The Scooby Guide

A Programming Guide for Writers of Mobile Agents and Locations.

By

Paul Barry
Institute of Technology, Carlow, Ireland
paul.barry@itcarlow.ie

http://glasnost.itcarlow.ie/~scooby/index.html

May 30th, 2003.

Introduction

This document is designed to get programmers up-and-running, writing mobile agents based on *Scooby* as quickly as possible. It is assumed that *Scooby* has been installed as per the instructions found on the *Scooby* Website, which is located on-line at:

http://glasnost.itcarlow.ie/~scooby/installation/index.html

Start Your Engines

Well, actually, start your Key Server ... or, at the very least, make sure a Key Server is operating. If a Key Server isn't available either contact your sysadmin or install/prepare/configure one using the installation instructions mentioned above.

Before You Do Anything Else

Before you do anything else, make sure the ".scoobyrc" configuration file exists in your HOME directory. This configuration file tells both your mobile agents and your Locations how to find their Key Server. An example ".scoobyrc" configuration file might look like this:

```
KEYSERVER=keyserver.itcarlow.ie
```

That's right, a single line with the word "KEYSERVER", followed by the equals sign "=", followed by the IP name or address of the machine running the Key Server. Be careful not to include spaces around the equals sign (or *Scooby* will, gulp, die).

Creating Locations

Before a mobile agent can relocate from one machine to another, it needs somewhere to relocate to. This is the role of the Location. Creating Locations within *Scooby* is straightforward, simply use the Mobile::Location class, and create a new Location object:

```
use strict;
use Mobile::Location;

my $location = Mobile::Location->new;
```

Once the object exists, start a concurrent server with:

```
$location->start_concurrent;
```

or start a sequential server with:

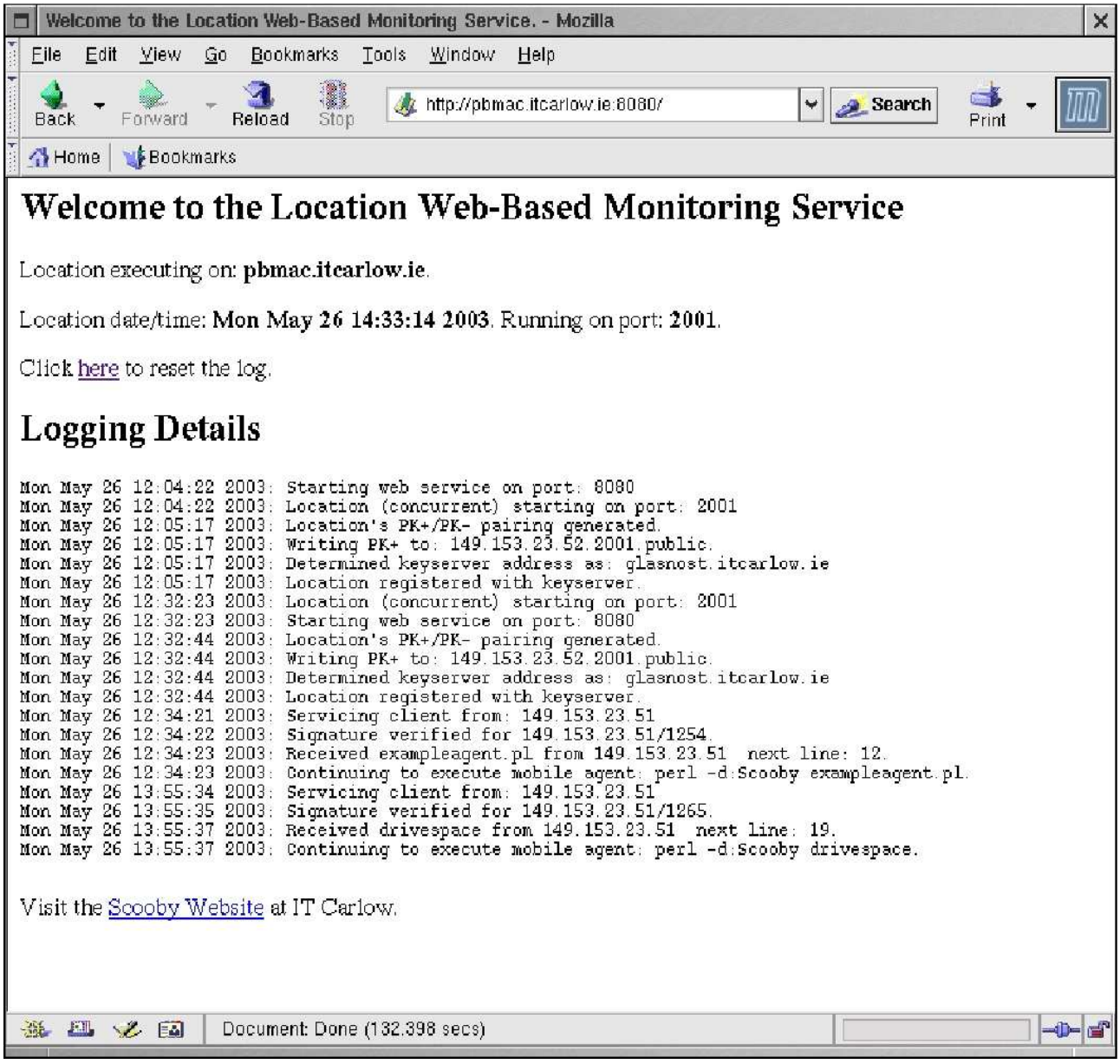
```
$location->start_sequential;
```

Whether you choose to run a concurrent or sequential server is entirely up to you.

And that's all there is to it! A Location is now executing on your computer at protocol port 2001. It will now wait passively forever for a mobile agent to arrive. It also has a web server listening at protocol port 8080, should you wish to use a web browser to remotely check on the status of the Location.

When the web server is contacted, it should display something similar to the following:

(Please see the screenshot on the next page).



The “new” constructor can take up to four named (optional) parameters:

Debug – set this to 1 to enable the production of screen status messages (on the Location). By default, this setting is off.
Port – set this to a protocol port of your choice to operate your Location on a protocol port other than 2001 (the default).
Log – set this to 1 to log a copy of any received mobile agent prior to mutation by the Location. This is really only useful when debugging *Scooby*, so you may wish to leave the default setting as it is, which is off.
Web – set this to 0 to switch off the web server (and its associated logging) running at protocol port 8080. By default, the web server is switched on.

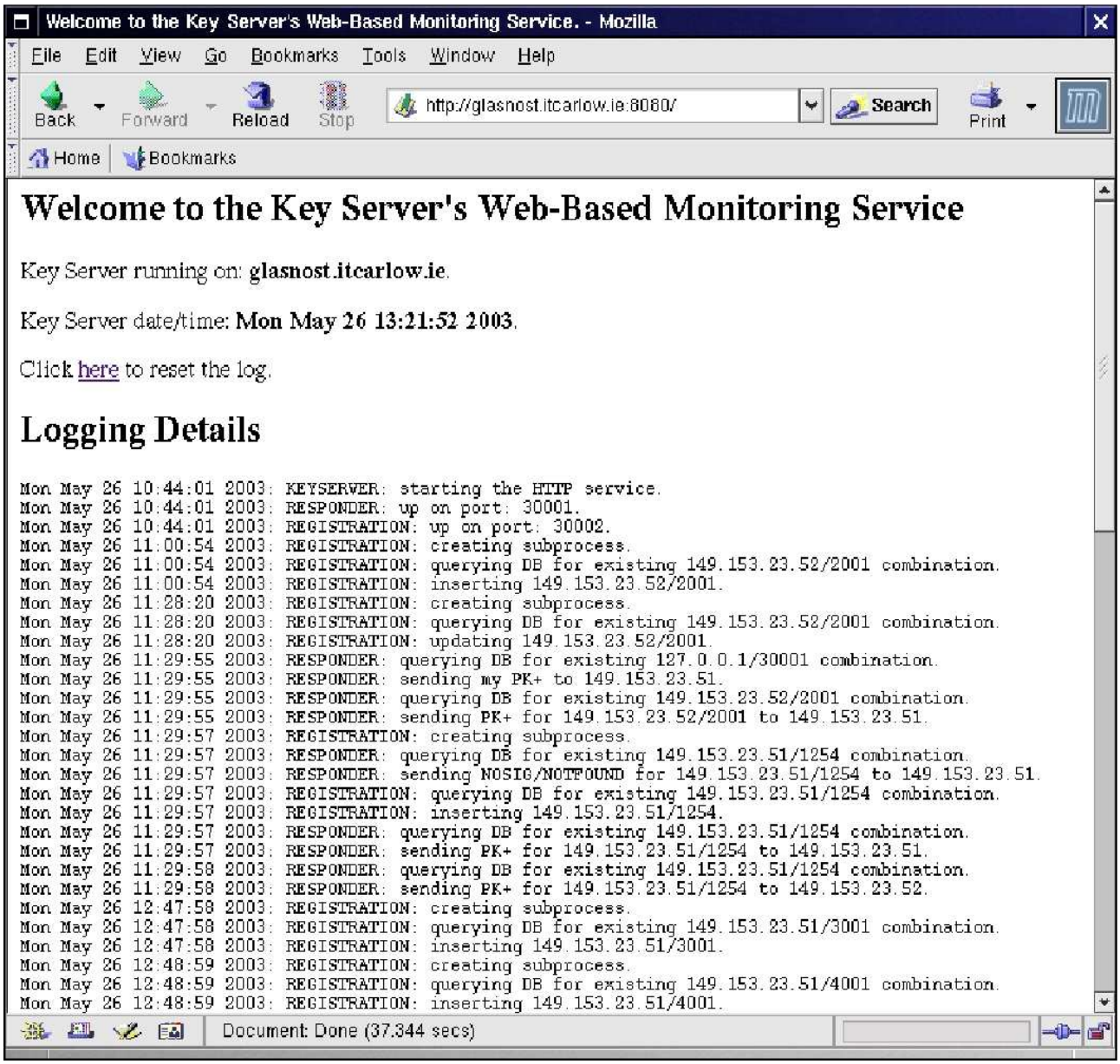
So, to start a Location running on protocol port 4444, with no web server, and debug status messages appearing, you would invoke “new” like this:

```
my $location = Mobile::Location->new( Port => 4444, Web => 0, Debug => 1 );
```

Be advised that due to security concerns you cannot run a Location on a well-known protocol port (that is ports with a value less that 1024). To do so requires “root access”, and running a Location as “root” is never a good idea. Just trust me on this.

Also, never run a Location on the same machine as your Key Server. A mobile agent could relocate to the Location and attempt to attack the Key Server (which would be intolerable, not to mention not very nice). If your Key Server stops running, no one can communicate. So, protect your Key Server by not running a Location on the same machine.

Even if you attempt to, be advised that the Key Server also runs a web server at protocol port 8080. It is not possible to run more than one service on any one particular protocol port. When contacted, the Key Server's web server should look something like this:



Creating Mobile Agents

Now for the fun part: creating mobile agents. This too is straightforward.

Before looking at an example mobile agent, let's review the “mobile agent writers rules”.

1. You must use the Mobile::Executive module at the start of your code.
2. The relocate command can only be invoked from within the main code of your mobile agent.
3. The relocate command cannot be invoked from within any construct that requires initialisation.
4. Only “my” variables automatically relocate, all else is left behind.

Obviously, without rule #1 things just won't work at all. Rule #2 means that it is not possible to call relocate from within a subroutine or object method. Rule #3 means that it is not possible to call relocate from within statements such as while, for, foreach, until and the like. However, it is possible to call relocate from within an if statement.

Here's a really simple mobile agent that starts off on one computer, prints a message, relocates to another computer, and prints the message again:

```
use Mobile::Executive;

my $message = “Hello from Scooby!”;

print “$message\n”;

relocate( 'some.other.computer.net', 3001 );

print “$message\n”;
```

Note that as the $message scalar was inititally declared as a “my” variable (rule #4), its value is available on the relocated-to computer. Also note that the call to relocate occurs within the mobile agents main code (rule #2), so relocation will work (assuming, of course, that a Location is running at protocol port 3001 on the computer identified by “some.other.computer.net”).

Simple, huh?

Scooby can successfully relocate scalars, arrays (of scalars), and hashes (of scalars), in addition to references to same. Simple objects can also relocate. At this time, more complex data structures cannot relocate. References to objects relocate poorly, so don't rely on them working as expected once relocation has occurred.

A Model for Scooby Mobile Agent Writers

I have found the following model to work well when writing mobile agents.

1. Decide what is it you want to do at each Location and write a subroutine to do it.
2. Ensure that the subroutine returns the results of any processing to its caller.
3. In the main program code, declare a data structure to hold the results.
4. Again, in the main program code, perform the relocations and remember the results produced by each Location in the data structure.
5. Once done, process the data structure that holds the results.

Let's assume for step #1 that we are interested in determining the local time reported by a collection of Locations. Here's a little subroutine, called do_it, that implements the required functionality, returning any processed results to its caller (step #2):

```
sub do_it {
    # This code is executed at each Location.

    return scalar localtime;
}
```

Let's create a hash, called %the_times, to hold the results (step #3):

```
my %the_times = ();
```

Step #4 performs a series of relocations and calls to do_it, like this:

```
relocate( 'pbmac.itcarlow.ie', 2001 );
$the_times{ 'pbmac.itcarlow.ie' } = do_it;
```

Note the assignment of the results from step #2 to the data structure. At the end of the mobile agent, process the data structure to display the processed results (step #5):

```
foreach my $host ( keys %the_times )
{
    print "It is $the_times{ $host } on $host.\n";
}
```

When put all together, we have the whattime.pl program, as follows:

```
#! /usr/bin/perl -w

#
# whattime.pl
#

use strict;
use Mobile::Executive;

my %the_times = ();

sub do_it {
    # This code is executed at each Location.

    return scalar localtime;
}

relocate( 'pbmac.itcarlow.ie', 2001 );
$the_times{ 'pbmac.itcarlow.ie' } = do_it;
```

```
relocate( 'pblinux.itcarlow.ie', 4001 );
$the_times{ 'pblinux.itcarlow.ie' } = do_it;

relocate( 'testimac.itcarlow.ie', 3001 );
$the_times{ 'testimac.itcarlow.ie' } = do_it;

relocate( 'pblinux.itcarlow.ie', 4001 );

foreach my $host ( keys %the_times )
{
    print "It is $the_times{ $host } on $host.\n";
}
```

Running the whattime.pl mobile agent is not too difficult. But first, let's check our syntax:

```
perl -c whattime.pl
whattime.pl syntax OK
```

Cool. Now let's run the mobile agent:

```
perl whattime.pl
```

After a little wait, something similar to the following should appear on screen:

```
It is Fri May 30 10:47:12 2003 on pblinux.itcarlow.ie.
It is Fri May 30 10:47:11 2003 on pbmac.itcarlow.ie.
It is Fri May 30 10:47:12 2003 on testimac.itcarlow.ie.
```

Whoops – the reported times are almost identical. What's going on? Well, the mobile agent has not relocated from the original computer, so the do_it subroutine is called three times on the same computer. To start the mobile agent on its way, run the whattime.pl program through the *Scooby* debugger, as follows:

```
perl -d:Scooby whattime.pl
```

This time the execution produces the results we were after:

```
It is Fri May 30 10:51:02 2003 on pblinux.itcarlow.ie.
It is Fri May 30 09:50:11 2003 on pbmac.itcarlow.ie.
It is Fri May 29 21:17:12 2003 on testimac.itcarlow.ie.
```

It looks like someone forgot to adjust the time on pbmac.itcarlow.ie for Daylight Savings, and the time on testimac.itcarlow.ie is well and truly out of sync with the other computers.

A Note on the Speed of the System

It can take quite a while to syntax-check and execute a mobile agent. This is due to the fact that the *Scooby* security mechanisms (based on RSA) generate a Public/Private Key Pairing at compile-time. If your computer is fast, this may only take a few seconds. If your computer is slow, this can take considerably longer (sometimes minutes). The testimac.itcarlow.ie computer, which is an original Green iMac (with only 32 meg of RAM) is running release 2.2 of YellowDogLinux. Generating keys on this platform takes several minutes.

Have fun with Scooby!