Distributed Applications Design on Mac OS X

Jean-Matthieu Schaffhauser
MSc. Computer Science
Student #03053404
Oxford Brookes University

09 th 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.*

19th 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 efficientely 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 retreived 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 tasks requires a lot of effort and should be greatly guided by different implementations already existing in C++.

Resources

Books, CPUs and cafeine ...

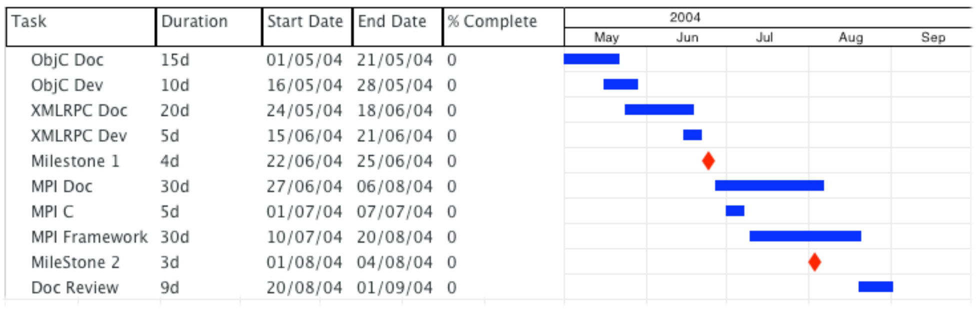
Books

- 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



Task	Duration	Start Date	End Date	% Complete
ObjC Doc	15d	01/05/04	21/05/04	0
ObjC Dev	10d	16/05/04	28/05/04	0
XMLRPC Doc	20d	24/05/04	18/06/04	0
XMLRPC Dev	5d	15/06/04	21/06/04	0
Milestone 1	4d	22/06/04	25/06/04	0
MPI Doc	30d	27/06/04	06/08/04	0
MPI C	5d	01/07/04	07/07/04	0
MPI Framework	30d	10/07/04	20/08/04	0
MileStone 2	3d	01/08/04	04/08/04	0
Doc Review	9d	20/08/04	01/09/04	0

Figure 1: Gantt chart

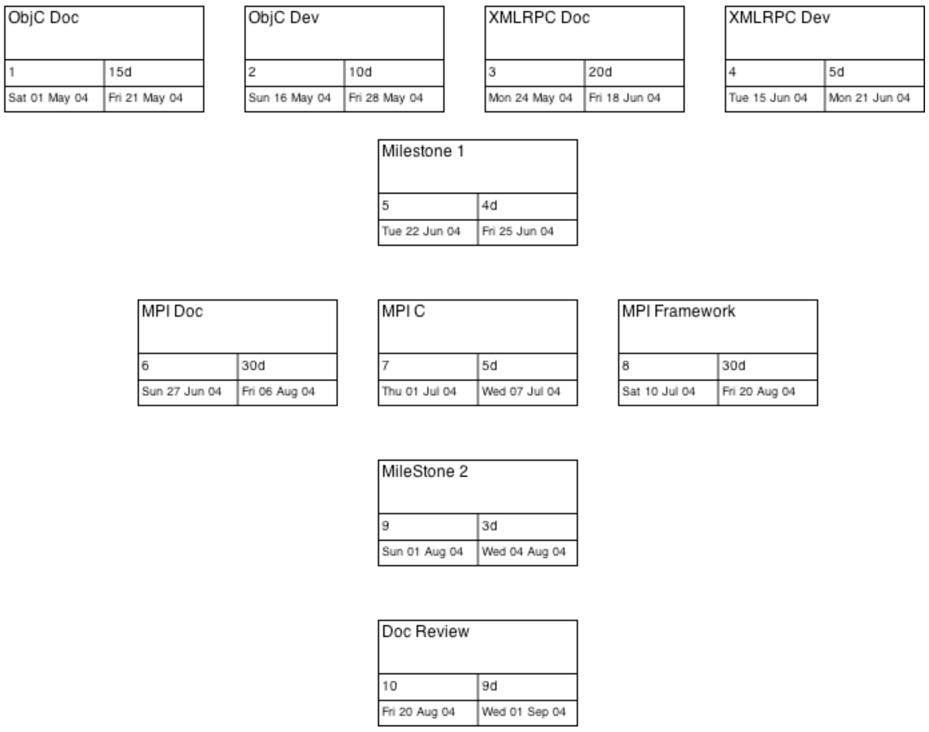


Figure 2: PERT chart

Table of Contents

List of Tables

List of Figures

List of Programs

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 transfered to a computer for processing on the 4th 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 independentely, 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[1] 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.

[1]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

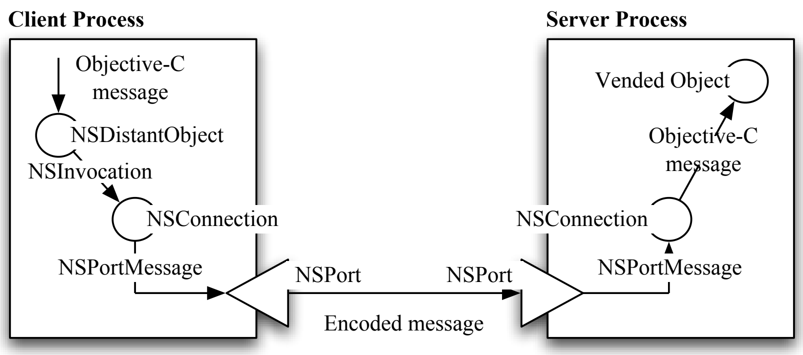


Figure 1.1: Sending a Message to a Vended Object. *Source: Apple Documentation - Distributed Objects*

.

object a server wants to vend, it will create an `NSConnection`[2] 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 refered 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 transparentely 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.

[2]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 AppController : NSObject
 NSTableView *tableView
@end

@interface AppController
-(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 implements 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
`oneway`	Used for sending **asynchronous** messages, when one doesn't need to wait for a reply.
`in`	Information is being passed in a message.
`out`	Indicates that an argument is being used to return information by reference.
`inout`	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 `const`, for which `in` is the default.
`bycopy`	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.)
`byref`	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 explicitely on a server before an object we shall vend is attached to it. On the client side, it is used explicitely 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 directely 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`

```
#import "MyVendedObject.h"
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[]) {
        NSSocketPort *receivePort;
    NSConnection *connection;

    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSRunLoop *runloop = [NSRunLoop currentRunLoop];
    MyVendedObject *vendedObject = [[MyVendedObject alloc] init];

    NS_DURING
        // This server will wait for requests on port 4242
        receivePort = [[NSSocketPort alloc] initWithTCPPort:4242];
    NS_HANDLER
        NSLog(@"Unable to get port 4242");
        exit(-1);
    NS_ENDHANDLER

    // Create the NSConnection object
    connection = [NSConnection connectionWithReceivePort:receivePort
                                                  sendPort:nil];

    // The port is retained by the connection
    [receivePort release];

    // When clients use this connection, they will
    // talk to the vendedObject
    [connection setRootObject:vendedObject];

    // The chatter server is retained by the connection
    [vendedObject release];



    // Start the runloop
    [runloop run];

    // If the run loop exits (and I do not know why it would), cleanup
    [connection release];
    [pool release];
    return 0;
}
```

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 `NSConnectionDidDieNotificatio` 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

```
#import "MyVendedObjectProtocol.h"
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[]) {
    NSSocketPort *sendPort;
    NSConnection *connection;
    id proxy;

    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    // Create the send port
    sendPort = [[NSSocketPort alloc] initRemoteWithTCPPort:8081
                                                     host:@"localhost"];

    // Create an NSConnection
    connection = [NSConnection connectionWithReceivePort:nil
                                                 sendPort:sendPort];

    // Set timeouts to something reasonable
    [connection setRequestTimeout:10.0];
    [connection setReplyTimeout:10.0];

    // The send port is retained by the connection
    [sendPort release];

    // Get the proxy
    proxy = [[connection rootProxy] retain];

    // By telling the proxy about the protocol for the object
    // it represents, we significantly reduce the network
    // traffic involved in each invocation
    [proxy setProtocolForProxy:@protocol(MyVendedObjectProtocol)];

    // The rest of your program code goes here

    // If the run loop exits (and I do not know why it would), cleanup
    [connection release];
    [pool release];
    return 0;
}
```

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[3].

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

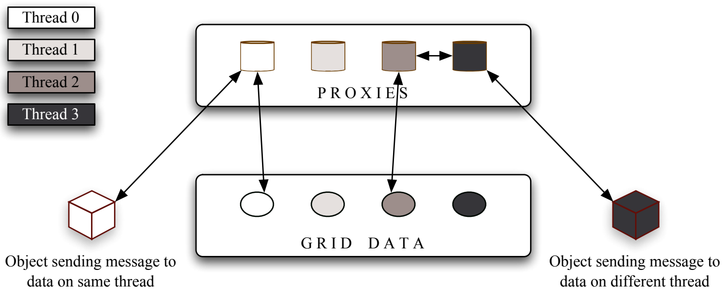

Thread 0
Thread 1
Thread 2
Thread 3
PROXIES
GRID DATA
Object sending message to data on same thread
Object sending message to data on different thread

Figure 1.2: Comminucation between proxies and objects. *Source: Russell Standish - Dsitributed 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

[3]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

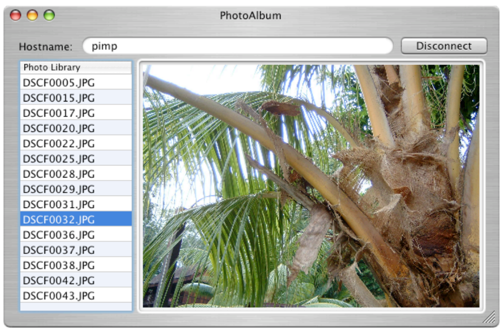
To illustrate Cocoa distributed objects system, I coded a photo album application called *PhotoAlbum*. PhotoAlbum is composed of a server (Program 1.3.1) `photoalbumd` that exports pictures inside a user folder for clients (Program 1.3.2) to browse on a remote computer. This XCode project is available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

Program 1.3.1: PhotoAlbum Daemon

```
/*
 *  PhotoAlbumProtocol.h
 *  Protocol definition for exchanging photos
 *  Project: PhotoAlbum
 *
 *  Created by Jean-Matthieu.
 *
 */

#import <Foundation/Foundation.h>

// Objective-C messages the client will send to the server

@protocol PhotoAlbumServer
// Get the pictures list
```

```

- (bycopy NSArray *) getPicturesList;


//
 Retrieve a picture from the server
- (bycopy NSData*) getPicture:(in bycopy NSString
      *)pictureName;

@end


//
//  PhotoAlbumServer.h
//  PhotoAlbum
//
//  Created by Jean-Matthieu.
//

#import "PhotoAlbumProtocol.h"
#import <Foundation/Foundation.h>


@interface PhotoAlbumServer : NSObject <
      PhotoAlbumServer> {
    NSMutableArray *photos;
}

- (void)setup;

@end


//
//  PhotoAlbumServer.m
//  PhotoAlbum
//
//  Created by Jean-Matthieu.
//

#import "PhotoAlbumServer.h"

static NSString *photoAlbumPath = nil;

@implementation PhotoAlbumServer

- (id)init
{
    self = [super init];
    [self setup];
    return self;
}

- (void)dealloc
{
    [photos release];
    [super dealloc];
}

- (void)setup
{
    if (photos)
        [photos release];
    photoAlbumPath = [NSString stringWithFormat:@"%
        @/photos", NSHomeDirectory()];
    photos = [[NSMutableArray alloc] initWithArray
        :[[NSFileManager defaultManager]
        directoryContentsAtPath:photoAlbumPath]];
}

#pragma mark Protocol Implementation
- (bycopy NSArray *) getPicturesList
{
    return photos;
}

- (bycopy NSData *) getPicture:(in bycopy NSString
      *)pictureName
{
    NSImage *anImage = [[[NSImage alloc]
        initWithContentsOfFile:[photoAlbumPath
        stringByAppendingPathComponent:pictureName
        ]] autorelease];
    return [anImage TIFFRepresentation];
}


@end


/*
 *  photoalbumd.m
 *  PhotoAlbum
 *
 *  Created by Jean-Matthieu.
 *
 */

#import "PhotoAlbumServer.h"
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[]) {
    NSSocketPort *receivePort;
    NSAutoreleasePool * pool = [[NSAutoreleasePool
        alloc] init];
    NSRunLoop *runloop = [NSRunLoop currentRunLoop
        ];
    PhotoAlbumServer *photoAlbumD = [[
        PhotoAlbumServer alloc] init];

    NS_DURING
        // This server will wait for requests
        // on port 4242
        receivePort = [[NSSocketPort alloc]
            initWithTCPPort:4242];
    NS_HANDLER
```

```
        NSLog(@"Unable to get port 4242");
        exit(1);
    NS_ENDHANDLER

    NSConnection *connection = [NSConnection
        connectionWithReceivePort:receivePort


    // The port is retained by the connection
    [receivePort release];


    // Set the responding server object as the root
        object for this connection.
    [connection setRootObject:photoAlbumD];

    // The photo album is retained by the
        connection
    [photoAlbumD release];

        sendPort
    // Start the runloop
    [runloop run];
          ];
    // If the run loop exits (and I do not know why
        it would), cleanup
    [connection release];
    [pool release];
    return 0;
}
```



Hostname: pimp

DSCF0005.JPG
DSCF0015.JPG
DSCF0017.JPG
DSCF0020.JPG
DSCF0022.JPG
DSCF0025.JPG
DSCF0028.JPG
DSCF0029.JPG
DSCF0031.JPG
DSCF0032.JPG
DSCF0036.JPG
DSCF0037.JPG
DSCF0038.JPG
DSCF0042.JPG
DSCF0043.JPG

Figure 1.3: PhotoAlbum Client Application Screenshot

Program 1.3.2: PhotoAlbum Client

```
/*
 *  PhotoAlbumController.h
 *  PhotoAlbum
 *
 *  Created by Jean-Matthieu.
 *
*/

#import <Cocoa/Cocoa.h>

@interface PhotoAlbumController : NSObject
{
    IBOutlet NSButton *connectButton;
    IBOutlet NSTextField *hostnameField;
    IBOutlet NSTableView *photoTable;
    IBOutlet NSImageView *photoViewer;
    id proxy;
    NSArray *myPhotos;
}
- (IBAction)connect:(id)sender;

- (void)doConnect;
- (void)doDisconnect;
@end


/*
 *  PhotoAlbumController.m
 *  PhotoAlbum
 *
 *  Created by Jean-Matthieu.
 *
*/

```

```
#import "PhotoAlbumController.h"


#
 import "PhotoAlbumProtocol.h"

@implementation PhotoAlbumController

-(void) dealloc
{
    if(proxy)
        [self doDisconnect];
    [super dealloc];
}

- (void) awakeFromNib
{
    proxy = nil;
    myPhotos = nil;
}


- (void)doConnect
{
   NSConnection *connection;
   NSSocketPort *sendPort;

   // Create the send port
   sendPort = [[NSSocketPort alloc] initRemoteWithTCPPort:4242
                                    host:[hostnameField stringValue]];

   // Create an NSConnection
   connection = [NSConnection connectionWithReceivePort:nil
                                          sendPort:sendPort];

   // Set timeouts to something reasonable
   [connection setRequestTimeout:10.0];
   [connection setReplyTimeout:10.0];
   // The send port is retained by the connection
   [sendPort release];


   NS_DURING
      // Get the proxy
      proxy = [[connection rootProxy] retain];

      // By telling the proxy about the protocol for the object
      // it represents, we significantly reduce the network
      // traffic involved in each invocation
      [proxy setProtocolForProxy:@protocol(PhotoAlbumServer)];

   NS_HANDLER
      // If the server does not respond in 10 seconds,
      // this handler will get called
      [self doDisconnect];
   NS_ENDHANDLER

}

- (void)doDisconnect
{
    NSConnection *connection = [proxy connectionForProxy];
    [connection invalidate];
    [proxy release];
    proxy = nil;
    [myPhotos release];
    myPhotos = nil;
    [connectButton setTitle:@"Connect"];
}

- (IBAction)connect:(id)sender
{
    if (!proxy){
        [self doConnect];
        if (!proxy)
            return;

        myPhotos = [[NSArray alloc] initWithArray:[proxy getPicturesList]];
        [connectButton setTitle:@"Disconnect"];
        [photoTable reloadData];
    } else {
        [self doDisconnect];
        [photoTable reloadData];
    }
}

#pragma mark TableView delegate and datasource
- (int)numberOfRowsInTableView:(NSTableView *)aTableView
{
    if (nil != myPhotos){
        return [myPhotos count];
    } else
        return 0;
}


-(id)tableView:(NSTableView *)aTableView objectValueForTableColumn:(NSTableColumn *)aTableColumn row:(int)rowIndex
{
    if (nil != myPhotos){
        return [myPhotos objectAtIndex:rowIndex];
    } else
}
        return nil;


- (void)tableViewSelectionDidChange:(NSNotification *)notification{
    if ([photoTable selectedRow] != -1){
        NSData *theData = [proxy getPicture:[myPhotos objectAtIndex:[photoTable selectedRow]]];
        NSImage *anImg = [[[NSImage alloc] initWithData:theData] autorelease];
        [photoViewer setImage:anImg];
    } else {
        [photoViewer setImage:nil];
    }
}
@end


//
//  main.m
//  PhotoAlbum
//
//  Created by Jean-Matthieu on Tue Jun 15 2004.
//  Copyright (c) 2004 __MyCompanyName__. All rights reserved.
//

#import <Cocoa/Cocoa.h>

int main(int argc, char *argv[])
{
    return NSApplicationMain(argc, argv);
}
```

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.

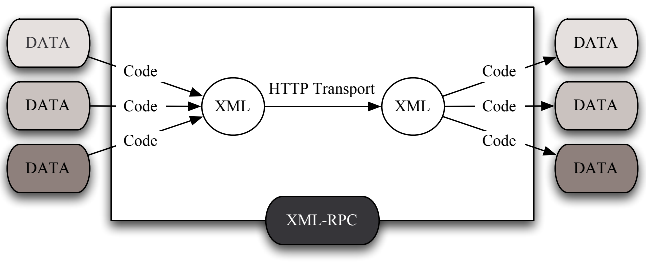


Figure 2.1: XML-RPC Transport. *Source: JY Stervinou*

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`[1] 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 formating.
- `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 41st state. I save you the response's header; it contains the `POST` request's return code (`200 OK`) and data similiar to the post request's header.

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

[1]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 occured 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 inherinting 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

```
/*
 *  xmlrpcserver.m
 *  A simple XML-RPC server
 *  Compile with: gcc xmlrpcserver.m -framework Foundation -framework
     XMLRPCObjC -o xmlrpcserver
 *  Run with ./xmlrpcserver
 *
 *  Created by Jean-Matthieu.
 *
*/

#import <Foundation/Foundation.h>
#import <XMLRPCObjC/XMLRPCObjC.h>

@interface telephoneDirectory : NSObject
- (NSDictionary *)cardForUser:(NSString *)aUser;
@end

@implementation telephoneDirectory
- (NSDictionary *)cardForUser:(NSString *)aUser;
{
        /* Telephone Directory
         * A dictionary where each entry is a username
         * representing user's personal information stored as a
             dictionary
         */
        NSDictionary *telephoneDict = [NSDictionary
           dictionaryWithContentsOfFile:@"/Users/jms/telephone.plist
```

```
            "];
        NSDictionary *result;
        if (nil != (result = [telephoneDict objectForKey:aUser])){
                return result;
        } else {
                return nil;
        }
}
@end

int main (int argc, const char *argv[]) {
        NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
        XMLRPCServer *server = [XMLRPCServer server];
        telephoneDirectory *td = [[telephoneDirectory alloc] init];

        [server setObject:td forKey:@"telephoneDirectory"];

        // The telephone directory is reatined by the server
        [td release];
        /*
         * run the server (never exits)
         */
        [server run];
        [pool release];

        exit(0);
}
```

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

```
<plist version="1.0">
<dict>
        <key>JMS</key>
        <dict>
                <key>Name</key>
                <string>Jean-Matthieu Schaffhauser</string>
                <key>School</key>
                <string>Oxford Brookes University</string>
                <key>Telephone</key>
                <string>+44 1865 765 535</string>
                <key>address</key>
                <string>27 York Avenue Headington OX38NS UK</string>
                <key>email</key>
                <string>jean-matthieu@users.sourceforge.net</string>
                <key>website</key>
                <string>http://cocoanut.net</string>
        </dict>
</dict>
</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

```
/*
 *  xmlrpcclient.m
 *  A simple XML-RPC Client
 *  Compile with: gcc xmlrpcclient.m -framework Foundation -framework
    XMLRPCObjC -o xmlrpcclient
 *  Run with ./xmlrpcclient
 *
 *  Created by Jean-Matthieu.
 *
*/

#import <Foundation/Foundation.h>
#import <XMLRPCObjC/XMLRPCObjC.h>

int main (int argc, const char *argv[]) {
        NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
        XMLRPCClient *client;
        id object;
        NSString *username = [NSString stringWithString:@"JMS"];
        NSArray *args = [NSArray arrayWithObject:username];

        client = [XMLRPCClient client:[NSURL URLWithString:@"http://
            localhost:8000/RPC2"]];
        object = [client invoke:@"telephoneDirectory.cardForUser"
            withArguments:args];
        NSLog(@"%@", [object description]);

        [pool release];
        exit(0);
}
```

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

```
/*
 *  xmlrpcproxy.m
 *  A simple XML-RPC Client
 *  Compile with: gcc xmlrpcproxy.m -framework Foundation -framework
    XMLRPCObjC -o xmlrpcproxy
 *  Run with ./xmlrpcclient
 *
 *  Created by Jean-Matthieu.
 *
*/

#import <Foundation/Foundation.h>
#import <XMLRPCObjC/XMLRPCObjC.h>

@protocol Bell
- (NSDictionary *)cardForUser:(NSString *)aUser;
@end

int main (int argc, const char *argv[]) {
        NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
        XMLRPCClient *client;
        XMLRPCProxy <Bell> *telDirectory;

        NSDictionary *result;

        client = [XMLRPCClient client:[NSURL URLWithString:@"http://
            localhost:8000/RPC2"]];
        telDirectory = (id <Bell>)[client proxyForTarget:@"
            telephoneDirectory"];
        [telDirectory setProtocolForProxy:@protocol(Bell)];

        result = [NSDictionary dictionaryWithDictionary:[telDirectory
            cardForUser:@"JMS"]];

        NSLog(@"%@", [result description]);

        [pool release];
        exit(0);
}
```

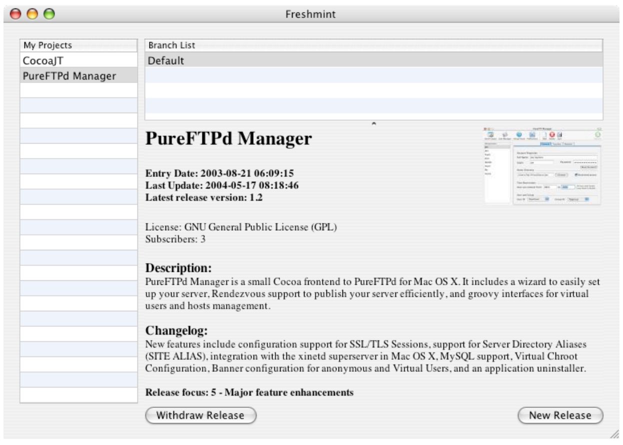
2.4 Freshmint: A Client to Freshmeat.net XML-RPC Interface

Freshmeat.net, a popular software publication website, offers to suscribers 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/`).



PureFTPd Manager

Entry Date: 2003-08-21 06:09:15
Last Update: 2004-05-17 08:18:46
Latest release version: 1.2

License: GNU General Public License (GPL)
Subscribers: 3

Description:
PureFTPd Manager is a small Cocoa frontend to PureFTPd for Mac OS X. It includes a wizard to easily set up your server, Rendezvous support to publish your server efficiently, and groovy interfaces for virtual users and hosts management.

Changelog:
New features include configuration support for SSL/TLS Sessions, support for Server Directory Aliases (SITE ALIAS), integration with the xinetd superserver in Mac OS X, MySQL support, Virtual Chroot Configuration, Banner configuration for anonymous and Virtual Users, and an application uninstaller.

Release focus: 5 - Major feature enhancements

Figure 2.2: Freshmint Screenshot

Program 2.4.1: Freshmint Implementation of Freshmeat.net API

```
//
//  Freshmeat.h
//  Freshmint
//
//  Created by Jean-Matthieu
//

#import <Foundation/Foundation.h>
#include <XMLRPCObjC/XMLRPCObjC.h>

@interface Freshmeat : NSObject
{
    XMLRPCClient *client;
    NSMutableDictionary *sessionDictionary;

    BOOL isConnected;
}

/* [ sessionDictionary method ]
    * Parameters:
    * None
    *
    * Returns:
    * NSDictionary with session informations
    *
    * Description:
    * Returns a dictionary containing SID, API
        Version, Lifetime, logintime
*/
- (NSDictionary *)sessionDictionary;



/* [ autoLogout method ]
```

```
    *
    * Parameters:
    * None


*
 Returns:
    * None
    *
    * Description:
    * Automatically logs out from Freshmeat.net
*/
- (void)autoLogout;



/* [ isConnected method ]
    * Parameters:
    * None
    *
    * Returns:
    * None
    *
    * Description:
    * Informs whether a session is active or not.
    */
- (BOOL)isConnected;



/* [ fetch_available_licenses method ]
    * Parameters:
    * None
    *
    * Returns:
    * Array of available licenses
*/
- (NSArray *)fetch_available_licenses;

 /* [ fetch_available_release_foci method ]
    * Parameters:
    * None
    *
    * Returns:
    * Struct of available release focus types and associated ID
*/
- (NSDictionary *)fetch_available_release_foci;

/* [ fetch_project_list method ]
 * Parameters (passed in struct form):
 * SID                       - Session ID to work with

 * Returns:
 * Struct consisting of "projectname_full",
 * "projectname_short", "project_status", and "project_version"
*/

- (NSDictionary *)fetch_project_list;


/* [ fetch_branch_list method ]
    * Parameters (passed in struct form):
    * SID                       - Session ID to work with
    * project_name              - Project name to fetch branches for
    *
    * Returns:
    * Array of branch name strings.
*/
- (NSArray *)fetch_branch_list_for_project:(NSString *)project_name;


/* [ fetch_release method ]
   * Parameters (passed in struct form):
   * SID                       - Session ID
   * project_name              - Project name
   * branch_name               - Branch name
   * version                   - Release version string
   *
   * Returns:
   * Struct consisting of "version", "changes", "release_focus", and "hide_from_frontpage"
*/
- (NSDictionary *)fetch_release_for_project:(NSString *)project_name branch:(NSString *)branch_name version:(NSString *)version;


/* [ login method ]
   * Parameters (passed in struct form):
   * username                  - Regular freshmeat username
   * password                  - Regular freshmeat password
   *
   * Returns:
   * Struct of SID, lifetime, and API Version
   * SID: Session ID to be used in subsequent requests to the XML-RPC service
   * Lifetime: Lifetime of the session ID in seconds
   * API Version: API Version currently in use
*/
- (void)login:(NSString *)username password:(NSString *)password;


/* [ logout method ]
   * Parameters (passed in struct form):
```

```
   * SID                        - Session ID to terminate
   *
   * Returns:
   * Struct of "OK" => "Logout successful." if logout was successful
*/
- (void)logout:(NSString *)SID;


/* [ publish_release method ]
   * Parameters (passed in struct form):
   * SID                        - Session ID to work with
   * project_name               - Project name to submit a release for
   * branch_name                - Branch name to submit a release for
   * version                    - Version string of new release
   * changes                    - Changes list, no HTML, character limit 600 chars
   * release_focus              - Release focus ID of new release (see Appendix A)
   * hide_from_frontpage        - Set to 'Y' if release is to be hidden from
   * frontpage, everything else does not hide it
   * license                    - Branch license
   * url_homepage               - Homepage
   * url_tgz                    - Tar/GZ
   * url_bz2                    - Tar/BZ2
   * url_zip                    - Zip
   * url_changelog              - Changelog
   * url_rpm                    - RPM package
   * url_deb                    - Debian package
   * url_osx                    - OS X package
   * url_bsdport                - BSD Ports URL
   * url_purchase               - Purchase
   * url_cvs                    - CVS tree (cvsweb)
   * url_list                   - Mailing list archive
   * url_mirror                 - Mirror site
   * url_demo                   - Demo site
   *
   * Returns:
   * Struct of "OK" => "submission successful"
   *
   * Notes:
   * 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'.
   *
   * For convinience, we pass a dictionary to this method
*/
- (void)publish_release:(NSDictionary *)newReleaseInfo;


/* [ withdraw_release method ]
   * Parameters (passed in struct form):
   * SID                        - Session ID
   * project_name               - Project name
   * branch_name                - Branch name
   * version                    - Release version string
   *
   * Returns:
   * Struct of "OK" => "Withdraw successful.".
*/
- (void)withdraw_release_for_project:(NSString *)project_name branch:(NSString *)branch_name version:(NSString *)version;

@end


//
//  Freshmeat.m
//  Freshmint
//
//  Created by Jean-Matthieu
//

#import "Freshmeat.h"


@implementation Freshmeat

- (id)init
{
    self = [super init];
    if (self) {
        client = [[XMLRPCClient client:[NSURL URLWithString:@"http://freshmeat.net/xmlrpc"]] retain];
        sessionDictionary = [[NSMutableDictionary alloc] init];
        isConnected = NO;
    }
    return self;
}

-(void) dealloc
{
    [sessionDictionary release];
    [client release];
    [super dealloc];
}

- (NSDictionary *)sessionDictionary
{
    return sessionDictionary;
}
```

```
- (void)autoLogout



{

    [self logout:[sessionDictionary objectForKey:@"
        SID"]];
}

- (BOOL)isConnected
{
    return isConnected;
}

// Freshmeat methods invocation
- (NSArray *)fetch_available_licenses
{
    NSArray *object;
    object = [client invoke:@"
        fetch_available_licenses" withArguments:[
        NSArray arrayWithObject:@""]];
    return object;
}

- (NSDictionary *)fetch_available_release_foci
{

    NSDictionary *object;
    object = [client invoke:@"
        fetch_available_release_foci"
        withArguments:[NSArray arrayWithObject:@
        ""]];
    NSLog([object description]);
    return object;
}

-(NSDictionary *)fetch_project_list
{
    id object;
    NSDictionary *myStruct = [NSDictionary
        dictionaryWithObjects:[NSArray
        arrayWithObjects:[sessionDictionary
        objectForKey:@"SID"], nil] forKeys:[
        NSArray arrayWithObjects:@"SID", nil]];
    NSArray *args = [NSArray arrayWithObject:
        myStruct];
    object = [client invoke:@"fetch_project_list"
        withArguments:args];

    // Order projects
    NSEnumerator *objEnumerator = [object
        objectEnumerator];
    id entry;

    NSMutableDictionary *projectDictionary = [[[
        NSMutableDictionary alloc] init]
        autorelease];

    while(entry = [objEnumerator nextObject]){
        NSMutableDictionary *projectDetails;

        if (nil == (projectDetails = [
            projectDictionary objectForKey:[entry
            objectForKey:@"projectname_full"]])){
            projectDetails = [[[NSMutableDictionary
                alloc] init] autorelease];
            NSArray *branches = [NSArray
                arrayWithArray:[self
                fetch_branch_list_for_project:[
                entry objectForKey:@"
                projectname_short"]]];
            [projectDetails setObject:branches
                forKey:@"project.branches"];
        }

        NSArray *objects = [NSArray
            arrayWithObjects:[entry objectForKey:@
            "projectname_full"], [entry
            objectForKey:@"project_version"],[
            entry objectForKey:@"projectname_short
            "], nil];
        NSArray *keys = [NSArray arrayWithObjects:@
            "project.name", @"project.version", @
            "project.shortname", nil];

        NSDictionary *projectInfo = [NSDictionary
            dictionaryWithObjects:objects forKeys:
            keys];
        [projectDetails setObject:projectInfo
            forKey:@"project.info"];

        [projectDictionary setObject:projectDetails
             forKey:[entry objectForKey:@"
            projectname_full"]];
    }

    NSLog(@"%@", [projectDictionary description]);


    [sessionDictionary setObject:projectDictionary
        forKey:@"MyProjects"];

    return nil;
}

- (NSArray *)fetch_branch_list_for_project:(
    NSString *)project_name
{
    id object;
    NSDictionary *myStruct = [NSDictionary
        dictionaryWithObjects:
    [NSArray arrayWithObjects:[sessionDictionary
        objectForKey:@"SID"], project_name, nil]
        forKeys:[NSArray arrayWithObjects:@"SID", @
        "project_name", nil]];

    NSArray *args = [NSArray arrayWithObject:
        myStruct];
    object = [client invoke:@"fetch_branch_list"
        withArguments:args];
    return object;
}

- (NSDictionary *)fetch_release_for_project:(
    NSString *)project_name branch:(NSString *)
    branch_name version:(NSString *)version
{
    id object;
    NSDictionary *myStruct = [NSDictionary
        dictionaryWithObjects:[NSArray
        arrayWithObjects:[sessionDictionary
        objectForKey:@"SID"], project_name,
        branch_name,version, nil] forKeys:[NSArray
        arrayWithObjects:@"SID", @"project_name
        ", @"branch_name", @"version", nil]];
    NSArray *args = [NSArray arrayWithObject:
        myStruct];
    object = [client invoke:@"fetch_release"
        withArguments:args];
    //NSLog(@"%@", [object description]);
    return object;
}

- (void)login:(NSString *)username password:(
    NSString *)password
{
    id object = nil;
    [sessionDictionary removeAllObjects];
    NSDictionary *myStruct = [NSDictionary
        dictionaryWithObjects:[NSArray
        arrayWithObjects:username, password, nil]
        forKeys:[NSArray arrayWithObjects:@"
        username", @"password", nil]];
    NSArray *args = [NSArray arrayWithObject:
        myStruct];
    object = [client invoke:@"login" withArguments:
        args];


    [sessionDictionary addEntriesFromDictionary:
        object];
    NSCalendarDate *date = [NSCalendarDate date];
    [sessionDictionary setObject:date forKey:@"date
        "];
    isConnected = YES;
    // Autologout 5 sec before session ends
    [NSTimer scheduledTimerWithTimeInterval:[[
        sessionDictionary objectForKey:@"Lifetime
        "] intValue] - 5 target:self selector:
        @selector(autoLogout) userInfo:nil repeats
        :NO];

    [self fetch_project_list];

}

- (void)logout:(NSString *)SID
{

    NSDictionary *myStruct = [NSDictionary
        dictionaryWithObjects:[NSArray
        arrayWithObject:SID] forKeys:[NSArray
        arrayWithObject:@"SID"]];
    NSArray *args = [NSArray arrayWithObject:
        myStruct];
    [client invoke:@"logout" withArguments:args];

    [sessionDictionary removeAllObjects];
    isConnected = NO;
    NSLog(@"Freshmeat session terminated");
}

- (void)publish_release:(NSDictionary *)
    newReleaseInfo
{
    NSLog([newReleaseInfo description]);
    NSArray *args = [NSArray arrayWithObject:
        newReleaseInfo];
    [client invoke:@"publish_release" withArguments
        :args];
}

- (void)withdraw_release_for_project:(NSString *)
    project_name branch:(NSString *)branch_name
    version:(NSString *)version
{
    NSDictionary *myStruct = [NSDictionary
        dictionaryWithObjects:[NSArray
        arrayWithObjects:[sessionDictionary
        objectForKey:@"SID"], project_name,
        branch_name, version, nil] forKeys:[
        NSArray arrayWithObjects:@"SID", @"
        project_name", @"branch_name", @"version
        "]];
    NSArray *args = [NSArray arrayWithObject:
        myStruct];
    [client invoke:@"withdraw_release"
        withArguments:args];

}



@end
```

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 s 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 *MPI_SUM*, *MPI_XOR*, ...), 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 *spawn*ing 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`[1] 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

[1]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[2]) 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.

[2]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`.

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

Program 3.2.1: MPI - Hello, World

```
#include <mpi.h>

int main(int argc, char **argv)
{
        int error_code, prank, size;

        // Initialize MPI
        error_code = MPI_Init(&argc, &argv);

        // Get the rank
        MPI_Comm_rank(MPI_COMM_WORLD, &prank);
        // Get nbr of processes
        MPI_Comm_size(MPI_COMM_WORLD, &size);

        printf("Process %d of %d: Hello MPI!\n", prank, size);

        // Terminate MPI
        MPI_Finalize();

        return 0;
}

// Output
// Machine 1 : Process 0 of 3: Hello MPI!
// Machine 2 : Process 1 of 3: Hello MPI!
// 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

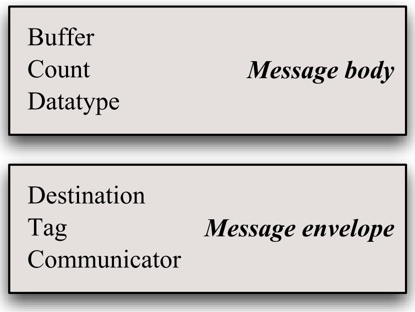


Figure 3.1: MPI_Send Arguments.

.

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

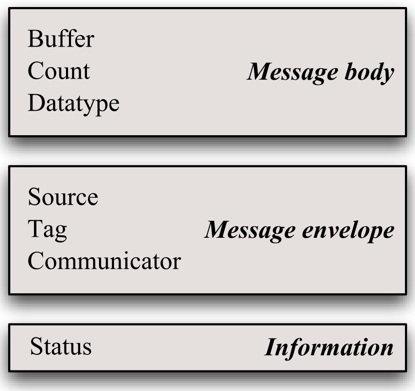


Figure 3.2: MPI_Recv Arguments.

.

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 specifiy 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`.

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

Program 3.2.2: Blocking Send and Receive

```
#include <mpi.h>
int main (int argc, char **argv)
{
  int myrank;
  MPI_Status status;
  doublea[100];

  /*Initialize MPI*/
  MPI_Init(&argc, &argv);

  /*Get rank*/
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);


  if(myrank ==0){
    /*Send a message*/
    MPI_Send(a, 100, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD);
  } else if(myrank ==1) {
    /*Receive a message*/
    MPI_Recv(a, 100, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD, &status);
  }

  MPI_Finalize(); /*Terminate MPI*/
  return 0;
}
```

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

```
#include <stdio.h>
#include <mpi.h>
int main (int argc, char **argv)
{
  int myrank;
  MPI_Request request;
  MPI_Status status;
  double a[100], b[100];

  /*Initialize MPI*/
  MPI_Init(&argc, &argv);

  /*Get rank*/
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);



  if(myrank ==0){
    /*Post a receive, send a message,then wait*/
    MPI_Irecv(b, 100, MPI_DOUBLE, 1, 19, MPI_COMM_WORLD, &request );
    MPI_Send(a, 100, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD);
    MPI_Wait(&request, &status);
  } else if(myrank ==1) {
    /*Receive a message*/
        MPI_Irecv(b, 100,MPI_DOUBLE, 0, 17, MPI_COMM_WORLD, &request )
            ;
        MPI_Send(a, 100, MPI_DOUBLE, 0, 19, MPI_COMM_WORLD);
        MPI_Wait(&request, &status);
  }

  MPI_Finalize(); /*Terminate MPI*/
  return 0;
}
```

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:

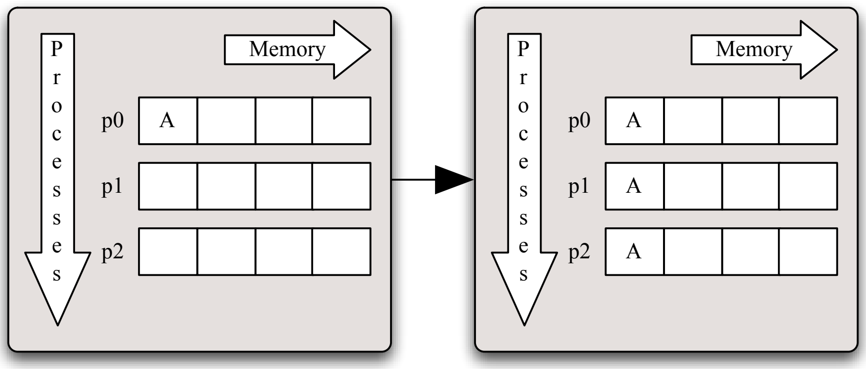


Figure 3.3: A Simple Broadcast Operation

.

```
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 an stores the result in **X** in the root process.

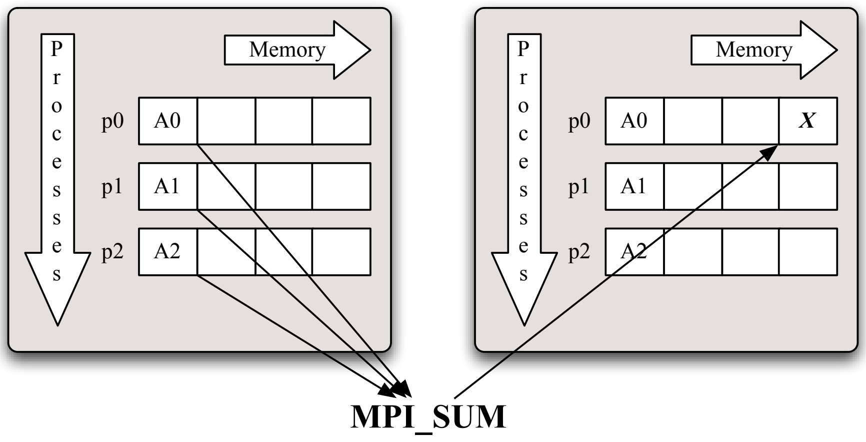


Figure 3.4: A Simple Reduce Operation

.

`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);
```

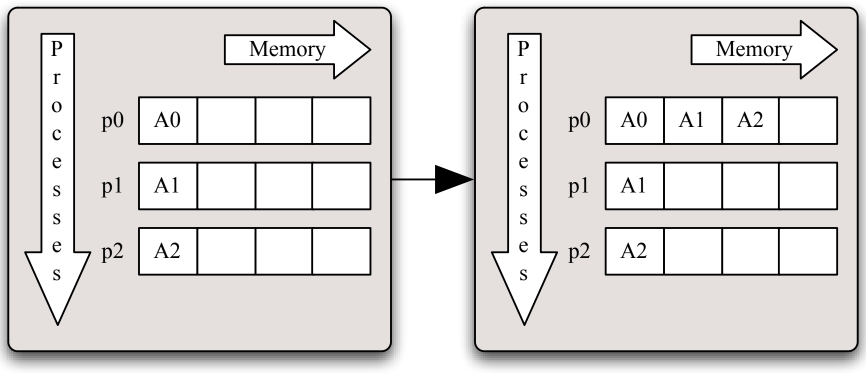


Figure 3.5: A Simple Gather Operation
.

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 Speration

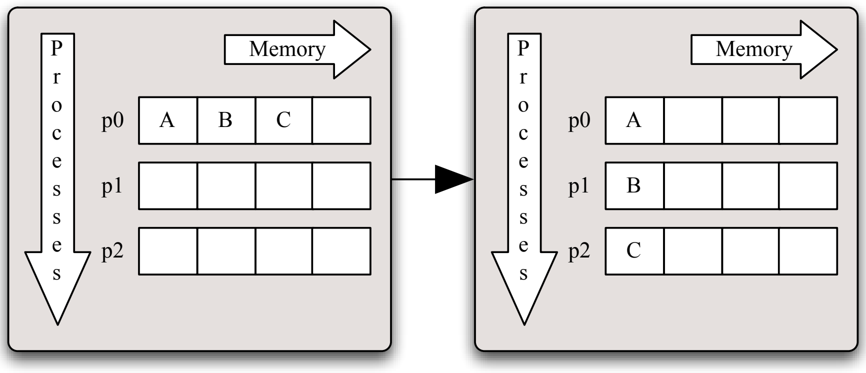


Figure 3.6: A Simple Scatter Operation
.

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 intented 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 initialized the MPI environment before calling this function.

Returns:

Returns a previously instanciated 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*

Instanciate MPI. Initialialized 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 assigment.

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) MPIIRecv: (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.

```

#import <Foundation/Foundation.h>
#import <MPIObjC/MPIObjC.h>

int main (int argc, char **argv)
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int numprocs, myrank;

    MPIInstance *mpi = [MPIInstance mpiWith:&argc :&argv];

    NSString *pname = [mpi MPIGetProcessorName];
    NSLog(pname);

    [mpi release];

    [pool release];
    return 0;

}
```

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.

```
/*
 *  MPICommTest.c
 *  MPIObjC
 *
 *  Test for the MPIComm object
 *
 */

#import <Foundation/Foundation.h>
#import <MPIObjC/MPIObjC.h>

int main (int argc, char **argv)
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    MPIInstance *mpi = [MPIInstance mpiWith:&argc :&argv];

    MPIComm *mCommWorld = [mpi commWorld];
    NSNumber *size = [mCommWorld MPICommSize];
    NSNumber *rank = [mCommWorld MPICommRank];

    NSLog(@"Process %@ of %@.", rank+1, size);



    [mpi release];

    [pool release];
    return 0;
}
```

Program4.4.3 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`

```
#import <Foundation/Foundation.h>
#import <MPIObjC/MPIObjC.h>

int main (int argc, char **argv)
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    MPIInstance *mpi = [MPIInstance mpiWith:&argc :&argv];

    MPIComm *mCommWorld = [mpi commWorld];
    NSNumber *size = [mCommWorld MPICommSize];
    int rank = [[mCommWorld MPICommRank] intValue];
    NSNumber *aNumber = [NSNumber numberWithInt:rank+1];
    int i=1;
    if (rank == 0) {
        for (i = 1; i< [size intValue]; i++)
        {
            int recv = 0;
            [mCommWorld MPIRecv:&recv ofSize:1 type:MPI_INT from:i
                withTag:1];
            NSLog(@"%d", recv);
        }
    } else {
        int rank = [aNumber intValue];
        [mCommWorld MPISend:&rank ofSize:1 ofType:MPI_INT toProcess:0
            withTag:1];
    }


    [mpi release];

    [pool release];
    return 0;
}
```

Program 4.4.4 is a paralllel 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

```
/* Sieve of Eratosthenes */

#import <Foundation/Foundation.h>
#import <MPIObjC/MPIObjC.h>

#import <math.h>
#define BLOCK_LOW(id,p,n) ((id)*(n)/(p))
#define BLOCK_HIGH(id,p,n) (BLOCK_LOW((id)+1,p,n) - 1)
#define BLOCK_SIZE(id,p,n) ((BLOCK_HIGH(id,p,n) - BLOCK_LOW(id,p,n))
    +1)


int main (int argc, char **argv)
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    int count;                  // Local prime count
    double elapsed_time;        // Parallel execution time
    int first;                 //index of first multiple
    int global_count;        // Global prime count
    int high_value;            // Highest value on this proc.
    int i;
    int rank;                    // Process id number
    int index=0;                 // Index of current prime
    int low_value;             // Lowest value on this proc.
    char *marked;              // Portion of 2,...n
    int n = 60;                // Sieving from 2 to n=60
    int p;                     // Number of processes
    int proc0_size;            // Size of proc0's subarray
    int prime;                 // Current prime
    int size;                  // Elements in 'marked'



    MPIInstance *mpi = [MPIInstance mpiWith:&argc :&argv];

    MPIComm *mCommWorld = [mpi commWorld];

    // Start the timer
    [mCommWorld MPIBarrier];
    elapsed_time = -[[mpi MPIWTick] doubleValue];


    p = [[mCommWorld MPICommSize] intValue];
    rank = [[mCommWorld MPICommRank] intValue];

    // Figure out this process's share of the array, as well as the
       integers
    // represented by the first and last array elements.
    low_value = 2 + BLOCK_LOW(rank,p,n-1);
    high_value = 2 + BLOCK_HIGH(rank,p,n-1);

    size =BLOCK_SIZE(rank,p, n-1 );

    // Bail out if all the primes used for the sieving are not all
       held by process 0

    proc0_size = (n-1)/p;
    if ((2 + proc0_size) < (int) sqrt((double) n))
    {
        if (!rank)
            NSLog(@"Too many processes");
        [mpi release];
        exit(1);
    }

    // Allocate this process share of the array
    marked = (char *) malloc(size);

    if (marked == NULL)
    {
        NSLog(@"Cannot allocate enough memory");
        [mpi release];
        exit(1);
    }

    for (i = 0; i < size; i++)
        marked[i] = 0;
    if (!rank)
        index = 0;

    prime = 2;

    do {
        if (prime * prime > low_value) {
            first = prime * prime - low_value;
        } else {
            if (!(low_value % prime)) {
                first = 0;
            } else {
                first = prime - (low_value % prime);
            }
        }

        for (i = first; i < size; i+=prime)
            marked[i]=1;

        if (!rank){
            while(marked[++index]);
            prime = index + 2;
        }

        [mCommWorld MPIBcast:&prime ofSize:1 ofType:MPI_INT rank:0];
    } while (prime * prime <= n);



    count = 0;
    for (i = 0; i < size; i++)
        if (!marked[i]) count++;

    [mCommWorld MPIReduce:&count outMessage:&global_count ofSize:1
       ofType:MPI_INT withOp:MPI_SUM rank:0];



    // Stop the timer
    elapsed_time += [[mpi MPIWTime] doubleValue];

    // Prints the results

    if (!rank)
    {
        NSLog(@"%d primes are less than or equal to %d", global_count
            , n);
        NSLog(@"Total elapsed time : %f", elapsed_time);
    }



    free(marked);
    [mpi release];

    [pool release];
    return 0;
}
```

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 Obective-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, 2nd edition*, Pearson Editions, 2003

[WILU96] George V. Wilson, Paul Lu, *Parallel Programing 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

```
//
//  DownloadFile.h
//  Freshmint
//
//  Created by Jean-Matthieu
//

#import <Foundation/Foundation.h>


@interface DownloadFile : NSObject {
    NSURLDownload   *m_download;
    NSURLRequest    *m_request;
    NSString        *m_filepath;
    id              m_delegate;
}

- (DownloadFile *)initWithURL:(NSURL *)url;
- (NSData *)contentsData;
- (NSString *)contentsPath;
- (id)delegate;
- (void)setDelegate:(id)delegate;

@end

@interface NSObject (DownloadFileDelegate)

- (void)downloadFileDidFinish:(DownloadFile *)
     download;
- (void)downloadFile:(DownloadFile *)download
     didFailWithError:(NSError *)error;

@end
//
//  DownloadFile.m
//  Freshmint
//
//  Created by Jean-Matthieu
//


#import "DownloadFile.h"
#import <WebKit/WebKit.h>
#import <Foundation/NSURLResponse.h>
#import <Foundation/NSError.h>

@implementation DownloadFile

- (DownloadFile *)initWithURL:(NSURL *)url
{
    self = [ super init ];
    if ( self ) {
        m_request = [ NSURLRequest requestWithURL:
             url cachePolicy:
             NSURLRequestUseProtocolCachePolicy
             timeoutInterval: 30 ];
        m_download = [ [ NSURLDownload alloc ]
             initWithRequest: m_request delegate:
             self ];
    }
    return self;
}

- (void)dealloc
{
    NSFileManager   *fm = [ NSFileManager
         defaultManager ];
    [ fm changeCurrentDirectoryPath: [ m_filepath
         stringByDeletingLastPathComponent ] ];
    [ fm removeFileAtPath: m_filepath handler: self
          ];

    [ m_download release ];
    [ m_filepath release ];
    [ super dealloc ];
}

- (NSData *)contentsData
{
    NSData  *reply = [ NSData
    return reply;
        dataWithContentsOfFile: m_filepath ];



}


- (NSString *)contentsPath
{
    return m_filepath;
}

- (id)delegate
{
    return m_delegate;
}

- (void)setDelegate:(id)delegate
{
    m_delegate = delegate;
}

#pragma mark -
#pragma mark == NSDownload Delegate ==
- (void)download:(NSURLDownload*)download
      decideDestinationWithSuggestedFilename:(
      NSString*)filename
{
    NSString    *path = [ @"/tmp"
        stringByAppendingPathComponent: filename
        ];
    [ download setDestination: path allowOverwrite
        : YES ];
}

- (void)download:(NSURLDownload *)download
      didCreateDestination:(NSString *)path
{
    m_filepath = [ path copy ];
}

- (void)downloadDidFinish:(NSURLDownload *)download
{
    [ m_delegate downloadFileDidFinish: self ];
}

- (void)download:(NSURLDownload *)download
      didFailWithError:(NSError *)error
{
    [ m_delegate downloadFile: self
        didFailWithError: error ];
}

@end//
//  Freshmeat.h
//  Freshmint
//
//  Created by Jean-Matthieu
//

#import <Foundation/Foundation.h>
#include <XMLRPCObjC/XMLRPCObjC.h>

@interface Freshmeat : NSObject
{
    XMLRPCClient *client;
    NSMutableDictionary *sessionDictionary;

    BOOL isConnected;
}

/* [ sessionDictionary method ]
    * Parameters:
    * None
    *
    * Returns:
    * NSDictionary with session informations
    *
    * Description:
    * Returns a dictionary containing SID, API
        Version, Lifetime, logintime
*/
- (NSDictionary *)sessionDictionary;



/* [ autoLogout method ]
    * Parameters:
    * None
    *
    * Returns:
    * None
    *
    * Description:
    * Automatically logs out from Freshmeat.net
*/
- (void)autoLogout;



/* [ isConnected method ]
    * Parameters:
    * None
    *
    * Returns:
    * None
    *
    * Description:
    * Informs whether a session is active or not.
    */
- (BOOL)isConnected;



/* [ fetch_available_licenses method ]
```

```
     * Parameters:
     * None
     *
     * Returns:
     * Array of available licenses
*/
- (NSArray *)fetch_available_licenses;

 /* [ fetch_available_release_foci method ]
     * Parameters:
     * None
     *
     * Returns:
     * Struct of available release focus types and associated ID
*/
- (NSDictionary *)fetch_available_release_foci;

/* [ fetch_project_list method ]
 * Parameters (passed in struct form):
 * SID                      - Session ID to work with

 * Returns:
 * Struct consisting of "projectname_full", "projectname_short", "project_status", and "project_version"
*/

- (NSDictionary *)fetch_project_list;


/* [ fetch_branch_list method ]
     * Parameters (passed in struct form):
     * SID                      - Session ID to work with
     * project_name             - Project name to fetch branches for
     *
     * Returns:
     * Array of branch name strings.
*/
- (NSArray *)fetch_branch_list_for_project:(NSString *)project_name;


/* [ fetch_release method ]
   * Parameters (passed in struct form):
   * SID                      - Session ID
   * project_name             - Project name
   * branch_name              - Branch name
   * version                  - Release version string
   *
   * Returns:
   * Struct consisting of "version", "changes", "release_focus", and "hide_from_frontpage"
*/
- (NSDictionary *)fetch_release_for_project:(NSString *)project_name branch:(NSString *)branch_name version:(NSString *)version;


/* [ login method ]
   * Parameters (passed in struct form):
   * username                 - Regular freshmeat username
   * password                 - Regular freshmeat password
   *
   * Returns:
   * Struct of SID, lifetime, and API Version
   * SID: Session ID to be used in subsequent requests to the XML-RPC service
   * Lifetime: Lifetime of the session ID in seconds
   * API Version: API Version currently in use
*/
- (void)login:(NSString *)username password:(NSString *)password;


/* [ logout method ]
   * Parameters (passed in struct form):
   * SID                      - Session ID to terminate
   *
   * Returns:
   * Struct of "OK" => "Logout successful." if logout was successful
*/
- (void)logout:(NSString *)SID;


/* [ publish_release method ]
   * Parameters (passed in struct form):
   * SID                      - Session ID to work with
   * project_name             - Project name to submit a release for
   * branch_name              - Branch name to submit a release for
   * version                  - Version string of new release
   * changes                  - Changes list, no HTML, character limit 600 chars
   * release_focus            - Release focus ID of new release (see Appendix A)
   * hide_from_frontpage      - Set to 'Y' if release is to be hidden from
   * frontpage, everything else does not hide it
   * license                  - Branch license
   * url_homepage             - Homepage
   * url_tgz                  - Tar/GZ
   * url_bz2                  - Tar/BZ2
   * url_zip                  - Zip
   * url_changelog            - Changelog
   * url_rpm                  - RPM package
```

```
    * url_deb                  - Debian package
    * url_osx                  - OS X package
    * url_bsdport              - BSD Ports URL
    * url_purchase             - Purchase
    * url_cvs                  - CVS tree (cvsweb)
    * url_list                 - Mailing list archive
    * url_mirror               - Mirror site
    * url_demo                 - Demo site
    *
    * Returns:
    * Struct of "OK" => "submission successful"
    *
    * Notes:
    * 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'.
    *
    * For convinience, we pass a dictionary to this method
*/
- (void)publish_release:(NSDictionary *)newReleaseInfo;


/* [ withdraw_release method ]
     * Parameters (passed in struct form):
     * SID                      - Session ID
     * project_name             - Project name
     * branch_name              - Branch name
     * version                  - Release version string
     *
     * Returns:
     * Struct of "OK" => "Withdraw successful.".
*/
- (void)withdraw_release_for_project:(NSString *)project_name branch:(NSString *)branch_name version:(NSString *)version;

@end


//
//  Freshmeat.m
//  Freshmint
//
//  Created by Jean-Matthieu
//

#import "Freshmeat.h"


@implementation Freshmeat

- (id)init
{
    self = [super init];
    if (self) {
        client = [[XMLRPCClient client:[NSURL URLWithString: @"http://freshmeat.net/xmlrpc"]] retain];
        sessionDictionary = [[NSMutableDictionary alloc] init];
        isConnected = NO;
    }
    return self;
}

-(void) dealloc
{
    [sessionDictionary release];
    [client release];
    [super dealloc];
}

- (NSDictionary *)sessionDictionary
{
    return sessionDictionary;
}

- (void)autoLogout
{
    [self logout:[sessionDictionary objectForKey:@"SID"]];
}

- (BOOL)isConnected
{
    return isConnected;
}

// Freshmeat methods invocation
- (NSArray *)fetch_available_licenses
{
    NSArray *object;
    object = [client invoke:@"fetch_available_licenses" withArguments:[NSArray arrayWithObject:@""]];
    return object;
}

- (NSDictionary *)fetch_available_release_foci
{

    NSDictionary *object;
    object = [client invoke:@"fetch_available_release_foci" withArguments:[NSArray arrayWithObject:@""]];
    NSLog([object description]);
    return object;
}

-(NSDictionary *)fetch_project_list
```

```
{
    id object;
    NSDictionary *myStruct = [NSDictionary
        dictionaryWithObjects:[NSArray
        arrayWithObjects:[sessionDictionary
        objectForKey:@"SID"], nil] forKeys:[
        NSArray arrayWithObjects:@"SID", nil]];
    NSArray *args = [NSArray arrayWithObject:
        myStruct];
    object = [client invoke:@"fetch_project_list"
        withArguments:args];

    // Order projects
    NSEnumerator *objEnumerator = [object
        objectEnumerator];
    id entry;

    NSMutableDictionary *projectDictionary = [[[
        NSMutableDictionary alloc] init]
        autorelease];

    while(entry = [objEnumerator nextObject]){
        NSMutableDictionary *projectDetails;

        if (nil == (projectDetails = [
            projectDictionary objectForKey:[entry
            objectForKey:@"projectname_full"]])){
            projectDetails = [[[NSMutableDictionary
                alloc] init] autorelease];
            NSArray *branches = [NSArray
                arrayWithArray:[self
                fetch_branch_list_for_project:[
                entry objectForKey:@"
                projectname_short"]]];
            [projectDetails setObject:branches
                forKey:@"project.branches"];
        }

        NSArray *objects = [NSArray
            arrayWithObjects:[entry objectForKey:@
            "projectname_full"], [entry
            objectForKey:@"project_version"],[
            entry objectForKey:@"projectname_short
            "], nil];
        NSArray *keys = [NSArray arrayWithObjects:@
            "project.name", @"project.version", @"
            project.shortname", nil];

        NSDictionary *projectInfo = [NSDictionary
            dictionaryWithObjects:objects forKeys:
            keys];
        [projectDetails setObject:projectInfo
            forKey:@"project.info"];


        [projectDictionary setObject:projectDetails
             forKey:[entry objectForKey:@"
            projectname_full"]];
    }

    NSLog(@"%@", [projectDictionary description]);


    [sessionDictionary setObject:projectDictionary
        forKey:@"MyProjects"];

    return nil;
}

- (NSArray *)fetch_branch_list_for_project:(
    NSString *)project_name
{
    id object;
    NSDictionary *myStruct = [NSDictionary
        dictionaryWithObjects:[NSArray
        arrayWithObjects:[sessionDictionary
        objectForKey:@"SID"], project_name, nil]
        forKeys:[NSArray arrayWithObjects:@"SID",
        @"project_name", nil]];
    NSArray *args = [NSArray arrayWithObject:
        myStruct];
    object = [client invoke:@"fetch_branch_list"
        withArguments:args];
    return object;
}

- (NSDictionary *)fetch_release_for_project:(
    NSString *)project_name branch:(NSString *)
    branch_name version:(NSString *)version
{
    id object;
    NSDictionary *myStruct = [NSDictionary
        dictionaryWithObjects:[NSArray
        arrayWithObjects:[sessionDictionary
        objectForKey:@"SID"],project_name,
        branch_name, version, nil]
                                                forKeys
                                                    :[
                                                    NSAr

                                                    array
                                                    :
                                                    @
                                                    "
                                                    SID
                                                    ",
                                                    @
                                                    "
                                                    proje
                                                    ",
                                                    @
                                                    "
                                                    branc
                                                    ",
                                                    @
                                                    "
                                                    versi
                                                    ",
    NSArray *args = [NSArray arrayWithObject:
         myStruct];
    object = [client invoke:@"fetch_release"
         withArguments:args];
    //NSLog(@"%@", [object description]);
    return object;
}

- (void)login:(NSString *)username password:(
     NSString *)password
{
    id object = nil;
    [sessionDictionary removeAllObjects];
    NSDictionary *myStruct = [NSDictionary
         dictionaryWithObjects:[NSArray
         arrayWithObjects:username, password, nil]
         forKeys:[NSArray arrayWithObjects:@"
         username", @"password", nil]];
    NSArray *args = [NSArray arrayWithObject:
         myStruct];
    object = [client invoke:@"login" withArguments:
         args];


    [sessionDictionary addEntriesFromDictionary:
         object];
    NSCalendarDate *date = [NSCalendarDate date];
    [sessionDictionary setObject:date forKey:@"date
         "];
    isConnected = YES;
    // Autologout 5 sec before session ends
    [NSTimer scheduledTimerWithTimeInterval:[[
         sessionDictionary objectForKey:@"Lifetime
         "] intValue] - 5
                                         target:
                                              self
                                       selector:
                                           @selector
                                           (
                                           autoLogout
                                           )
                                       userInfo:nil
                                        repeats:NO
                                             ];

    [self fetch_project_list];

}

- (void)logout:(NSString *)SID
{

    NSDictionary *myStruct = [NSDictionary
         dictionaryWithObjects:[NSArray
         arrayWithObject:SID] forKeys:[NSArray
         arrayWithObject:@"SID"]];
    NSArray *args = [NSArray arrayWithObject:
         myStruct];
    [client invoke:@"logout" withArguments:args];

    [sessionDictionary removeAllObjects];
    isConnected = NO;
    NSLog(@"Freshmeat session terminated");
}

- (void)publish_release:(NSDictionary *)
     newReleaseInfo
{
    NSLog([newReleaseInfo description]);
    NSArray *args = [NSArray arrayWithObject:
         newReleaseInfo];
    [client invoke:@"publish_release" withArguments
         :args];
}

- (void)withdraw_release_for_project:(NSString *)
     project_name branch:(NSString *)branch_name
     version:(NSString *)version
{
    NSDictionary *myStruct = [NSDictionary
         dictionaryWithObjects:[NSArray
         arrayWithObjects:[sessionDictionary
         objectForKey:@"SID"], project_name,
         branch_name, version, nil]
                                                  forKeys
                                                     :[
                                                     NSAr

                                                     array
                                                     :
                                                     @
                                                     "
                                                     SID
                                                     ",
                                                     @
                                                     "
                                                     proje
                                                     ",
                                                     @
                                                     "
                                                     branc
                                                     ",
                                                     @
                                                     "
                                                     versi
                                                     "]];
    NSArray *args = [NSArray arrayWithObject:
         myStruct];
    [client invoke:@"withdraw_release"
         withArguments:args];
}
```

```
@end
//
//  NSStringEx.h
//  Freshmint
//
//  Created by Jean-Matthieu
//

#import <Foundation/Foundation.h>


@interface NSString (NSStringEx)

- (NSString *)trimWhitespace;
- (NSString *)trimHTML;

@end

//
//  NSStringEx.m
//  Freshmint
//
//  Created by Jean-Matthieu
//

#import "NSStringEx.h"


@implementation NSString (NSStringEx)

- (NSString *)trimWhitespace
{
    NSMutableString *str = [ [ self mutableCopy ]
        autorelease ];
    CFStringTrimWhitespace( (CFMutableStringRef)str
        );

    return (NSString *)( [ [ str copy ] autorelease
        ] );
}

- (NSString *)trimHTML
{
    int             len = [ self length ];
    NSMutableString *str = [ NSMutableString
        stringWithCapacity: len ];
    int             i = 0, level = 0;

    for ( i = 0; i < len; i++ ) {
        NSString    *check = [ self
            substringWithRange: NSMakeRange( i
            , 1 ) ];
        if ( [ check isEqualTo: @"<" ] ) {
            level++;
        } else if ( [ check isEqualTo: @">" ] ) {
            level--;
            if ( level == 0 ) {
                [ str appendString: @" " ];
            }
        } else if ( level == 0 ) {
            [ str appendString: check ];
        }
    }

    return (NSString *)( [ [ str copy ] autorelease
        ] );
}

@end
//
//  SessionController.h
//  Freshmint
//
//  Created by Jean-Matthieu
//

#import <Cocoa/Cocoa.h>
#import <webKit/WebKit.h>

#import "Freshmeat.h"

@interface SessionController : NSObject
{
    IBOutlet NSTextField *loginField;
    IBOutlet NSWindow *loginWindow;
    IBOutlet NSButton *closeLoginBtn;
    IBOutlet NSButton *newReleaseBtn;
    IBOutlet NSTableView *allProjectsTable;
    IBOutlet NSTextField *nrBranchName;
    IBOutlet NSFormCell *nrBSDField;
    IBOutlet NSFormCell *nrBZ2Field;
    IBOutlet NSFormCell *nrChangelogField;
    IBOutlet NSTextView *nrChangesTV;
    IBOutlet NSFormCell *nrCVSField;
    IBOutlet NSFormCell *nrDEBField;
    IBOutlet NSFormCell *nrDemoField;
    IBOutlet NSPopUpButton *nrFocusPopUp;
    IBOutlet NSButton *nrHideBtn;
    IBOutlet NSFormCell *nrHomeField;
    IBOutlet NSPopUpButton *nrLicensePopUp;
    IBOutlet NSFormCell *nrMirrorField;
    IBOutlet NSFormCell *nrMLField;
    IBOutlet NSFormCell *nrOSXField;
    IBOutlet NSTextField *nrProjectName;
    IBOutlet NSFormCell *nrPurchaseField;
    IBOutlet NSFormCell *nrRPMField;
    IBOutlet NSFormCell *nrTGZField;
    IBOutlet NSTextField *nrVersionField;
    IBOutlet NSWindow *nrWindow;
    IBOutlet NSFormCell *nrZipField;
    IBOutlet NSSecureTextField *passwordField;
    IBOutlet NSTableView *projectBranchTable;
    IBOutlet NSWindow *projectWindow;
    IBOutlet NSButton *withdrawBtn;
    IBOutlet WebView *releaseWebView;
    IBOutlet NSProgressIndicator *spinWheel;

    Freshmeat *freshmeat;
    NSMutableDictionary *myProjects;
    //NSDictionary *releaseFocus;
    NSDictionary *projectDictionary;
    NSArray *licenseArray;
    NSString *currentProject;
    NSString *currentBranch;
    BOOL closeAndQuit;
}

+ (id)getInstance;
- (void)sortMyProjects;
- (void)closeNow:(NSWindow *)sheet;
- (void)showLoginSheet;

- (void)downloadXMLFileForProject:(NSString *) project;
- (void)displayXMLData:(NSData *)theData;
- (void)renderHTMLWithDictionary:(NSDictionary *) aDict;
- (void) setReleaseAndLicenseMenus;

- (IBAction)doLogin:(id)sender;
- (IBAction)doNewRelease:(id)sender;
- (IBAction)doPublish:(id)sender;
- (IBAction)doWithdraw:(id)sender;
- (IBAction)closeModal:(id)sender;

@end
//
//  SessionController.m
//  Freshmint
//
//  Created by Jean-Matthieu
//

#import "SessionController.h"
#import "XMLFeeder.h"
#import "DownloadFile.h"

static SessionController *theSessionController = nil;
@implementation SessionController

+(id) getInstance
{
    // TODO: Mutex Begin
    if (theSessionController == nil) {
        theSessionController = [[SessionController alloc] init];
    }
    // TODO: Mutex End
    return theSessionController;
}


-(id) init
{
    self = [super init];
    if (self)
    {
        theSessionController = self;
        freshmeat = [[Freshmeat alloc] init];
        currentProject = nil;
        currentBranch=nil;
        closeAndQuit = YES;
    }
    return self;
}

-(void) dealloc
{
    [freshmeat release];
    //[releaseFocus release];
    if (projectDictionary)
        [projectDictionary release];
    [licenseArray release];
    [myProjects release];
    [super dealloc];
}



- (void)applicationWillTerminate:(NSNotification *) aNotification
{
    [[NSFileManager defaultManager] removeFileAtPath:@"/tmp/project.html" handler:nil];
    NSString *sid = nil;
    if ((sid = [[freshmeat sessionDictionary] objectForKey:@"SID"]) != nil)
        [freshmeat logout:sid];
}

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    NSString *bgPath = [NSBundle pathForResource:@"bg" ofType:@"tif" inDirectory:[[NSBundle mainBundle] bundlePath]];
    NSString *projectHTML = [NSString stringWithFormat:@"<HTML><HEAD></HEAD><BODY MARGINHEIGHT=0 MARGINWIDTH=0 BACKGROUND=%@ BGCOLOR=#FFFFFF></BODY></HTML>", bgPath];
    [projectHTML writeToFile:@"/tmp/project.html" atomically:NO];
    [[releaseWebView mainFrame] loadRequest:
        [NSURLRequest requestWithURL:[NSURL URLWithString:[NSString stringWithString:@"file:///tmp/project.html"]]]]];

    if (![freshmeat isConnected])
    {
        [self showLoginSheet];
    }

```

```
}

- (void) sortMyProjects
{
    // [prefsProjectTable reloadData];
    [allProjectsTable reloadData];
}

- (void)closeNow:(NSWindow *)sheet
{
    if(closeAndQuit)
        [NSApp terminate:nil];
}

- (void)showLoginSheet
{
    [NSApp beginSheet: loginWindow
       modalForWindow: projectWindow
        modalDelegate: self
       didEndSelector: @selector(closeNow:)
          contextInfo: nil];
    [NSApp runModalForWindow: loginWindow];
    // Sheet is up here.
    [NSApp endSheet: loginWindow];
    [loginWindow orderOut: self];
}

- (IBAction)doLogin:(id)sender
{

    NSString *login = [loginField stringValue];
    NSString *password = [passwordField stringValue];

    /* Handle exception that may raise on login failure... */
NS_DURING
    [freshmeat login:login password:password];
NS_HANDLER
    NSRunCriticalAlertPanel(NSLocalizedString(@"Login Incorrect",@"Login Incorrect"),
                            NSLocalizedString(@"The username and password your entered could not be authenticated. Please try again.",@"Please try again"),
                            nil, nil, nil);
    return;
NS_ENDHANDLER

    // or continue
    [NSApp stopModal];
    closeAndQuit = NO;
    myProjects = [[NSMutableDictionary alloc] initWithDictionary:[[freshmeat sessionDictionary] objectForKey:@"MyProjects"]];
    [self setReleaseAndLicenseMenus];
    [allProjectsTable reloadData];
}

- (void)setReleaseAndLicenseMenus
{
    licenseArray =[[NSArray alloc] initWithArray:[freshmeat fetch_available_licenses]];

    NSEnumerator *licenseEnum = [licenseArray objectEnumerator];
    NSString *licenseKind;
    NSMenu *licenseMenu = [[[NSMenu alloc] init] autorelease];
    while(licenseKind = [licenseEnum nextObject]){
        NSMenuItem *anItem= [[[NSMenuItem alloc] initWithTitle:licenseKind action:nil keyEquivalent:@""] retain];
        [licenseMenu addItem:anItem];
        [anItem release];
    }
    [nrLicensePopUp setMenu:licenseMenu];

    /*releaseFocus = [[NSDictionary alloc] initWithDictionary:[freshmeat fetch_available_release_foci]];
    NSMenu *releaseMenu = [[[NSMenu alloc] init] autorelease];
    int i=0;
    for (i=0; i<[releaseFocus count]; i++){
        NSMenuItem *anItem= [[[NSMenuItem alloc] initWithTitle:[NSString stringWithFormat:@"%d - %@", i, [releaseFocus objectForKey:[NSString stringWithFormat:@"%d", i]]] action:nil keyEquivalent:@""] retain];
        [releaseMenu addItem:anItem];
        [anItem release];
    }
    [nrFocusPopUp setMenu:releaseMenu];*/

}

- (IBAction)doNewRelease:(id)sender
{
    [nrProjectName setStringValue:currentProject];
    [nrBranchName setStringValue:currentBranch];
    [nrHideBtn setState:NSOffState];
    [nrChangesTV setString:@""];
    [nrVersionField setStringValue:@""];
    [nrHomeField setStringValue:@""];
    [nrTGZField setStringValue:@""];
    [nrZipField setStringValue:@""];
    [nrBZ2Field setStringValue:@""];
    [nrChangelogField setStringValue:@""];
    [nrRPMField setStringValue:@""];
    [nrDEBField setStringValue:@""];
    [nrOSXField setStringValue:@""];
    [nrBSDField setStringValue:@""];
    [nrCVSField setStringValue:@""];
    [nrMLField setStringValue:@""];
    [nrMirrorField setStringValue:@""];
    [nrDemoField setStringValue:@""];



    [nrLicensePopUp selectItemWithTitle:[
        projectDictionary objectForKey:@"license
        "]];

    [NSApp beginSheet: nrWindow
       modalForWindow: projectWindow
        modalDelegate: nil
       didEndSelector: nil
          contextInfo: nil];
    [NSApp runModalForWindow: nrWindow];
    // Sheet is up here.
    [NSApp endSheet: nrWindow];
    [nrWindow orderOut: self];
}

- (IBAction)closeModal:(id)sender
{
    [NSApp stopModal];
}



- (IBAction)doPublish:(id)sender
{
    //
    NSMutableDictionary *releaseInfo = [[
        NSMutableDictionary alloc] init];
    [releaseInfo setObject:[[freshmeat
        sessionDictionary] objectForKey:@"SID"]
        forKey:@"SID"];
    [releaseInfo setObject:currentProject forKey:@"
        project_name"];
    [releaseInfo setObject:currentBranch forKey:@"
        branch_name"];
    [releaseInfo setObject:[nrVersionField
        stringValue] forKey:@"version"];
    [releaseInfo setObject:[nrChangesTV string]
        forKey:@"changes"];
    [releaseInfo setObject:[NSString
        stringWithFormat:@"%d", [nrFocusPopUp
        indexOfSelectedItem]] forKey:@"
        release_focus"];
    if([nrHideBtn state])
        [releaseInfo setObject:@"Y" forKey:@"
            hide_from_frontpage"];
    else
        [releaseInfo setObject:@"N" forKey:@"
            hide_from_frontpage"];
    [releaseInfo setObject:[nrLicensePopUp
        titleOfSelectedItem] forKey:@"license"];

    //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:@""])
        [releaseInfo setObject:optString forKey:@"url_list"];

    if(![(optString = [nrMirrorField stringValue]) isEqualToString:@""])
        [releaseInfo setObject:optString forKey:@"url_mirror"];

    if(![(optString = [nrDemoField stringValue]) isEqualToString:@""])
        [releaseInfo setObject:optString forKey:@"url_demo"];

    NS_DURING
        [freshmeat publish_release:releaseInfo];
    NS_HANDLER
        NSRunCriticalAlertPanel(NSLocalizedString(@"Publish error",@"Publish error"),
                                NSLocalizedString(@"Could not publish new release!",@"Could not publish new release!"),
                                nil, nil, nil);
        return;
    NS_ENDHANDLER

    [releaseInfo release];

    [NSApp stopModal];
}

- (IBAction)doWithdraw:(id)sender
{
    if (NSRunCriticalAlertPanel(NSLocalizedString(@"Confirm withdraw",@"Withdraw confirmation"),
                                NSLocalizedString(@"Are you sure you want to withdraw this release?",@"Are you sure you want to withdraw this release?"),
                                nil, NSLocalizedString(@"Abort",@"Abort"), nil) != NSOKButton)
        return;
    NS_DURING
        [freshmeat withdraw_release_for_project:currentProject branch:currentBranch version:[projectDictionary objectForKey:@"latest_release_version"]]];
    NS_HANDLER
        NSRunCriticalAlertPanel(NSLocalizedString(@"Withdraw error",@"Withdraw error"),
                                NSLocalizedString(@"Could not withdraw latetst release for the selected project",@"Could not withdraw latetst release for the selected project"),
                                nil, nil, nil);
        return;
    NS_ENDHANDLER


}

- (void)downloadXMLFileForProject:(NSString *)project
{
    [ spinWheel startAnimation: self ];
    NSString *urlString = [NSString stringWithFormat:@"http://freshmeat.net/projects-xml/%@/%@.xml", project, project];
    NSURL       *theURL = [ NSURL URLWithString: urlString ];
    DownloadFile    *download = [ [ [ DownloadFile alloc ] initWithURL: theURL ] autorelease ];
    [ download setDelegate: self ];

}

- (void)displayXMLData:(NSData *)theData
{
    XMLFeeder   *parser = [ [ [ XMLFeeder alloc ] initWithData: theData ] autorelease ];
    BOOL        result = NO;
    if ( parser ) {
        result = [ parser parse ];
        if ( result == NO ) {
            NSError     *theErr = [ parser parserError ];
            if ( [ theErr code ] != 0 ) {
                NSLog( [ theErr localizedDescription ] );
            }
        } else { /* Render HTML display */
            if (projectDictionary)
                [projectDictionary release];
            projectDictionary = [[NSDictionary alloc] initWithDictionary:[parser
```

```
                projectDictionary]];
            [self renderHTMLWithDictionary:
                projectDictionary];
            //NSLog( [ [ parser projectDictionary
                ] description ] );
            //NSLog( [ [ parser projectItems ]
                description ] );
        }
    }
}

- (void)renderHTMLWithDictionary:(NSDictionary *)
    aDict
{
    NSString *percent =@"%";
    NSString *bgPath = [NSBundle pathForResource:@"
        bg" ofType:@"tif" inDirectory:[[NSBundle
        mainBundle] bundlePath]];
    if (nil == aDict){
        NSString *projectHTML = [NSString
            stringWithFormat:@"<HTML><HEAD></HEAD
            ><BODY MARGINHEIGHT=0 MARGINWIDTH=0
            BACKGROUND=%@ BGCOLOR=#FFFFFF></BODY
            ></HTML>", bgPath];
        [projectHTML writeToFile:@"/tmp/project.
            html" atomically:NO];
        [[releaseWebView mainFrame] loadRequest:
            [NSURLRequest requestWithURL:[NSURL
                URLWithString:[NSString
                stringWithString:@"file:///tmp/
                project.html"]]]];
    } else {
        NSDictionary *releaseInfo = [freshmeat
            fetch_release_for_project:[[[
            myProjects objectForKey:currentProject
            ] objectForKey:@"project.info"]
            objectForKey:@"project.shortname"]
                                        branch
                                          :[[[
                                          myProjects
                                          objectForKey
                                          :
                                          currentProject
                                          ]
                                          objectForKey
                                          :@"
                                          project
                                          .
                                          branches
                                          "]
                                          objectAtIndex
                                          :[
                                          projectBranchTable
                                          selectedRow
                                          ]]
                                        version
                                          :[[[
                                          myProjects
                                          objectForKey
                                          :
                                          currentProject
                                          ]
                                          objectForKey
                                          :@"
                                          project
                                          .info
                                          "]
                                          objectForKey
                                          :@"
                                          project
                                          .
                                          version
                                          "]];
        NSString *changelog = [NSString
            stringWithString:[releaseInfo
            objectForKey:@"changes"]];
        NSString *projectHTML = [NSString
            stringWithFormat:@"\
        <HTML>\n\
        <HEAD>\n\
        </HEAD>\n\
        <BODY MARGINHEIGHT=0 MARGINWIDTH=0
            BACKGROUND=%@ BGCOLOR=#FFFFFF>\n\
        <TABLE BORDER=0 WIDTH=100%@ CELLSPACING=0>\
            n\
        <TR><TD VALIGN=TOP WIDTH=50%@>\n\
        <H2>%@</H2>\n\
        <H5>Entry Date: %@<BR>\n\
        Last Update: %@<BR>\n\
        Latest release version: %@</H5>\n\
        <FONT SIZE=2 COLOR=#000000>\n\
        License: %@<BR>\n\
        Subscribers: %@<BR>\n\
        </FONT>\n\
        </TD>\n\
        <TD VALIGN=TOP WIDTH=50%@ ALIGN=RIGHT>\n\
        <IMG SRC=%@ WIDTH=150>\n\
        </TD>\n\
        </TR>\n\
        <TR><TD COLSPAN=2><BR>\n\
        <FONT SIZE=3 COLOR=#000000><B>Description
            :</B></FONT><BR>\n\
        <FONT SIZE=2 COLOR=#000000>%@<BR><BR>\n\
        <FONT SIZE=3 COLOR=#000000><B>Changelog:</B
            ></FONT><BR>\n\
        <FONT SIZE=2 COLOR=#000000>%@<BR><BR>\n\
        <B>Release focus: %@</B></FONT>\n\
        </TD>\n\
        </TABLE>\n\
        </BODY>\n\
        </HTML>\n", bgPath, percent, percent,
            [aDict objectForKey:@"projectname_full
                "], [aDict objectForKey:@"
                date_added"], [aDict objectForKey:
                @"date_updated"], [aDict
```

```
                  objectForKey:@"
                  latest_release_version"], [aDict
                  objectForKey:@"license"],
              [aDict objectForKey:@"subscriptions"],
                  percent, [aDict objectForKey:@"
                  screenshot_thumb"], [aDict
                  objectForKey:@"desc_full"],
                  changelog, [releaseInfo
                  objectForKey:@"release_focus"]];
        [projectHTML writeToFile:@"/tmp/project.
            html" atomically:NO];
        [[releaseWebView mainFrame] loadRequest:
            [NSURLRequest requestWithURL:[NSURL
                URLWithString:[NSString
                stringWithString:@"file:///tmp/
                project.html"]]]];
    }
}

#pragma mark  DownloadFile delegates
- (void)downloadFileDidFinish:(DownloadFile *)
    download
{
    [ self displayXMLData: [ download contentsData
         ] ];
    [newReleaseBtn setEnabled:YES];
    [withdrawBtn setEnabled:YES];
    [ spinWheel stopAnimation: self ];
}

- (void)downloadFile:(DownloadFile *)download
    didFailWithError:(NSError *)error
{
   [ spinWheel stopAnimation: self ];
}



#pragma mark Tableviews Datasource & delegates
- (int)numberOfRowsInTableView:(NSTableView *)
    aTableView
{
    if ([aTableView isEqualTo:allProjectsTable]){
        return [myProjects count];
    } else if ([aTableView isEqualTo:
         projectBranchTable] && (nil !=
         currentProject)) {
        return [[[myProjects objectForKey:
             currentProject] objectForKey:@"project
             .branches"] count];
    } else
        return 0;
}


-(id)tableView:(NSTableView *)aTableView
    objectValueForTableColumn:(NSTableColumn *)
    aTableColumn row:(int)rowIndex
{
    if ([aTableView isEqualTo:allProjectsTable]){
        return [[myProjects allKeys] objectAtIndex:
             rowIndex];
    } else if ([aTableView isEqualTo:
         projectBranchTable] && (nil !=
         currentProject)) {
        return [[[myProjects objectForKey:
             currentProject] objectForKey:@"project
             .branches"] objectAtIndex:rowIndex];
    } else
        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>
```

```


@interface XMLFeeder : NSXMLParser {
    int               m_stat;
    NSDictionary      *m_projectDictionary;
    NSMutableArray    *m_projectItems;

    NSString          *m_name;
    NSMutableString   *m_value;

    NSMutableDictionary *m_elemValue;
    NSMutableArray    *m_elements;
}

- (XMLFeeder *)initWithURL:(NSURL *)url;
- (XMLFeeder *)initWithData:(NSData *)data;

- (NSDictionary *)projectDictionary;
- (NSMutableArray *)projectItems;

@end
//
//  XMLFeeder.m
//  Freshmint
//
//  Created by Jean-Matthieu
//

#import ”XMLFeeder.h”
#import ”NSStringEx.h”

@implementation XMLFeeder

- (XMLFeeder *)initWithURL:(NSURL *)url
{
    self = [ super initWithContentsOfURL: url ];
    if ( self ) {
        [ self setDelegate: self ];
        m_stat = 0;
    }
    return self;
}

- (XMLFeeder *)initWithData:(NSData *)data
{
    self = [ super initWithData: data ];
    if ( self ) {
        [ self setDelegate: self ];
        m_stat = 0;
    }
    return self;
}

- (void)dealloc
{
    [ m_projectDictionary release ];
    [ m_projectItems release ];    [ super dealloc
        ];
}

- (NSDictionary *)projectDictionary
{
    return [m_projectItems objectAtIndex:0];
}

- (NSMutableArray *)projectItems
{
    return (m_projectItems);
}



#pragma mark -
#pragma mark == Delegate ==
- (void)parserDidStartDocument:(NSXMLParser *)
    parser
{
   // NSLog( @”parserDidStartDocument” );

    m_stat = 0;
    m_value = [ NSMutableString string ];
    m_elemValue = [ NSMutableDictionary dictionary
        ];
    m_elements = [ NSMutableArray array ];
}

- (void)parserDidEndDocument:(NSXMLParser *)parser
{
   // NSLog( @”parserDidEndDocument” );

    m_projectItems = [ m_elements copy ];
}

- (void)parser:(NSXMLParser *)parser
    didStartElement:(NSString *)elementName
    namespaceURI:(NSString *)namespaceURI
    qualifiedName:(NSString *)qName attributes:(
    NSDictionary *)attributeDict
{
   // NSLog( @”didStartElement : %@, %@, %@, %@”,
       elementName, namespaceURI, qName, [
       attributeDict description ] );

    switch ( m_stat ) {
        case 0:
            // header.
            if ( [ elementName isEqualToString: @”
                project-listing” ] ) {
            }
            m_stat++;
            break;
        case 1:
            // header elements.
            if ( [ elementName isEqualToString: @”
                project” ] ) {
                // go next.
                m_projectDictionary = [ m_elemValue
                    copy ];
                m_elemValue = [ NSMutableDictionary dictionary ];
                m_stat++;
            } else {
                [ m_value setString: @"" ];
            }
            break;
    }
}

- (void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName
{
    //NSLog( @"didEndElement : %@, %@, %@", elementName, namespaceURI, qName );
    BOOL        isItemEnd = NO;
    NSString    *desc = [ m_value trimWhitespace ];
    desc = [ desc trimHTML ];


    if ( [ elementName isEqualToString: @"project" ] ) {
        isItemEnd = YES;
    } else {
        if ( desc == nil ) {
            return;
        } else if ( [ desc isEqualToString: @"" ] ) {
            return;
        }
    }

    //NSLog( @"stat = %d : <%@>%@</%@>", m_stat, elementName, desc, elementName );

    switch ( m_stat ) {
        case 1:
            [ m_elemValue setObject: desc forKey: elementName ];
            break;
        case 2:
            if ( isItemEnd ) {
                [ m_elements addObject: m_elemValue ];
                m_elemValue = [ NSMutableDictionary dictionary ];
            } else {
                [ m_elemValue setObject: desc forKey: elementName ];
            }
            break;
    }

    [ m_value setString: @"" ];
}

- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string
{
    //NSLog( @"foundCharacters : %@", string );
    if ( [ string isEqualToString: @"" ] != YES ) {
        NSString    *desc = [ string trimWhitespace ];
        [ m_value appendString: desc ];
    }
}

@end
//
//  main.m
//  Freshmint
//
//  Created by Jean-Matthieu on Mon May 24 2004.
//  Copyright (c) 2004 __MyCompanyName__. All rights reserved.
//

#import <Cocoa/Cocoa.h>

int main(int argc, char *argv[])
{
    return NSApplicationMain(argc, argv);
}
```

MPIObjC source code

```
//
//  MPIComm.h
//  MPIObjC
//
//  Created by Jean-Matthieu on 14/08/2004.
//  Copyright 2004 __MyCompanyName__. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "mpi.h"
#import "MPIRequest.h"

@class MPIInstance;

/**
* MPIComm - MPI Communication object.
*/
```

```
    MPI_Comm comm;

@interface MPIComm : NSObject {



}


#pragma mark -
#pragma mark Constructor
/**
* Initiate a MPIComm object.
* @param aComm: A communicator, such as
      MPI_COMM_WORLD, MPI_COMM_SELF, MPI_COMM_NULL
        ...
* @return Returns an MPIComm instance; a
      communicator for your MPI environment.
*/
- (id) initWithCommunicator:(MPI_Comm)aComm;

#pragma mark -
#pragma mark Communicator functions

/**
* Communicator size.
* Determines the size of the group associated with
      a communicator.
* @return The size of the group as a NSNumber.
*/
- (NSNumber *) MPICommSize;

/**
* Process rank.
* Determines the rank of the calling process in the
       communicator.
* @return The rank of the calling process as a
      NSNumber.
*/
- (NSNumber *) MPICommRank;

/**
* Communicator duplicator.
* Duplicates an existing communicator with all its
      cached information.
* @return A duplicated MPIComm object.
*/
- (MPIComm *) MPICommDup;

/**
* Communicator splitter
* Creates new communicators based on colors and
      keys.
* @param color: An integer to specify the color,
      control of subset assignment. The
* color must be non-negative or MPI_UNDEFINED.
* @param aKey: An integer to specify the key,
      control of rank assigment.
* @return A new MPIComm instance.
*/
- (MPIComm *) MPICommSplit:(int)color andKey:(int)
      aKey;

/**
* Free a communicator.
* Marks the communicator object for deallocation
*/
- (void) MPICommFree;

#pragma mark -
#pragma mark Point-to-Point Communications (
      Blocking)

/**
* Sending data.
* Performs a basic send.
* @param message: Message to send.
* @param messageSize: Number of element in the sent
       message.
* @param type: The message type (ie: MPI_CHAR,
      MPI_INT ...).
* @param dest: Rank of process to send the data to
       (integer).
* @param tag: Message tag.
*/
- (void) MPISend:(void *)message ofSize:(int)
      messageSize ofType:(MPI_Datatype)type
      toProcess:(int)dest withTag:(int)tag;



/**
* Receiving data.
* Basic receive from a process.
* @param outBuffer: A buffer to store the received
      message.
* @param messageSize: The expected message size.
* @param type: The message type (ie: MPI_CHAR,
      MPI_INT ...).
* @param src: Rank of the sending process.
* @param tag: Message tag.
* @return
*/
- (void) MPIRecv:(void*)outBuffer ofSize:(int)
      messageSize type:(MPI_Datatype)type from:(int)
      src withTag:(int)tag;



/**
* Send and receive a message.
* This method sends and receive a message.
* @param message: Message to send.
* @param sMessageSize: Number of element in the
      sent message.
* @param sType: The message type (ie: MPI_CHAR,
      MPI_INT ...)
* @param dest: Rank of process to send the data to
       (integer).
```

```
* @param sTag: Message tag.
* @param outBuffer: A buffer to store the received
     message.
* @param rMessageSize: The expected message size.
* @param rType: The message type (ie: MPI_CHAR,
     MPI_INT ...)
* @param src: Rank of the sending process.
* @param rtag: Message tag.
* @return
*/
- (void) MPISendRecv:(void *)message ofSize:(int)
     sMessageSize ofType:(MPI_Datatype)sType
     toProcess:(int)dest withTag:(int)sTag
     outMessage:(void*)outBuffer outSize:(int)
     rMessageSize type:(MPI_Datatype)rType from:(
     int)src withTag:(int)rTag;

/**
* Get the number of received elements.
* @param forType: The message type (ie: MPI_CHAR,
     MPI_INT ...)
* @return The number of received elements as a
     NSNumber.
*/
- (NSNumber *) MPIGetCount:(MPI_Datatype)forType;

#pragma mark -
#pragma mark Point-to-Point Communications (Non-
     Blocking)

/**
* Sending data.
* Performs a basic send.
* @param message: Message to send.
* @param messageSize: Number of element in the sent
      message.
* @param type: The message type (ie: MPI_CHAR,
     MPI_INT ...).
* @param dest: Rank of process to send the data to
      (integer).
* @param tag: Message tag.
*/
- (void) MPIISend:(void *)message ofSize:(int)
     messageSize ofType:(MPI_Datatype)type
     toProcess:(int)dest withTag:(int)tag;


/**
* Receiving data.
* Basic receive from a process.
* @param A buffer to store the received message.
* @param The expected message size.
* @param The message type (ie: MPI_CHAR, MPI_INT
      ...).
* @param Rank of the sending process.
* @param tag: Message tag.
* @return
*/
- (void) MPIIRecv:(void*)outBuffer ofSize:(int)
     messageSize type:(MPI_Datatype)type from:(int)
     src withTag:(int)tag;

#pragma mark -
#pragma mark Collective Communications

/**
* Process Synchronization
* Performs a barrier synchronization among all
     process in the communicator.
*/
- (void) MPIBarrier;

/**
* Message Broadcast.
* Broadcast a message to all process in the
     communicator world.
* @param message: Message to send.
* @param size: Number of elements in the buffer.
* @param type: The message type (ie: MPI_CHAR,
     MPI_INT ...).
* @param rootProcess: Rank of process with message
     to broadcast.
*/

- (void) MPIBcast:(void *)message ofSize:(int)size
     ofType:(MPI_Datatype)type rank:(int)
     rootProcess;

#pragma mark Message gathering

/**
* Basic Message gathering.
* Gather messages from all process in the
     communicator.
* @param message: Message to send.
* @param count: Number of element in the sent
     message.
* @param type: The sent message type (ie: MPI_CHAR
     , MPI_INT ...).
* @param outBuffer: A buffer to store the received
     message.
* @param outSize: Number of element in the received
      buffer.
* @param outType: The received message type (ie:
     MPI_CHAR, MPI_INT ...).
* @param rootProcess: Rank of gathering process.
* @return
*/

- (void) MPIGather:(void *)message ofSize:(int)
     count ofType:(MPI_Datatype)type outMessage:(
     void*) outBuffer outSize:(int)outSize outType
     :(MPI_Datatype)outType rank:(int)rootProcess;

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

```
* @param message: Message to send.
* @param count: Number of element in the sent
    message.
* @param type: The sent message type (ie: MPI_CHAR
    , MPI_INT ...).
* @param outBuffer: A buffer to store the received
    message.
* @param outSize: Number of element in the received
    buffer.
* @param displs: Displacement in received message
    of elements gathered from all processes.
* @param outType: The received message type (ie:
    MPI_CHAR, MPI_INT ...).
* @param rootProcess: Rank of gathering process.
* @return
*/

- (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;

/**
* All-gather operation.
* Gather messages from all process in all proccess
    in the communicator.
* @param message: Message to send.
* @param count: Number of element in the sent
    message.
* @param type: The sent message type (ie: MPI_CHAR
    , MPI_INT ...).
* @param outBuffer: A buffer to store the received
    message.
* @param outSize: Number of element in the received
    buffer.
* @param outType: The received message type (ie:
    MPI_CHAR, MPI_INT ...).
* @return
*/

- (void) MPIAllGather:(void *)message ofSize:(int)
    count ofType:(MPI_Datatype)type outMessage:(
    void*)outBuffer outSize:(int)outSize outType:(
    MPI_Datatype)outType;


#pragma mark Message scattering

/**
* Basic Message scattering.
* Distribute individual messages from root to each
    process in the communicator.
* @param message: Message to send.
* @param count: Number of element in the sent
    message.
* @param type: The sent message type (ie: MPI_CHAR
    , MPI_INT ...).
* @param A buffer to store the received message.
* @param outSize: Number of element in the received
    buffer.
* @param outType: The received message type (ie:
    MPI_CHAR, MPI_INT ...).
* @param rootProcess: Rank of scattering process.
* @return
*/

- (void) MPIScatter:(void *)message ofSize:(int)
    count ofType:(MPI_Datatype)type outMessage:(
    void*)outBuffer outSize:(int)outSize outType:(
    MPI_Datatype)outType rank:(int)rootProcess;

/**
* Complex message scattering
* Distribute individual messages from root to each
    process in the communicator.
* Messages can have different sizes and
    displacements.
* @param message: Message to send.
* @param count: Number of element in the sent
    message.
* @param displs: Displacement in sent message.
* @param type: The sent message type (ie: MPI_CHAR
    , MPI_INT ...).
* @param A buffer to store the received message.
* @param outSize: Number of element in the received
    buffer.
* @param outType: The received message type (ie:
    MPI_CHAR, MPI_INT ...).
* @param rootProcess: Rank of scattering process.
* @return
*/

- (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;


#pragma mark Reductions

/**
* Reduction computation.
* Reduces values on all processes to a single value
    .
* @param message: Message to send.
* @param outBuffer: A buffer to store the received
    message.
* @param size: Number of elements in the received
    message.
* @param type: The message datatype.
* @param operation: The reduction operation.
* @param rank: Rank of the reducing process.
* @return
*/

- (void) MPIReduce:(void *)message outMessage:(void
     *)outBuffer ofSize:(int)size ofType:(
     MPI_Datatype)type withOp:(MPI_Op)operation
     rank:(int)rank;

/**
* Reduction computation.
* Combines values from all processes and
     distributes the result back to all processes.
* @param message: Message to send.
* @param outBuffer: A buffer to store the received
     message.
* @param size: Number of elements in the received
     message.
* @param type: The message datatype.
* @param operation: The reduction operation.
* @return
*/

- (void) MPIAllReduce:(void *)message outMessage:(
     void*)outBuffer ofSize:(int)size ofType:(
     MPI_Datatype)type withOp:(MPI_Op)operation;

/**
* Reduces and scatters a message.
* Combines values and scatters the results.
* @param message: Message to send.
* @param outBuffer: A buffer to store the received
     message.
* @param rSize: Number of elements in the received
     message.
* @param type: The message datatype.
* @param operation:The reduction operation.
* @return
*/

- (void) MPIReduceScatter:(void *)sMessage
     outMessage:(void*)outBuffer outSize:(int *)
     rSize ofType:(MPI_Datatype)type withOp:(MPI_Op
     ) operation;

/**
* Partial reduction computation
* Computes the scan (partial reductions) of data
      on a collection of processes.
* @param message: Message to send.
* @param outBuffer: A buffer to store the received
     message.
* @param size: Number of elements in the received
     message.
* @param type: The message datatype.
* @param operation:The reduction operation.
* @return
*/

- (void)MPIScan:(void*)message outMessage:(void*)
     outBuffer  ofSize:(int)size ofType:(
     MPI_Datatype)type withOp:(MPI_Op)operation;

#pragma mark All to all communications


/**
* Sends data from all to all processes.
* @param message: Message to send.
* @param inSize: Number of element in the sent
     message.
* @param inType: Type of sent message.
* @param outBuffer: A buffer to store the received
     message.
* @param outSize: Size of the received message.
* @param outType: Type of received message.
* @return
*/

- (void) MPIAllToAll:(void *)message ofSize:(int)
     inSize ofType:(MPI_Datatype)inType outMessage
     :(void*)outBuffer  outSize:(int)outSize
     outType:(MPI_Datatype)outType;

/**
* Sends data from all to all processes, with a
     displacement.
* @param message: Message to send.
* @param inSize: Number of element in the sent
     message.
* @param inDispls: Displacement in sent message.
* @param inType: Type of sent message.
* @param outBuffer: A buffer to store the received
     message.
* @param outSize: Size of the received message.
* @param outDispls: Displacement in received
     message.
* @param outType: Type of received message.
* @return
*/
- (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;


@end
//
//  MPIComm.m
//  MPIObjC
//
//  Created by Jean-Matthieu on 14/08/2004.
//  Copyright 2004 __MyCompanyName__. All rights
     reserved.
//

#import "MPIComm.h"
#import "MPIInstance.h"

@implementation MPIComm
- (id) initWithCommunicator:(MPI_Comm)aComm
{
    self = [super init];
    if (self)
    {
        comm = aComm;
        return self;
    }
    return nil;
}

- (void) dealloc
{
    MPI_Comm_free(&comm);
    [super dealloc];
}


#pragma mark -
#pragma mark Communicator functions

- (NSNumber *) MPICommSize
{
    int size = 0;
    MPI_Comm_size(comm, &size);
    NSNumber *theSize = [[[NSNumber alloc]
        initWithInt:size] autorelease];
    return theSize;
}

- (NSNumber *) MPICommRank
{
    int rank = 0;
    MPI_Comm_rank(comm, &rank);
    NSNumber *theRank = [[[NSNumber alloc]
        initWithInt:rank] autorelease];
    return theRank;

}

- (MPIComm *) MPICommDup
{
    MPI_Comm newComm = MPI_COMM_NULL;
    MPI_Comm_dup(comm, &newComm);
    MPIComm *aComm = [[MPIComm alloc]
        initWithCommunicator:newComm];
    return [aComm autorelease];
}

- (MPIComm *) MPICommSplit:(int)color andKey:(int)
    aKey
{
    MPI_Comm newComm = MPI_COMM_NULL;
    MPI_Comm_split(comm, color, aKey, &newComm);
    MPIComm *aComm = [[[MPIComm alloc]
        initWithCommunicator:newComm] autorelease
        ];
    return [aComm autorelease];
}

- (void) MPICommFree
{
    MPI_Comm_free(&comm);
}


#pragma mark -
#pragma mark Point-to-Point Communications (
    Blocking)

- (void) MPISend:(void *)message ofSize:(int)
    messageSize ofType:(MPI_Datatype)type
    toProcess:(int)dest withTag:(int)tag
{
    MPI_Send(message, messageSize, type, dest, tag
        , comm);
}

- (void) MPIRecv:(void*)outBuffer ofSize:(int)
    messageSize type:(MPI_Datatype)type from:(int)
    src withTag:(int)tag
{
    MPI_Status status;
    MPI_Recv(outBuffer, messageSize, type, src, tag
        , comm, &status);
    [[MPIInstance getInstance] setStatus:status];
}

- (void) MPISendRecv:(void *)message ofSize:(int)
    sMessageSize ofType:(MPI_Datatype)sType
    toProcess:(int)dest withTag:(int)sTag
    outMessage:(void*)outBuffer outSize:(int)
    rMessageSize type:(MPI_Datatype)rType from:(
    int)src withTag:(int)rTag{
    [self MPISend:message ofSize:sMessageSize
        ofType:sType toProcess:dest withTag:sTag];
    [self MPIRecv:outBuffer ofSize:rMessageSize
        type:rType from:src withTag:rTag];
}

- (NSNumber *) MPIGetCount:(MPI_Datatype)forType
{
    MPI_Status status = [[MPIInstance getInstance]
        status];
    int count = 0;
    MPI_Get_count(&status, forType, &count);
    NSNumber *theCount = [[[NSNumber alloc]
        initWithInt:count] autorelease];
    return theCount;
}

#pragma mark -
#pragma mark Point-to-Point Communications (
    Blocking)

- (void) MPIISend:(void *)message ofSize:(int)
    messageSize ofType:(MPI_Datatype)type
    toProcess:(int)dest withTag:(int)tag{
    NSString *dictionaryKey = [NSString
        stringWithFormat:@"S-%d", tag];
    MPI_Request request;

    MPI_Issend(message, messageSize, type, dest, tag, comm, &request);

    MPIRequest *aRequest = [[[MPIRequest alloc] initWithRequest:request] autorelease];
    [[[MPIInstance getInstance] requestDictionary] setObject:aRequest forKey:dictionaryKey];
}

- (void) MPIIRecv:(void*)outBuffer ofSize:(int) messageSize type:(MPI_Datatype)type from:(int) src withTag:(int)tag
{
    NSString *dictionaryKey = [NSString stringWithFormat:@"S-%d", tag];
    MPI_Request request;
    MPI_Irecv(outBuffer, messageSize, type, src, tag, comm, &request);

    MPIRequest *aRequest = [[[MPIRequest alloc] initWithRequest:request] autorelease];
    [[[MPIInstance getInstance] requestDictionary] setObject:aRequest forKey:dictionaryKey];
}


#pragma mark -
#pragma mark Collective Communications

- (void) MPIBarrier
{
    MPI_Barrier(comm);
}

- (void) MPIBcast:(void *)message ofSize:(int)size ofType:(MPI_Datatype)type rank:(int) rootProcess
{
    MPI_Bcast(message, size, type, rootProcess, comm);
}

#pragma mark Message gathering

- (void) MPIGather:(void *)message ofSize:(int) count ofType:(MPI_Datatype)type outMessage:(void*)outBuffer outSize:(int)outSize outType:(MPI_Datatype)outType rank:(int)rootProcess
{
    MPI_Gather(message, count, type, outBuffer, outSize, outType, rootProcess, comm);
}

- (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
{
    MPI_Gatherv(message, count, type, outBuffer, outSize, displs, outType, rootProcess, comm);
}

- (void) MPIAllGather:(void *)message ofSize:(int) count ofType:(MPI_Datatype)type outMessage:(void*)outBuffer outSize:(int)outSize outType:(MPI_Datatype)outType
{
    MPI_Allgather(message,count,type,outBuffer, outSize,outType,comm);
}


#pragma mark Message scattering
- (void) MPIScatter:(void *)message ofSize:(int) count ofType:(MPI_Datatype)type outMessage:(void*)outBuffer outSize:(int)outSize outType:(MPI_Datatype)outType rank:(int)rootProcess
{
    MPI_Scatter(message, count, type, outBuffer, outSize, outType, rootProcess, comm);
}

- (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
{
    MPI_Scatterv(message, count, displs, type, outBuffer, outSize, outType, rootProcess, comm);
}


#pragma mark Reductions
- (void) MPIReduce:(void *)message outMessage:(void *)outBuffer ofSize:(int)size ofType:(MPI_Datatype)type withOp:(MPI_Op)operation rank:(int)rank
{
    MPI_Reduce(message, outBuffer, size, type, operation, rank, comm);
}

- (void) MPIAllReduce:(void *)message outMessage:(void*)outBuffer ofSize:(int)size ofType:(MPI_Datatype)type withOp:(MPI_Op)operation
{
    MPI_Allreduce(message, outBuffer, size, type, operation, comm);
}

- (void) MPIReduceScatter:(void *)sMessage outMessage:(void*)outBuffer outSize:(int *) rSize ofType:(MPI_Datatype)type withOp:(MPI_Op
) operation
{
    MPI_Reduce_scatter(sMessage, outBuffer, rSize, type, operation, comm);
}

- (void)MPIScan:(void*)message outMessage:(void*)outBuffer  ofSize:(int)size ofType:(MPI_Datatype)type withOp:(MPI_Op)operation
{
    MPI_Scan(message, outBuffer, size, type, operation, comm);
}

#pragma mark All to all communications
- (void) MPIAllToAll:(void *)message ofSize:(int)inSize ofType:(MPI_Datatype)inType outMessage:(void*)outBuffer  outSize:(int)outSize outType:(MPI_Datatype)outType
{
    MPI_Alltoall(message, inSize, inType, outBuffer, outSize, outType, comm);
}

- (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
{
    MPI_Alltoallv(message, inSize, inDispls, inType, outBuffer, outSize, outDispls, outType, comm);
}

@end
/*
 *  MPICommTest.h
 *  MPIObjC
 *
 *  Created by Jean-Matthieu on 16/08/2004.
 *  Copyright 2004 __MyCompanyName__. All rights reserved.
 *
 */

#include <Carbon/Carbon.h>

//
//  MPIInstance.h
//  MPIObjC
//
//  Created by Jean-Matthieu on 14/08/2004.
//  Copyright 2004 __MyCompanyName__. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "mpi.h"

#import "MPIComm.h"


/**
* MPIInstance. An instance class for MPI operation.
*/

@interface MPIInstance : NSObject {

    MPIComm *mCommWorld;
    MPI_Status mMPIStatus;
    NSString *mBuffer;

    NSMutableDictionary *requestDictionary;
}


#pragma mark -
#pragma mark Class methods

/**
* Instanciate MPI.
* Initialialized the MPI environment. Always call this function !
* @param argc : from the command line.
* @param argv : from the command line.
* @return Returns an MPI Instance
*/
+ (id) mpiWith:(int*)argc :(char ***)argv;

/**
* Get MPI Instance.
* A utility function to access the MPI environment. It is better to initialized
* your MPI environment before calling this function.
* @return Returns a previously instanciated MPIInstance
*/
+ (MPIInstance *) getInstance;


#pragma mark -
#pragma mark MPI Functions

/**
* MPI Environmental functions.
* Functions to access MPI environment variables
*/


/**
* Finalize MPI.
* Terminates MPI execution environment. All processes must call this routine
* before exiting.
* You do not need to call this method, MPI_Finalize() is called automatically
* when you release MPIInstance.
*/
- (void) MPIFinalize;

/**
* Abort MPI.
* Terminates MPI execution environment.
*/
- (void) MPIAbort;

/**
* Check MPI state.
* Indicates whether MPI_Init has been called.
* @return Returns TRUE if MPI_Init has been called
    , FALSE otherwise.
*/
- (BOOL) MPIInitialized;

/**
* MPI Time.
* Returns an elapsed time on the calling processor
* @return Time in seconds since an arbitrary time
    in the past.
*/
- (NSNumber *) MPIWTime;

/**
* Returns the resolution of MPI_Wtime.
* @return Time in seconds of resolution of
    MPI_Wtime
*/
- (NSNumber *) MPIWTick;

/**
* MPI Processor name.
* Gets the name of the processor.
* @return Returns the name of the processor as a
    NSString.
*/
- (NSString *) MPIGetProcessorName;

#pragma mark -
#pragma mark Accessors

/**
* Accesors.
* MPIInstance variables Accesors.
*/

/**
* @return Returns MPI World Communicator.
*/
- (MPIComm *) commWorld;

/**
* Set MPIInstance communicator.
* @param An MPIComm object.
*/
- (void) setCommWorld:(MPIComm *) aComm;


/**
* MPI Environment status.
* Get MPI environment status.
* @return Returns MPI Status.
*/
- (MPI_Status) status;

/**
* Set MPI environment status.
* @param status: A MPI_Status tag.
*/
- (void) setStatus:(MPI_Status) status;
/**
* Dictionary of MPI non blocking request. Keys are
    message tags.
*/
- (NSMutableDictionary *) requestDictionary;

#pragma mark -
#pragma mark Utility functions
/**
* 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).
*/
- (void) fixMacMPI;

@end



//
//  MPIInstance.m
//  MPIObjC
//
//  Created by Jean-Matthieu on 14/08/2004.
//  Copyright 2004 __MyCompanyName__. All rights
    reserved.
//

#import "MPIInstance.h"
#import <unistd.h>

MPIInstance *theMPIInstance = nil;

@implementation MPIInstance

- (id) init
{
    self = [super init];

    if (self)
    {
        //[self _fixMacMPI];
        mBuffer = [[NSString alloc] init];
        requestDictionary = [[NSMutableDictionary
            alloc] init];
        theMPIInstance = self;
        return self;
    }

    return nil;
}

-(void) dealloc
{
    MPI_Finalize();
    [MPIComm release];
    [requestDictionary release];
    [mBuffer release];
    [super dealloc];
}

#pragma mark -
#pragma mark Class factory methods

+ (id) mpiWith:(int*) argc :(char ***)argv
{
    MPIInstance *mpi = [[MPIInstance alloc] init];

    // Initialize MPI
    MPI_Init(argc, argv);
    MPIComm *worldComm = [[MPIComm alloc]
        initWithCommunicator:MPI_COMM_WORLD];
    [mpi setCommWorld:worldComm];

    [worldComm release];

    return mpi;
}

+ (MPIInstance *) getInstance
{
    // TODO: Mutex Begin
    if (theMPIInstance == nil) {
        int argc = 0;
        char **argv = NULL;
        theMPIInstance = [MPIInstance mpiWith:&argc
            :&argv];
        return theMPIInstance;
    }
    // TODO: Mutex End
    return theMPIInstance;
}


#pragma mark -
#pragma mark MPI Functions

- (void) MPIFinalize
{
    MPI_Finalize();
}

- (void) MPIAbort
{
    MPI_Abort(MPI_COMM_WORLD, 1);
}

- (BOOL) MPIInitialized
{
    int *flag = 0;
    MPI_Initialized(flag);
    if (flag)
        return TRUE;
    else
        return FALSE;

    return FALSE;
}

- (NSNumber *) MPIWTime
{
    double wtime = MPI_Wtime();
    NSNumber *time = [[[NSNumber alloc]
        initWithDouble:wtime] autorelease];
    return time;

}

- (NSNumber *) MPIWTick
{
    double wtick = MPI_Wtick();
    NSNumber *tick = [[[NSNumber alloc]
        initWithDouble:wtick] autorelease];
    return tick;

}

- (NSString *) MPIGetProcessorName
{
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Get_processor_name(processor_name,&namelen)
        ;

    NSString *name = [[[NSString alloc]
        initWithCString:processor_name length:
        namelen] autorelease];

    return name;
}

#pragma mark -
#pragma mark Accessors

- (MPIComm *) commWorld
{
    return mCommWorld;
}
```

```

- (void) setCommWorld:(MPIComm *) aComm
{
    if (mCommWorld)
        [mCommWorld release];
    mCommWorld = [aComm retain];
}

- (MPI_Status) status
{
    return mMPIStatus;
}

- (void) setStatus:(MPI_Status) status
{
    mMPIStatus = status;
}

- (NSMutableDictionary *) requestDictionary
{
    return requestDictionary;
}

#pragma mark -
#pragma mark Utility functions
- (void) fixMacMPI
{
    NSString *path = [[NSBundle mainBundle] bundlePath];
    char cpath[1024];

    [path getCString:cpath];

    {
        char *lastSlash, *tp;
        for(lastSlash=tp=cpath; tp=strchr(tp, '/'); lastSlash=tp++) ;
        lastSlash[1]=0; //specifies the parent of the bundle directory
    }

    chdir(cpath);
}


@end
//
//  MPIMessage.h
//  MPIObjC
//
//  Created by Jean-Matthieu on 16/08/2004.
//  Copyright 2004 __MyCompanyName__. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "mpi.h"

@interface MPIMessage : NSObject {
    id buffer;
    int length;
    MPI_Datatype datatype;
}

- (id) initWithBuffer:(id)buffer andDatatype:(MPI_Datatype) dType;

- (id) buffer;
- (void) setBuffer:(id)aBuffer;
- (int) length;
- (MPI_Datatype) datatype;

@end
//
//  MPIMessage.m
//  MPIObjC
//
//  Created by Jean-Matthieu on 16/08/2004.
//  Copyright 2004 __MyCompanyName__. All rights reserved.
//

#import "MPIMessage.h"


@implementation MPIMessage
- (id) initWithBuffer:(id)aBuffer andDatatype:(MPI_Datatype) dType
{
    self = [super init];
    if (self)
    {
        [self setBuffer:buffer];
        datatype = aDatatype;
        return self;
    }
    return nil;
}

- (void) dealloc
{
    [buffer release];
    [super dealloc];
}

- (id) buffer
{
    return buffer;
}

- (void) setBuffer:(id)aBuffer
{
    if (buffer)
        [buffer release];
    buffer = [aBuffer retain];

    if ([buffer isKindOfClass:[NSString class]]){
        length = [buffer length];
    } else if ([buffer isKindOfClass:[NSNumber class]]){
        length = [[buffer stringValue] length];
    }
}

- (int) length{
    return length;
}

- (MPI_Datatype) datatype
{
    return datatype;
}
@end
//
//  MPIRequest.h
//  MPIObjC
//
//  Created by Jean-Matthieu on 16/08/2004.
//  Copyright 2004 __MyCompanyName__. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "mpi.h"


/**
* MPIRequest - MPI Request object.
*/

@interface MPIRequest : NSObject {
    MPI_Request request;
    MPI_Status status;
}
/**
* MPIRequest Constructor.
* Initiate a MPIRequest object with request aRequest.
* @param aRequest: A MPI_Request handle.
* @return Returns an instantiated MPIRequest object.
*/
- (id) initWithRequest:(MPI_Request)aRequest;

/**
* Completes a non-blocking operation.
* MPIWait waits for a MPI send or receive to complete.
*/
- (void) MPIWait;

/**
* Tests a non-blocking operation.
* Tests for the completion of a send or receive.
* @return True if operation competed. Sets the status member of the MPIRequest object.
*/
- (int) MPITest;

/**
* Frees a request.
* Frees a communication request object.
*/
- (void) MPIRequestFree;

/*
- (void) MPIWaitAll:(int)count;

- (int) MPIWaitAny:(int)count;
*/
- (MPI_Status) status;

- (void) setStatus:(MPI_Status) aStatus;

@end
//
//  MPIRequest.m
//  MPIObjC
//
//  Created by Jean-Matthieu on 16/08/2004.
//  Copyright 2004 __MyCompanyName__. All rights reserved.
//

#import "MPIRequest.h"


@implementation MPIRequest
- (id) initWithRequest:(MPI_Request)aRequest
{
    self = [super init];
    if (self)
    {
        request = aRequest;
        return self;
    }
    return nil;
}

- (void) dealloc
{
    [super dealloc];
}


- (void) MPIWait
{
    MPI_Wait(&request, &status);
}

- (int) MPITest
{
    int flag = 0;
    MPI_Test(&request, &flag, &status);
    return flag;
}
```

```

- (void) MPIRequestFree
{
    MPI_Request_free(&request);
}
/*
- (void) MPIWaitAll:(int)count
{
    MPI_Waitall(count, &request, &status);
}

- (int) MPIWaitAny:(int)count
{
    int index = 0;
    MPI_Waitany(count, &request, &index, &status);
    return index;
}
*/

- (MPI_Status) status
{
    return status;
}

- (void) setStatus:(MPI_Status) aStatus
{
    status = aStatus;
}
@end
/*
 *  MpiObjC.h
 *  MPIObjC
 *
 *  Created by Jean-Matthieu on 15/08/2004.
 *  Copyright 2004 __MyCompanyName__. All rights
     reserved.
 *
 */

#import "mpi.h"
#import "MPIInstance.h"
#import "MPIComm.h"
```