\	
	project	<FONT SIZE=2 COLOR=#000000>%@  \n\	
	.	<FONT SIZE=3 COLOR=#000000><B>Changelog:</B>	
	branches	></FONT> \n\	
	"]	<FONT SIZE=2 COLOR=#000000>%@  \n\	
	objectAtIndex	<B>Release focus: %@</B></FONT>\n\	
	: [	</TD>\n\	
	projectBranchTable	</TABLE>\n\	
		<BODY>\n\	
	selectedRow	</HTML>\n", bgPath, percent, percent,	
	]	[aDict objectForKey:@"projectname_full	
945	version	"], [aDict objectForKey:@"	
	: [ [	date_added"], [aDict objectForKey:	
		@ "date.updated"]. [aDict	



```

        objectForKey:@"latest_release_version"], [aDict
        objectForKey:@"license"],
978    [aDict objectForKey:@"subscriptions"]1017
        percent, [aDict objectForKey:@"screenshot_thumb"], [aDict
        objectForKey:@"desc_full"],
        changelog, [releaseInfo
        objectForKey:@"release_focus"]];
979    [projectHTML writeToFile:@"tmp/project.1019
        html" atomically:NO];
980    [[releaseWebView mainFrame] loadRequest:1021 }
981    [NSURLRequest requestWithURL:[NSURL
        URLWithString:[NSString
        stringWithString:@"file:///tmp/
        project.html"]]];
982    }
983 }
984
985 #pragma mark DownloadFile delegates
986 - (void)downloadFileDidFinish:(DownloadFile *)1027
        download
987 {
988     [self displayXMLData:[download contentsData
        ]];
989     [newReleaseBtn setEnabled:YES];
990     [withdrawBtn setEnabled:YES];
991     [spinWheel stopAnimation:self];
992 }
993
994 - (void)downloadFile:(DownloadFile *)download1029
        didFailWithError:(NSError *)error
995 {
996     [spinWheel stopAnimation:self];
997 }
998
999
1000
1001 #pragma mark Tableviews Datasource & delegates
1002 - (int)numberOfRowsInTableView:(NSTableView *)1037
        aTableView
1003 {
1004     if ([aTableView isEqualTo:allProjectsTable]){1038
1005         return [myProjects count];
1006     } else if ([aTableView isEqualTo:
        projectBranchTable] && (nil !=
        currentProject)) {
1007         return [[[myProjects objectForKey:
        currentProject] objectForKey:@"projed1044 }
        .branches"] count];
1008     } else
1009         return 0;
1010 }
1011
1012
1013 -(id)tableView:(NSTableView *)aTableView
        objectValueForTableColumn:(NSTableColumn *)
        aTableColumn row:(int)rowIndex
1014 {
        objectForKey:@"release_info"]];
        objectForKey:@"project
        .branches"] objectAtIndex:rowIndex];
        return nil;
        - (void)tableViewSelectionDidChange:(NSNotification
        *)notification
        {
        if ([[notification object] isEqualTo:
        projectBranchTable]){
        if ([projectBranchTable selectedRow] != -1)
        {
        currentBranch = [[[myProjects
        objectForKey:currentProject]
        objectForKey:@"project.branches"]
        objectAtIndex:[projectBranchTable
        selectedRow]];
        [self downloadXMLFileForProject:[[[
        myProjects objectForKey:
        currentProject] objectForKey:@"
        project.info"] objectForKey:@"
        project.shortname"]]];
        } else {
        currentBranch=nil;
        [self renderHTMLWithDictionary:nil];
        [newReleaseBtn setEnabled:NO];
        [withdrawBtn setEnabled:NO];
        }
        } else if ([[notification object] isEqualTo:
        allProjectsTable]){
        if ([allProjectsTable selectedRow] != -1){
        currentProject = [[myProjects allKeys]
        objectAtIndex:[allProjectsTable
        selectedRow]];
        } else {
        currentProject = nil;
        }
        [projectBranchTable reloadData];
        [projectBranchTable deselectAll:nil];
        }
        }
        @end
        //
        XMLFeeder.h
        Freshmint
        //
        // Created by Jean-Matthieu
        //
        #import <Foundation/Foundation.h>

```

```

1055
1056
1057 @interface XMLFeeder : NSXMLParser {
1058     int            m_stat;
1059     NSDictionary   *m_projectDictionary;
1060     NSMutableArray *m_projectItems;
1061
1062     NSString        *m_name;
1063     NSMutableString *m_value;
1064
1065     NSMutableDictionary *m_elemValue;
1066     NSMutableArray     *m_elements;
1067 }
1068
1069 - (XMLFeeder *)initWithURL:(NSURL *)url;
1070 - (XMLFeeder *)initWithData:(NSData *)data;
1071
1072 - (NSDictionary *)projectDictionary;
1073 - (NSMutableArray *)projectItems;
1074
1075 @end
1076 //
1077 // XMLFeeder.m
1078 // Freshmint
1079 //
1080 // Created by Jean-Matthieu
1081 //
1082
1083 #import "XMLFeeder.h"
1084 #import "NSStringEx.h"
1085
1086 @implementation XMLFeeder
1087
1088 - (XMLFeeder *)initWithURL:(NSURL *)url
1089 {
1090     self = [ super initWithContentsOfURL: url ];
1091     if ( self ) {
1092         [ self setDelegate: self ];
1093         m_stat = 0;
1094     }
1095     return self;
1096 }
1097
1098 - (XMLFeeder *)initWithData:(NSData *)data
1099 {
1100     self = [ super initWithData: data ];
1101     if ( self ) {
1102         [ self setDelegate: self ];
1103         m_stat = 0;
1104     }
1105     return self;
1106 }
1107
1108 - (void)dealloc
1109 {
1110     [ m_projectDictionary release ];
1111     [ m_projectItems release ]; [ super dealloc
1112 }
1113
1114 - (NSDictionary *)projectDictionary
1115 {
1116     return [ m_projectItems objectAtIndex:0];
1117 }
1118
1119 - (NSMutableArray *)projectItems
1120 {
1121     return ( m_projectItems );
1122 }
1123
1124
1125
1126 #pragma mark -
1127 #pragma mark == Delegate ==
1128 - (void)parserDidStartDocument:(NSXMLParser *)
1129     parser
1130 {
1131     // NSLog( @"parserDidStartDocument" );
1132
1133     m_stat = 0;
1134     m_value = [ NSMutableString string ];
1135     m_elemValue = [ NSMutableDictionary dictionary
1136     ];
1137     m_elements = [ NSMutableArray array ];
1138 }
1139
1140 - (void)parserDidEndDocument:(NSXMLParser *)parser
1141 {
1142     // NSLog( @"parserDidEndDocument" );
1143
1144     m_projectItems = [ m_elements copy ];
1145 }
1146
1147 - (void)parser:(NSXMLParser *)parser
1148     didStartElement:(NSString *)elementName
1149     namespaceURI:(NSString *)namespaceURI
1150     qualifiedName:(NSString *)qName attributes:(
1151     NSDictionary *)attributeDict
1152 {
1153     // NSLog( @"didStartElement : %@, %@, %@, %@",
1154     //     elementName, namespaceURI, qName, [
1155     //     attributeDict description ] );
1156
1157     switch ( m_stat ) {
1158     case 0:
1159         // header.
1160         if ( [ elementName isEqualToString: @"
1161         project-listing" ] ) {
1162             }
1163             m_stat++;
1164             break;
1165     case 1:
1166         // header elements.
1167         if ( [ elementName isEqualToString: @"
1168         project" ] ) {
1169             // go next.
1170             m_projectDictionary = [ m_elemValue
1171             copy ];

```

## MPIObjC source code

Oxford Brookes University

```

20     MPIComm comm;
18
19 @interface MPIComm : NSObject {
21
22     }
23
24 #pragma mark -
25 #pragma mark Constructor
26 /**
27  * Initiate a MPIComm object.
28  * @param aComm: A communicator, such as
29     MPI_COMM_WORLD, MPI_COMM_SELF, MPI_COMM_NULL
30     ...
29  * @return Returns an MPIComm instance; a
31     communicator for your MPI environment.
32  */
33 #pragma mark -
34 #pragma mark Communicator functions
35
36 /**
37  * Communicator size.
38  * Determines the size of the group associated with
39     a communicator.
39  * @return The size of the group as a NSNumber.
40  */
41 - (NSNumber *) MPICommSize;
42
43 /**
44  * Process rank.
45  * Determines the rank of the calling process in the
46     communicator.
46  * @return The rank of the calling process as a
47     NSNumber.
47  */
48 - (NSNumber *) MPICommRank;
49
50 /**
51  * Communicator duplicator.
52  * Duplicates an existing communicator with all its
53     cached information.
53  * @return A duplicated MPIComm object.
54  */
55 - (MPIComm *) MPICommDup;
56
57 /**
58  * Communicator splitter
59  * Creates new communicators based on colors and
60     keys.
60  * @param color: An integer to specify the color,
61     control of subset assignment. The
61  * color must be non-negative or MPL_UNDEFINED.
62  * @param aKey: An integer to specify the key,
63     control of rank assignment.
63  * @return A new MPIComm instance.
64  */
65 - (MPIComm *) MPICommSplit:(int)color andKey:(int)
66     aKey;
67 /**
68  * Free a communicator.
69  * Marks the communicator object for deallocation
70  */
71 - (void) MPICommFree;
72
73 #pragma mark -
74 #pragma mark Point-to-Point Communications (
75     Blocking)
76 /**
77  * Sending data.
78  * Performs a basic send.
79  * @param message: Message to send.
80  * @param messageSize: Number of element in the sent
81     message.
81  * @param type: The message type (ie: MPL_CHAR,
82     MPL_INT ...).
82  * @param dest: Rank of process to send the data to
83     (integer).
83  * @param tag: Message tag.
84  */
85 - (void) MPISend:(void *)message ofSize:(int)
86     messageSize ofType:(MPI_Datatype)type
87     toProcess:(int)dest withTag:(int)tag;
88 /**
89  * Receiving data.
90  * Basic receive from a process.
91  * @param outBuffer: A buffer to store the received
92     message.
92  * @param messageSize: The expected message size.
93  * @param type: The message type (ie: MPL_CHAR,
94     MPL_INT ...).
94  * @param src: Rank of the sending process.
95  * @param tag: Message tag.
96  * @return
97  */
98 - (void) MPIRecv:(void *)outBuffer ofSize:(int)
99     messageSize type:(MPI_Datatype)type from:(int)
100     src withTag:(int)tag;
101 /**
102  * Send and receive a message.
103  * This method sends and receive a message.
104  * @param message: Message to send.
105  * @param sMessageSize: Number of element in the
106     sent message.
106  * @param sType: The message type (ie: MPL_CHAR,
107     MPL_INT ...)
107  * @param dest: Rank of process to send the data to
108     (integer).

```

```

108 * @param sTag: Message tag.                                     messageSize type:(MPI_Datatype)type from:(int)
109 * @param outBuffer: A buffer to store the received             src withTag:(int)tag;
    message.                                                       151
110 * @param rMessageSize: The expected message size.             152 #pragma mark -
111 * @param rType: The message type (ie: MPLCHAR,                 153 #pragma mark Collective Communications
    MPLINT ...).                                                    154
112 * @param src: Rank of the sending process.                     155 /**
113 * @param rtag: Message tag.                                     156 * Process Synchronization
114 * @return                                                  157 * Performs a barrier synchronization among all
115 */                                                            process in the communicator.
116 - (void) MPISendRecv:(void *)message ofSize:(int)             158 */
    sMessageSize ofType:(MPI_Datatype)sType                         159 - (void) MPIBarrier;
    toProcess:(int)dest withTag:(int)sTag                           160
    outMessage:(void *)outBuffer outSize:(int)                     161 /**
    rMessageSize type:(MPI_Datatype)rType from:(int)               162 * Message Broadcast.
    int)src withTag:(int)rtag;                                       163 * Broadcast a message to all process in the
                                                                    communicator world.
117                                                                164 * @param message: Message to send.
118 /**                                                            165 * @param size: Number of elements in the buffer.
119 * Get the number of received elements.                           166 * @param type: The message type (ie: MPLCHAR,
    MPLINT ...).                                                    MPLINT ...).
120 * @param forType: The message type (ie: MPLCHAR,               167 * @param rootProcess: Rank of process with message
    MPLINT ...).                                                    to broadcast.
121 * @return The number of received elements as a                 168 */
    NSNumber.
122 */
123 - (NSNumber *) MPIGetCount:(MPI_Datatype)forType;             169
124
125 #pragma mark -
126 #pragma mark Point-to-Point Communications (Non-
    Blocking)
127
128 /**
129 * Sending data.
130 * Performs a basic send.
131 * @param message: Message to send.
132 * @param messageSize: Number of element in the sent
    message.
133 * @param type: The message type (ie: MPLCHAR,
    MPLINT ...).
134 * @param dest: Rank of process to send the data to
    (integer).
135 * @param tag: Message tag.
136 */
137 - (void) MPIISend:(void *)message ofSize:(int)
    messageSize ofType:(MPI_Datatype)type
    toProcess:(int)dest withTag:(int)tag;
138
139
140 /**
141 * Receiving data.
142 * Basic receive from a process.
143 * @param A buffer to store the received message.
144 * @param The expected message size.
145 * @param The message type (ie: MPLCHAR, MPLINT
    ...).
146 * @param Rank of the sending process.
147 * @param tag: Message tag.
148 * @return
149 */
150 - (void) MPIIRecv:(void *)outBuffer ofSize:(int)
    messageSize ofType:(MPI_Datatype)type from:(int)
    src withTag:(int)tag;
    151
    152 #pragma mark -
    153 #pragma mark Collective Communications
    154
    155 /**
    156 * Process Synchronization
    157 * Performs a barrier synchronization among all
    process in the communicator.
    158 */
    159 - (void) MPIBarrier;
    160
    161 /**
    162 * Message Broadcast.
    163 * Broadcast a message to all process in the
    communicator world.
    164 * @param message: Message to send.
    165 * @param size: Number of elements in the buffer.
    166 * @param type: The message type (ie: MPLCHAR,
    MPLINT ...).
    167 * @param rootProcess: Rank of process with message
    to broadcast.
    168 */
    169
    170 - (void) MPIBcast:(void *)message ofSize:(int)size
    ofType:(MPI_Datatype)type rank:(int)
    rootProcess;
    171
    172 #pragma mark Message gathering
    173
    174 /**
    175 * Basic Message gathering.
    176 * Gather messages from all process in the
    communicator.
    177 * @param message: Message to send.
    178 * @param count: Number of element in the sent
    message.
    179 * @param type: The sent message type (ie: MPLCHAR
    , MPLINT ...).
    180 * @param outBuffer: A buffer to store the received
    message.
    181 * @param outSize: Number of element in the received
    buffer.
    182 * @param outType: The received message type (ie:
    MPLCHAR, MPLINT ...).
    183 * @param rootProcess: Rank of gathering process.
    184 * @return
    185 */
    186
    187 - (void) MPIGather:(void *)message ofSize:(int)
    count ofType:(MPI_Datatype)type outMessage:(
    void *) outBuffer outSize:(int)outSize outType:
    :(MPI_Datatype)outType rank:(int)rootProcess;
    188
    189 /**
    190 * More complex message gathering.
    191 * Gather message with variable length from all
    process in the communicator.

```

```

192 * @param message: Message to send.
193 * @param count: Number of element in the sent
    message.
194 * @param type: The sent message type (ie: MPLCHAR
    , MPLINT ...).
195 * @param outBuffer: A buffer to store the received
    message.
196 * @param outSize: Number of element in the received
    buffer.
197 * @param displs: Displacement in received message
    of elements gathered from all processes.
198 * @param outType: The received message type (ie:
    MPLCHAR, MPLINT ...).
199 * @param rootProcess: Rank of gathering process.
200 * @return
201 */
202
203 - (void) MPIGatherv:(void *)message ofSize:(int)
    count ofType:(MPI_Datatype)type outMessage:(
    void*)outBuffer outSize:(int *)outSize
    displacement:(int*) displs outType:(
    MPI_Datatype)outType rank:(int)rootProcess;
204
205 /**
206 * All-gather operation.
207 * Gather messages from all process in all proccess
    in the communicator.
208 * @param message: Message to send.
209 * @param count: Number of element in the sent
    message.
210 * @param type: The sent message type (ie: MPLCHAR
    , MPLINT ...).
211 * @param outBuffer: A buffer to store the received
    message.
212 * @param outSize: Number of element in the received
    buffer.
213 * @param outType: The received message type (ie:
    MPLCHAR, MPLINT ...).
214 * @return
215 */
216
217 - (void) MPIAllGather:(void *)message ofSize:(int)
    count ofType:(MPI_Datatype)type outMessage:(
    void*)outBuffer outSize:(int)outSize outType:(
    MPI_Datatype)outType;
218
219
220 #pragma mark Message scattering
221
222 /**
223 * Basic Message scattering.
224 * Distribute individual messages from root to each
    process in the communicator.
225 * @param message: Message to send.
226 * @param count: Number of element in the sent
    message.
227 * @param type: The sent message type (ie: MPLCHAR
    , MPLINT ...).
228 * @param A buffer to store the received message.
229 * @param outSize: Number of element in the received
    buffer.
230 * @param outType: The received message type (ie:
    MPLCHAR, MPLINT ...).
231 * @param rootProcess: Rank of scattering process.
232 * @return
233 */
234
235 - (void) MPIScatter:(void *)message ofSize:(int)
    count ofType:(MPI_Datatype)type outMessage:(
    void*)outBuffer outSize:(int)outSize outType:(
    MPI_Datatype)outType rank:(int)rootProcess;
236
237 /**
238 * Complex message scattering
239 * Distribute individual messages from root to each
    process in the communicator.
240 * Messages can have different sizes and
    displacements.
241 * @param message: Message to send.
242 * @param count: Number of element in the sent
    message.
243 * @param displs: Displacement in sent message.
244 * @param type: The sent message type (ie: MPLCHAR
    , MPLINT ...).
245 * @param A buffer to store the received message.
246 * @param outSize: Number of element in the received
    buffer.
247 * @param outType: The received message type (ie:
    MPLCHAR, MPLINT ...).
248 * @param rootProcess: Rank of scattering process.
249 * @return
250 */
251
252 - (void) MPIScatterv:(void *)message ofSize:(int*)
    count displacement:(int*) displs ofType:(
    MPI_Datatype)type outMessage:(void*)outBuffer
    outSize:(int)outSize outType:(MPI_Datatype)
    outType rank:(int)rootProcess;
253
254
255 #pragma mark Reductions
256
257 /**
258 * Reduction computation.
259 * Reduces values on all processes to a single value
    .
260 * @param message: Message to send.
261 * @param outBuffer: A buffer to store the received
    message.
262 * @param size: Number of elements in the received
    message.
263 * @param type: The message datatype.
264 * @param operation: The reduction operation.
265 * @param rank: Rank of the reducing process.
266 * @return
267 */
268
269 - (void) MPIReduce:(void *)message outMessage:(void

```

```

        *)outBuffer ofSize:(int)size ofType:(
MPI.Datatype)type withOp:(MPI.Op)operation
rank:(int)rank;
270
271 /**
272 * Reduction computation.
273 * Combines values from all processes and
    distributes the result back to all processes.
274 * @param message: Message to send.
275 * @param outBuffer: A buffer to store the received
    message.
276 * @param size: Number of elements in the received
    message.
277 * @param type: The message datatype.
278 * @param operation: The reduction operation.
279 * @return
280 */
281
282 - (void) MPIAllReduce:(void *)message outMessage:(
    void*)outBuffer ofSize:(int)size ofType:(
MPI.Datatype)type withOp:(MPI.Op)operation;
283
284 /**
285 * Reduces and scatters a message.
286 * Combines values and scatters the results.
287 * @param message: Message to send.
288 * @param outBuffer: A buffer to store the received
    message.
289 * @param rSize: Number of elements in the received
    message.
290 * @param type: The message datatype.
291 * @param operation:The reduction operation.
292 * @return
293 */
294
295 - (void) MPIReduceScatter:(void *)sMessage
    outMessage:(void*)outBuffer outSize:(int *)
    rSize ofType:(MPI.Datatype)type withOp:(MPI.Op
    ) operation;
296
297 /**
298 * Partial reduction computation
299 * Computes the scan (partial reductions) of data
    on a collection of processes.
300 * @param message: Message to send.
301 * @param outBuffer: A buffer to store the received
    message.
302 * @param size: Number of elements in the received
    message.
303 * @param type: The message datatype.
304 * @param operation:The reduction operation.
305 * @return
306 */
307
308 - (void)MPIScan:(void*)message outMessage:(void*)
    outBuffer ofSize:(int)size ofType:(
MPI.Datatype)type withOp:(MPI.Op)operation;
309
310 #pragma mark All to all communications
311
312 /**
313 * Sends data from all to all processes.
314 * @param message: Message to send.
315 * @param inSize: Number of element in the sent
    message.
316 * @param inType: Type of sent message.
317 * @param outBuffer: A buffer to store the received
    message.
318 * @param outSize: Size of the received message.
319 * @param outType: Type of received message.
320 * @return
321 */
322
323 - (void) MPIAllToAll:(void *)message ofSize:(int)
    inSize ofType:(MPI.Datatype)inType outMessage
    :(void*)outBuffer outSize:(int)outSize
    outType:(MPI.Datatype)outType;
324
325 /**
326 * Sends data from all to all processes , with a
    displacement.
327 * @param message: Message to send.
328 * @param inSize: Number of element in the sent
    message.
329 * @param inDispls: Displacement in sent message.
330 * @param inType: Type of sent message.
331 * @param outBuffer: A buffer to store the received
    message.
332 * @param outSize: Size of the received message.
333 * @param outDispls: Displacement in received
    message.
334 * @param outType: Type of received message.
335 * @return
336 */
337 - (void) MPIAllToAllv:(void *)message ofSize:(int*)
    inSize inDispls:(int*)inDispls ofType:(
MPI.Datatype)inType outMessage:(void*)
    outBuffer outSize:(int *)outSize outDispls:(
    int*)outDispls outType:(MPI.Datatype)outType;
338
339
340 @end
341 //
342 // MPIComm.m
343 // MPIObjC
344 //
345 // Created by Jean-Matthieu on 14/08/2004.
346 // Copyright 2004 __MyCompanyName__. All rights
    reserved.
347 //
348
349 #import "MPIComm.h"
350 #import "MPIInstance.h"
351
352 @implementation MPIComm
353 - (id) initWithCommunicator:(MPIComm)aComm
354 {
355     self = [super init];

```

```

356     if (self)
357     {
358         comm = aComm;
359         return self;
360     }
361     return nil;
362 }
363
364 - (void) dealloc
365 {
366     MPI_Comm_free(&comm);
367     [super dealloc];
368 }
369
370
371 #pragma mark -
372 #pragma mark Communicator functions
373
374 - (NSNumber *) MPICommSize
375 {
376     int size = 0;
377     MPI_Comm_size(comm, &size);
378     NSNumber *theSize = [[NSNumber alloc]
379         initWithInt:size] autorelease];
379     return theSize;
380 }
381
382 - (NSNumber *) MPICommRank
383 {
384     int rank = 0;
385     MPI_Comm_rank(comm, &rank);
386     NSNumber *theRank = [[NSNumber alloc]
387         initWithInt:rank] autorelease];
387     return theRank;
388 }
389 }
390
391 - (MPIComm *) MPICommDup
392 {
393     MPLComm newComm = MPLCOMM_NULL;
394     MPI_Comm_dup(comm, &newComm);
395     MPIComm *aComm = [[MPIComm alloc]
396         initWithCommunicator:newComm];
396     return [aComm autorelease];
397 }
398
399 - (MPIComm *) MPICommSplit:(int) color andKey:(int)
400     aKey
401 {
402     MPLComm newComm = MPLCOMM_NULL;
403     MPI_Comm_split(comm, color, aKey, &newComm);
404     MPIComm *aComm = [[MPIComm alloc]
405         initWithCommunicator:newComm] autorelease];
405     return [aComm autorelease];
406 }
407
408 - (void) MPICommFree
409 {
410     MPI_Comm_free(&comm);
411 }
412
413 #pragma mark -
414 #pragma mark Point-to-Point Communications (
415     Blocking)
416
417 - (void) MPISend:(void *)message ofSize:(int)
418     messageSize ofType:(MPI_Datatype)type
419     toProcess:(int)dest withTag:(int)tag
420 {
421     MPI_Send(message, messageSize, type, dest, tag,
422         comm);
423 }
424
425 - (void) MPIRecv:(void *)outBuffer ofSize:(int)
426     messageSize type:(MPI_Datatype)type from:(int)
427     src withTag:(int)tag
428 {
429     MPI_Status status;
430     MPI_Recv(outBuffer, messageSize, type, src, tag,
431         comm, &status);
432     [[MPIInstance getInstance] setStatus:status];
433 }
434
435 - (void) MPISendRecv:(void *)message ofSize:(int)
436     sMessageSize ofType:(MPI_Datatype)sType
437     toProcess:(int)dest withTag:(int)sTag
438     outMessage:(void *)outBuffer outSize:(int)
439     rMessageSize type:(MPI_Datatype)rType from:(int)
440     src withTag:(int)rTag{
441     [self MPISend:message ofSize:sMessageSize
442         ofType:sType toProcess:dest withTag:sTag];
443     [self MPIRecv:outBuffer ofSize:rMessageSize
444         type:rType from:src withTag:rTag];
445 }
446
447 - (NSNumber *) MPIGetCount:(MPI_Datatype)forType
448 {
449     MPI_Status status = [[MPIInstance getInstance]
450         status];
451     int count = 0;
452     MPI_Get_count(&status, forType, &count);
453     NSNumber *theCount = [[NSNumber alloc]
454         initWithInt:count] autorelease];
454     return theCount;
455 }
456
457 #pragma mark -
458 #pragma mark Point-to-Point Communications (
459     Blocking)
460
461 - (void) MPIISend:(void *)message ofSize:(int)
462     messageSize ofType:(MPI_Datatype)type
463     toProcess:(int)dest withTag:(int)tag{
464     NSString *dictionaryKey = [NSString
465         stringWithFormat:@"%S-%d", tag];
466     MPI_Request request;

```



```

448                                     487 {
449     MPI_Issend(message, messageSize, type, dest, 488     MPI_Gatherv(message, count, type, outBuffer,
        tag, comm, &request);                outSize, displs, outType, rootProcess,
450                                     comm);
451     MPIRequest *aRequest = [[[MPIRequest alloc] 489 }
        initWithRequest:request] autorelease]; 490
452     [[[MPIInstance getInstance] requestDictionary]491 - (void) MPIAllGather:(void *)message ofSize:(int)
        setObject:aRequest forKey:dictionaryKey]; count ofType:(MPI_Datatype)type outMessage:(
453 }                                           void*)outBuffer outSize:(int)outSize outType:(
454                                           MPI_Datatype)outType
455 - (void) MPIIrecv:(void*)outBuffer ofSize:(int) 492 {
        messageSize type:(MPI_Datatype)type from:(int)493     MPI_Allgather(message, count, type, outBuffer,
        src withTag:(int)tag                outSize, outType, comm);
456 {                                           494 }
457     NSString *dictionaryKey = [NSString
        stringWithFormat:@"%S-%d", tag]; 495
458     MPI_Request request; 496
459     MPI_Irecv(outBuffer, messageSize, type, src, 497 #pragma mark Message scattering
        tag, comm, &request); 498 - (void) MPIScatter:(void *)message ofSize:(int)
600 count ofType:(MPI_Datatype)type outMessage:(
460                                           void*)outBuffer outSize:(int)outSize outType:(
461     MPIRequest *aRequest = [[[MPIRequest alloc] MPI_Datatype)outType rank:(int)rootProcess
        initWithRequest:request] autorelease]; 499 {
462     [[[MPIInstance getInstance] requestDictionary]500     MPI_Scatter(message, count, type, outBuffer,
        setObject:aRequest forKey:dictionaryKey]; outSize, outType, rootProcess, comm);
463 }                                           501 }
464                                           502
465                                           503 - (void) MPIScatterv:(void *)message ofSize:(int*)
466 #pragma mark - count displacement:(int*) displs ofType:(
467 #pragma mark Collective Communications MPI_Datatype)type outMessage:(void*)outBuffer
468 outSize:(int)outSize outType:(MPI_Datatype)
469 - (void) MPIBarrier outType rank:(int)rootProcess
470 {                                           504 {
471     MPI_Barrier(comm); 505     MPI_Scatterv(message, count, displs, type,
472 } outBuffer, outSize, outType, rootProcess,
473 comm);
474 - (void) MPIBcast:(void *)message ofSize:(int)size506 }
        ofType:(MPI_Datatype)type rank:(int) 507
        rootProcess 508
475 {                                           509 #pragma mark Reductions
476     MPI_Bcast(message, size, type, rootProcess, 510 - (void) MPIReduce:(void *)message outMessage:(void
        comm); *)outBuffer ofSize:(int)size ofType:(
477 } MPI_Datatype)type withOp:(MPI_Op)operation
478 rank:(int)rank
479 #pragma mark Message gathering 511 {
480 512     MPI_Reduce(message, outBuffer, size, type,
481 - (void) MPIGather:(void *)message ofSize:(int) operation, rank, comm);
        count ofType:(MPI_Datatype)type outMessage:( 513 }
        void*)outBuffer outSize:(int)outSize outType:( 514
        MPI_Datatype)outType rank:(int)rootProcess 515 - (void) MPIAllReduce:(void *)message outMessage:(
482 { void*)outBuffer ofSize:(int)size ofType:(
483     MPI_Gather(message, count, type, outBuffer, MPI_Datatype)type withOp:(MPI_Op)operation
        outSize, outType, rootProcess, comm); 516 {
484 } 517     MPI_Allreduce(message, outBuffer, size, type,
485 operation, comm);
486 - (void) MPIGatherv:(void *)message ofSize:(int) 518 }
        count ofType:(MPI_Datatype)type outMessage:( 519
        void*)outBuffer outSize:(int *)outSize 520 - (void) MPIReduceScatter:(void *)sMessage
        displacement:(int*) displs outType:( outMessage:(void*)outBuffer outSize:(int *)
        MPI_Datatype)outType rank:(int)rootProcess rSize ofType:(MPI_Datatype)type withOp:(MPI_Op

```

```

        ) operation
521 {
522     MPI_Reduce_scatter(sMessage, outBuffer, rSize,
        type, operation, comm);
523 }
524
525 - (void) MPI_Scan:(void *)message outMessage:(void *)
        outBuffer ofSize:(int) size ofType:(
        MPI_Datatype) type withOp:(MPI_Op) operation
526 {
527     MPI_Scan(message, outBuffer, size, type,
        operation, comm);
528 }
529
530 #pragma mark All to all communications
531 - (void) MPI_AllToAll:(void *)message ofSize:(int)
        inSize ofType:(MPI_Datatype) inType outMessage
        :(void *) outBuffer outSize:(int) outSize
        ofType:(MPI_Datatype) outType
532 {
533     MPI_Alltoall(message, inSize, inType, outBuffer
        , outSize, outType, comm);
534 }
535
536 - (void) MPI_AllToAllv:(void *)message ofSize:(int *)
        inSize inDispls:(int *) inDispls ofType:(
        MPI_Datatype) inType outMessage:(void *)
        outBuffer outSize:(int *) outSize outDispls:(
        int *) outDispls ofType:(MPI_Datatype) outType
537 {
538     MPI_Alltoallv(message, inSize, inDispls, inType
        , outBuffer, outSize, outDispls, outType,
        comm);
539 }
540
541 @end
542 /*
543 * MPICommTest.h
544 * MPIObjC
545 *
546 * Created by Jean-Matthieu on 16/08/2004.
547 * Copyright 2004 __MyCompanyName__. All rights
        reserved.
548 *
549 */
550
551 #include <Carbon/Carbon.h>
552
553 //
554 // MPIInstance.h
555 // MPIObjC
556 //
557 // Created by Jean-Matthieu on 14/08/2004.
558 // Copyright 2004 __MyCompanyName__. All rights
        reserved.
559 //
560
561 #import <Foundation/Foundation.h>
562 #import "mpi.h"

```

```

616 * when you release MPIInstance.
617 */
618 - (void) MPIFinalize;
619
620 /**
621 * Abort MPI.
622 * Terminates MPI execution environment.
623 */
624 - (void) MPIAbort;
625
626 /**
627 * Check MPI state.
628 * Indicates whether MPI.Init has been called.
629 * @return Returns TRUE if MPI.Init has been called
630 *         , FALSE otherwise.
631 */
632 - (BOOL) MPIInitialized;
633
634 /**
635 * MPI Time.
636 * Returns an elapsed time on the calling processor
637 * @return Time in seconds since an arbitrary time
638 *         in the past.
639 */
640 - (NSNumber *) MPIWTime;
641
642 /**
643 * Returns the resolution of MPI.Wtime.
644 * @return Time in seconds of resolution of
645 *         MPI.Wtime
646 */
647 - (NSNumber *) MPIWTick;
648
649 /**
650 * MPI Processor name.
651 * Gets the name of the processor.
652 * @return Returns the name of the processor as a
653 *         NSString.
654 */
655 - (NSString *) MPIGetProcessorName;
656
657 /**
658 * MPI Instance variables Accessors.
659 * MPIInstance variables Accessors.
660 */
661 /**
662 * @return Returns MPI World Communicator.
663 */
664 - (MPIComm *) commWorld;
665
666 /**
667 * Set MPIInstance communicator.
668 * @param An MPIComm object.
669 */
670 - (void) setCommWorld:(MPIComm *) aComm;
671
672 /**
673 * MPI Environment status.
674 * Get MPI environment status.
675 * @return Returns MPI Status.
676 */
677 - (MPI_Status) status;
678
679 /**
680 * Set MPI environment status.
681 * @param status: A MPI_Status tag.
682 */
683 - (void) setStatus:(MPI_Status) status;
684
685 /**
686 * Dictionary of MPI non blocking request. Keys are
687 *         message tags.
688 */
689 - (NSMutableDictionary *) requestDictionary;
690
691 /**
692 * A fix for Cocoa MacMPI applications.
693 * Because MacMPI expects to read the nodelist_ip
694 * file using fopen, and this
695 * file is generally placed in the same directory
696 * where the Cocoa bundle
697 * application resides, it is necessary to set the
698 * default directory to the
699 * directory of the application very early in the
700 * code (before calling MPI.Init).
701 */
702 - (void) fixMacMPI;
703
704 @end
705
706 //
707 // MPIInstance.m
708 // MPIObjC
709 //
710 // Created by Jean-Matthieu on 14/08/2004.
711 // Copyright 2004 __MyCompanyName__. All rights
712 // reserved.
713
714 #import "MPIInstance.h"
715 #import <unistd.h>
716
717 MPIInstance *theMPIInstance = nil;
718
719 @implementation MPIInstance
720
721 - (id) init
722 {
723     self = [super init];
724
725     if (self)

```

```

724     {
725         // [self _fixMacMPI];
726         mBuffer = [[NSString alloc] init];
727         requestDictionary = [[NSMutableDictionary alloc] init];
728         theMPIInstance = self;
729         return self;
730     }
731
732     return nil;
733 }
734
735 -(void) dealloc
736 {
737     MPI_Finalize();
738     [MPIComm release];
739     [requestDictionary release];
740     [mBuffer release];
741     [super dealloc];
742 }
743
744 #pragma mark -
745 #pragma mark Class factory methods
746
747 +(id) mpiWith:(int*) argc :(char ***)argv
748 {
749     MPIInstance *mpi = [[MPIInstance alloc] init];
750
751     // Initialize MPI
752     MPI_Init(argc, argv);
753     MPIComm *worldComm = [[MPIComm alloc]
754         initWithCommunicator:MPLCOMMLWORLD];
755     [mpi setCommWorld:worldComm];
756     [worldComm release];
757
758     return mpi;
759 }
760
761 +(MPIInstance *) getInstance
762 {
763     // TODO: Mutex Begin
764     if (theMPIInstance == nil) {
765         int argc = 0;
766         char **argv = NULL;
767         theMPIInstance = [MPIInstance mpiWith:&argc&22
768             :&argv];
769         return theMPIInstance;
770     }
771     // TODO: Mutex End
772     return theMPIInstance;
773 }
774
775 #pragma mark -
776 #pragma mark MPI Functions
777
778 -(void) MPIFinalize
779 {
780     MPI_Finalize();
781 }
782
783 -(void) MPIAbort
784 {
785     MPI_Abort(MPLCOMMLWORLD, 1);
786 }
787
788 -(BOOL) MPIInitialized
789 {
790     int *flag = 0;
791     MPI_Initialized(flag);
792     if (flag)
793         return TRUE;
794     else
795         return FALSE;
796
797     return FALSE;
798 }
799
800 -(NSNumber *) MPIWTime
801 {
802     double wtime = MPI_Wtime();
803     NSNumber *time = [[[NSNumber alloc]
804         initWithDouble:wtime] autorelease];
805     return time;
806 }
807
808 -(NSNumber *) MPIWTick
809 {
810     double wtick = MPI_Wtick();
811     NSNumber *tick = [[[NSNumber alloc]
812         initWithDouble:wtick] autorelease];
813     return tick;
814 }
815
816 -(NSString *) MPIGetProcessorName
817 {
818     int namelen;
819     char processor_name[MPL_MAX_PROCESSOR_NAME];
820     MPI_Get_processor_name(processor_name, &namelen);
821
822     NSString *name = [[[NSString alloc]
823         initWithCString:processor_name length:
824         namelen] autorelease];
825     return name;
826 }
827
828 #pragma mark -
829 #pragma mark Accessors
830
831 -(MPIComm *) commWorld
832 {
833     return mCommWorld;
834 }

```

```

834
835 - (void) setCommWorld:(MPIComm *) aComm
836 {
837     if (mCommWorld)
838         [mCommWorld release];
839     mCommWorld = [aComm retain];
840 }
841
842 - (MPI_Status) status
843 {
844     return mMPIStatus;
845 }
846
847 - (void) setStatus:(MPI_Status) status
848 {
849     mMPIStatus = status;
850 }
851
852 - (NSMutableDictionary *) requestDictionary
853 {
854     return requestDictionary;
855 }
856
857 #pragma mark -
858 #pragma mark Utility functions
859 - (void) fixMacMPI
860 {
861     NSString *path = [[NSBundle mainBundle]
862         bundlePath];
863     char cpath[1024];
864     [path getCString:cpath];
865
866     {
867         char *lastSlash, *tp;
868         for (lastSlash=tp=cpath; tp=strchr(tp, '/')
869             ; lastSlash=tp++) ;
870         lastSlash[1]=0; // specifies the parent of
871             the bundle directory
872     }
873     chdir(cpath);
874 }
875
876 @end
877 //
878 // MPIMessage.h
879 // MPIObjC
880 //
881 // Created by Jean-Matthieu on 16/08/2004.
882 // Copyright 2004 __MyCompanyName__. All rights
883 // reserved.
884
885 #import <Foundation/Foundation.h>
886 #import "mpi.h"
887
888 @interface MPIMessage : NSObject {
889     id buffer;
890     int length;
891     MPI_Datatype datatype;
892 }
893
894 - (id) initWithBuffer:(id)buffer andDatatype:(
895     MPI_Datatype) dType;
896
897 - (id) buffer;
898 - (void) setBuffer:(id)aBuffer;
899 - (int) length;
900 - (MPI_Datatype) datatype;
901
902 @end
903 //
904 // MPIMessage.m
905 //
906 // Created by Jean-Matthieu on 16/08/2004.
907 // Copyright 2004 __MyCompanyName__. All rights
908 // reserved.
909
910 #import "MPIMessage.h"
911
912 @implementation MPIMessage
913
914 - (id) initWithBuffer:(id)aBuffer andDatatype:(
915     MPI_Datatype) dType
916 {
917     self = [super init];
918     if (self)
919     {
920         [self setBuffer:buffer];
921         datatype = aDatatype;
922         return self;
923     }
924     return nil;
925 }
926
927 - (void) dealloc
928 {
929     [buffer release];
930     [super dealloc];
931 }
932
933 - (id) buffer
934 {
935     return buffer;
936 }
937
938 - (void) setBuffer:(id)aBuffer
939 {
940     if (buffer)
941         [buffer release];
942     buffer = [aBuffer retain];
943
944     if ([buffer isKindOfClass:[NSString class]]){
945         length = [buffer length];
946     }
947 }

```

```

945     } else if ([buffer isKindOfClass:[NSNumber
946         class]]){
947         length = [[buffer stringValue] length];
948     }
949 }
950 - (int) length{
951     return length;
952 }
953 - (MPI_Datatype) datatype
954 {
955     return datatype;
956 }
957 @end
958 //
959 // MPIRequest.h
960 // MPIObjC
961 //
962 // Created by Jean-Matthieu on 16/08/2004.
963 // Copyright 2004 __MyCompanyName__. All rights
964 // reserved.
965 //
966 #import <Foundation/Foundation.h>
967 #import "mpi.h"
968
969 /**
970 * MPIRequest - MPI Request object.
971 */
972 @interface MPIRequest : NSObject {
973     MPI_Request request;
974     MPI_Status status;
975 }
976 /**
977 * MPIRequest Constructor.
978 * Initiate a MPIRequest object with request
979 * aRequest.
980 * @param aRequest: A MPI_Request handle.
981 * @return Returns an instantiated MPIRequest object.
982 */
983 - (id) initWithRequest:(MPI_Request)aRequest;
984
985 /**
986 * Completes a non-blocking operation.
987 * MPIWait waits for a MPI send or receive to
988 * complete.
989 */
990 - (void) MPIWait;
991
992 /**
993 * Tests a non-blocking operation.
994 * Tests for the completion of a send or receive.
995 * @return True if operation completed. Sets the
996 * status member of the MPIRequest object.
997 */
998 - (int) MPITest;
999
1000 /**
1001 * Frees a request.
1002 * Frees a communication request object.
1003 */
1004 - (void) MPIRequestFree;
1005
1006 /**
1007 * MPIWaitAll:(int)count;
1008
1009 * MPIWaitAny:(int)count;
1010 */
1011 - (MPI_Status) status;
1012
1013 - (void) setStatus:(MPI_Status) aStatus;
1014
1015 @end
1016 //
1017 // MPIRequest.m
1018 // MPIObjC
1019 //
1020 // Created by Jean-Matthieu on 16/08/2004.
1021 // Copyright 2004 __MyCompanyName__. All rights
1022 // reserved.
1023
1024 #import "MPIRequest.h"
1025
1026 @implementation MPIRequest
1027 - (id) initWithRequest:(MPI_Request)aRequest
1028 {
1029     self = [super init];
1030     if (self)
1031     {
1032         request = aRequest;
1033         return self;
1034     }
1035     return nil;
1036 }
1037
1038 - (void) dealloc
1039 {
1040     [super dealloc];
1041 }
1042
1043
1044 - (void) MPIWait
1045 {
1046     MPI_Wait(&request, &status);
1047 }
1048
1049
1050 - (int) MPITest
1051 {
1052     int flag = 0;
1053     MPI_Test(&request, &flag, &status);
1054     return flag;
1055 }

```

```
1056
1057 - (void) MPIRequestFree
1058 {
1059     MPI_Request_free(&request);
1060 }
1061 /*
1062 - (void) MPIWaitAll:(int)count
1063 {
1064     MPI_Waitall(count, &request, &status);
1065 }
1066
1067 - (int) MPIWaitAny:(int)count
1068 {
1069     int index = 0;
1070     MPI_Waitany(count, &request, &index, &status);
1071     return index;
1072 }
1073 */
1074
1075 - (MPI_Status) status
1076 {
1077     return status;
1078 }
1079
1080 - (void) setStatus:(MPI_Status) aStatus
1081 {
1082     status = aStatus;
1083 }
1084 @end
1085 /*
1086 * MpiObjC.h
1087 * MPIObjC
1088 *
1089 * Created by Jean-Matthieu on 15/08/2004.
1090 * Copyright 2004 __MyCompanyName__. All rights
1091 * reserved.
1092 */
1093
1094 #import "mpi.h"
1095 #import "MPIInstance.h"
1096 #import "MPIComm.h"
```