The On-line Documentation

This appendix presents the on-line documentation available with the facility. Once each of the modules/programs are installed, the on-line documentation is available as Linux `man` pages (or via the `perldoc` utility).

1. The Key Server On-line Documentation

NAME

keyserver - an RSA-based public keyserver for use with **Devel::Scooby** (which includes HTTP monitoring facility at port 8080).

VERSION

1.04

SYNOPSIS

Create a ".keyserverrc" configuration file (see FILES), set-up the required database (see ENVIRONMENT), then invoke the keyserver:

```
./keyserver
```

DESCRIPTION

This keyserver provides three services to clients that communicate with it.

1. The "Responder Service" runs on port **RESPONDER_PPORT** and listens for requests from clients. These take the form of an IP address in dotted-decimal notation, followed by a protocol port number. The IP address/port-number are looked-up in the SCOOBY.publics table (see ENVIRONMENT), and - if found - the associated public key is extracted from the table and signed using this keyserver's private key. Both the signature and the public key are then sent to the client.

If the lookup fails, the strings "NOSIG" followed by "NOTFOUND" are returned to the client.

If the IP address is LOCALHOST (which defaults to 127.0.0.1) and the protocol port number is RESPONDER_PPORT (which defaults to 30001), then this program returns the string "SELFSIG" followed by an UNSIGNED copy of this keyserver's public key. In this way, a client can retrieve the public key to use when verifying signatures.

2. The "Registration Service" runs on port **REGISTRATION_PPORT** and listens for connections from clients. When on arrives, it is immediately followed by a protocol port number, then a public key. This key is added to the SCOOBY.publics table (see ENVIRONMENT) together with the clients IP address in dotted-decimal notation and the protocol port number. For obvious reasons, the received public key is NOT signed by the client.

Note that changing the constant values for **REGISTRATION_PPORT** and **RESPONDER_PPORT** from their defaults will require source code changes to programs that interact with this keyserver (which includes the **Devel::Scooby**, **Mobile::Executive** and **Mobile::Location** modules). So, don't change these constant values unless you really have to.

3. The "HTTP-based Monitoring Service" runs on port HTTP_PORT (which defaults to 8080), and provides a mechanism to remotely check the status of the keyserver via the world-wide-web. The LOGFILE can be viewed and (optionally) reset via the web-based interface. Resetting the LOGFILE results in an archived copy of the LOGFILE-to-date being created on the keyserver's local storage.

ENVIRONMENT

It is assumed that the MySQL RDBMS is executing on the same machine as this keyserver. Here's a quick list of MySQL-specific instructions for creating a database and table required to support this program:

```
mysql -u root -p
```

```
mysql> create database SCOOBY;
mysql> use mysql;
mysql> grant all on SCOOBY.* to perlagent identified
            by 'passwordhere';
mysql> quit
```

```
mysql -u perlagent -p SCOOBY < create_publics.sql
```

If you use a different user-id/password combo to that shown above, be sure to change the two constants defined at the start of the source code (KEYDB_USER and KEYDB_PASS).

where the **create_publics.sql** disk-file contains:

```
create table publics
    (
        ip_address       varchar (16)  not null,
        protocol_port    varchar (6)   not null,
        public_key       text          not null
    )
```

FILES

A configuration file, called ".keyserverrc" needs to exist in the same directory as this keyserver. Its contents detail the IP address and protocol port numbers that connections will be allowed from. Typically, it will look something like this:

```
127.0.0.1:*
192.168.22.14:*
```

which allows any connection (on any port) from 127.0.0.1 and 192.168.22.14. Note that (at the moment), specifying a protocol port number in place of "*" has no effect. Connection from all ports on the specified IP address are allowed. This will change in a future release.

When first executed, this keyserver creates two disk-files:

```
"LOCALHOST.RESPONDER_PPORT.public", and
"LOCALHOST.RESPONDER_PPORT.private".
```

These contain this keyserver's RSA public and private keys, respectively. The public key is also added to the MySQL database.

DO NOT remove these files from the directory that runs this keyserver.

DO NOT edit these files, either.

The keyserver also logs all communication with it (in a disk-file called "keyserver.log" The contents of this log can be viewed (and archives of it created) using the "HTTP-based Monitoring Service" (see DESCRIPTION).

FOUR IMPORTANT CONSTANTS

Near the start of the keyserver's source code, four constants are defined as follows:

```
use constant KEYSRV_PASSWD       => 'keyserver';
use constant KEY_SIZE            => 1024;


use constant ENABLED_LOGGING     => 1;
use constant ENABLED_PRINTS      => 1;
```

Change the first two constants to values of your choosing to set the password (KEYSRV_PASSWD) and the key size (KEY_SIZE) to use during the PK+/PK- generation. Note: the larger the key size, the stronger the encryption, but, the slower this software will run. The default value for KEY_SIZE should suffice for most situations.

Set ENABLED_LOGGING to 0 switch off disk-based logging and the HTTP-based Monitoring Service.

Set ENABLED_PRINTS to 0 to disable the the display of status messages on STDOUT.

SEE ALSO

The **Devel::Scooby**, **Mobile::Executive** and **Mobile::Location** modules.

The following CPAN modules are assumed to be installed: **Net::MySQL** and **Crypt::RSA**.

The HTTP server requires **HTTP::Daemon** and **HTTP::Status**, which are installed as part of the **libwww-perl** library (also available on CPAN).

The Scooby Website: **http://glasnost.itcarlow.ie/~scooby/**.

AUTHOR

Paul Barry, Institute of Technology, Carlow,
paul.barry@itcarlow.ie,
http://glasnost.itcarlow.ie/~barryp/.

COPYRIGHT

2. The Executive.pm On-line Documentation

NAME

"Mobile::Executive" - used to signal the intention to relocate a Scooby mobile agent from the current Location to some other (possibly remote) Location.

VERSION

2.03 (version 1.0x never released).

SYNOPSIS

```
use Mobile::Executive;

...

relocate( $remote_location, $remote_port );
```

DESCRIPTION

Part of the Scooby mobile agent machinery, the **Mobile::Executive** module provides a means to signal the agents intention to relocate to another Location. Typical usage is as shown in the **SYNOPSIS** section above. Assuming an instance of **Mobile::Location** is executing on **$remote_location** at protocol port number **$remote_port**, the agent stops executing on the current Location, relocates to the remote Location, then continues to execute from the statement immediately AFTER the **relocate** statement.

Note: a functioning keyserver is required.

Overview

The only subroutine provided to programs that use this module is:

```
relocate
```

and it takes two parameters: a IP address (or name) of the remote Location, and the protocol port number that the Location is listening on.

Internal methods/subroutines

A Perl **BEGIN** block determines the absolute path to the mobile agents source code file, and puts it into the **$absolute_fn** scalar (which is automatically exported). This block also generates a PK+/PK- pairing (in **$public_key** and **$private_key**) and exports both values (as they are used by **Devel::Scooby**).

RULES FOR WRITING MOBILE AGENTS

There used to be loads, but now there is only one. Read the **Scooby Guide**, available on-line at: **http://glasnost.itcarlow.ie/~scooby/guide.html**.

SEE ALSO

The **Mobile::Location** class (for creating Locations), and **Devel::Scooby** (for running mobile agents).

The Scooby Website: **http://glasnost.itcarlow.ie/~scooby/**.

AUTHOR

Paul Barry, Institute of Technology, Carlow,
paul.barry@itcarlow.ie,
http://glasnost.itcarlow.ie/~barryp/.

COPYRIGHT

3. The Scooby.pm On-line Documentation

NAME

"Scooby" - the internal machinery that works with **Mobile::Location** and **Mobile::Executive** to provide a mobile agent execution and location environment for the Perl Programming Language.

VERSION

4.0x (versions 1.x and 2.x were never released; version 3.x did not support encryption and authentication).

SYNOPSIS

```
perl -d:Scooby mobile_agent
```

DESCRIPTION

This is an internal module that is not designed to be "used" directly by a program. Assuming a mobile agent called **multiwho** exists (that "uses" the **Mobile::Executive** module), this module can be used to execute it, as follows:

```
perl -d:Scooby multiwho
```

The **-d** switch to `perl` invokes Scooby as a debugger. Unlike a traditional debugger that expects to interact with a human, Scooby runs automatically. It NEVER interacts with a human, it interacts with the mobile agent machinery.

Scooby can be used to relocate Perl source code which contains the following:

SCALARs (both numbers and strings).

An ARRAY of SCALARs (known as a simple ARRAY).

A HASH of SCALARs (known as a simple HASH).

References to SCALARs.

References to a simple ARRAY.

References to a simple HASH.

Objects.

References to objects are **not** supported and are in no way guaranteed to behave the way you expect them to after relocation (even though they do relocate).

The relocation of more complex data structures is **not** supported at this time (refer to the TO DO LIST section, below).

Internal methods/subroutines

DB::DB - called for every executable statement contained in the mobile agent source code file.

DB::sub - called for every subroutine call contained in the mobile agent source code file.

_DB::storable_decode - takes the stringified output from **Storable**'s **thaw** subroutine and turns it back into Perl code (with a little help from the Data::Dumper module for objects).

DB::_check_modules_on_remote - checks to see if a list of modules/classes "used" within the mobile agent actually exist on the remote Location's Perl system.

DB::_get_store_pkplus - contacts the key server and requests a PK+, then stores the PK+ in a named disk-file.

DB::_wait_for_pkplus_confirm - repeatedly contacts the key server until requested PK+ is returned (i.e., ACKed).

ENVIRONMENT

This module must be installed in your Perl system's **Devel/** directory. This module will only work on an operating system that supports the Perl modules listed in the SEE ALSO section, below. (To date, I've only tested it on various Linux distributions).

TO DO LIST

Loads. The biggest item on the list would be to enhance Scooby to allow it to handle more complex data structures, such as ARRAYs of HASHes and HASHes of ARRAYs, etc., etc.

My initial plan was to allow for the automatic relocation of open disk-files. However, on reflection, I decided not to do this at this time, but may return to the idea at some stage in the future.

The current implementation checks to see if "used" classes are available on the next Location before attempting relocation, but does not check to see if "used" modules are available. It would be nice if it did.

It would also be nice to incorporate an updated **Class::Tom** (by James Duncan) to handle the relocation of objects to a Location without the need to have the module exist on the remote Location. On my system (Linux), the most recent **Class::Tom** generates compile/run-time errors.

SEE ALSO

The **Mobile::Executive** module and the **Mobile::Location** class. Internally, this module uses the following CPAN modules: **PadWalker** and **Storable**, in addition to the standard **Data::Dumper** module.

The **Crypt::RSA** modules provides encryption and authentication services.

The Scooby Website: **http://glasnost.itcarlow.ie/~scooby/**.

AUTHOR

Paul Barry, Institute of Technology, Carlow,
paul.barry@itcarlow.ie,
http://glasnost.itcarlow.ie/~barryp/.

COPYRIGHT

4. The Location.pm On-line Documentation

NAME

"Mobile::Location" - a class that provides for the creation of Scooby mobile agent environments (aka Location, Site or Place).

VERSION

4.0x (the v1.0x, v2.0x and v3.0x series were never released).

SYNOPSIS

```
use Mobile::Location;

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

$location->start_sequential;
```

or

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

SOME IMPORTANT NOTES FOR LOCATION WRITERS

1. Never, ever run a Location as 'root'. If you do, this module will die. Running as 'root' is a serious security risk, as a mobile agent is foreign code that you are trusting to execute in a non-threatening way on your computer. (Can you spell the word 'v', 'i', 'r', 'u', 's'?!?)

2. The **Mobile::Location** class executes mobile agents within a restricted environment. See the **Ops** argument to the **new** method, below, for more details.

3. Never, ever run a Location on the same machine that is acting as your keyserver (it's a really bad idea, so don't even think about it).

DESCRIPTION

Part of the Scooby mobile agent machinery, the **Mobile::Location** class provides a convenient abstraction of a mobile agent environment. Typical usage is as shown in the **SYNOPSIS** section above. This class allows for the creation of a passive, TCP-based mobile agent Location.

Overview

Simply create an object of type **Mobile::Location** with the **new** method. To start a sequential server, use the **start_sequential** method. To start a concurrent server, use the **start_concurrent** method.

Construction and initialization

Create a new instance of the **Mobile::Location** object by calling the **new** method:

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

Optional named parameters (with default values) are:

Debug (0) - set to 1 to receive STDERR status messages from the object.

Port (2001) - sets the protocol port number to accept connections on.

Log (0) - set to 1 to instruct the Location to log the received mobile agent to disk prior to performing any mutation. The name of the logged agent is "last_agent_PID.log", where PID is the process identifier of the Location. On sequential Locations, the PID is always the same value for each received agent. On concurrent Locations, the PID is the PID of the child process that services the relocation/re-execution, so it is always different for each received agent (so watch your disk space). It is often useful to switch this option on (by setting Log to 1) when debugging. Note that the received mobile agent persists on the Location's local disk storage.

Ops (") - add a list of Opcodes to the Opcode mask that is in effect when the mobile agent executes. Study the standard **Opcode** and **Ops** modules for details on Opcodes and how they are set. One way to secure your Location against attack is to ensure that the Opcodes in effect while a mobile agent executes are "safe". This is NOT an easy task, as protecting the mobile

agent environment from malicious mobile agents is never easy. Note that the default set of Opcodes in effect are enough to allow the relocation mechanism to execute. **NOTE**: if the mobile agent uses a operation not allowed by the Opcode mask, it is killed and stops executing. The Location continues to execute, and waits passively for the next mobile agent to arrive. The default set of enabled Opcodes is restrictive. Provide a space-delimited list of Opcodes to this argument to add to the list of allowed opcodes. NOTE: this functionality is currently **disabled** due to conflicts/incompatibilities with the current version of Crypt::RSA (version 1.50).

Web (1) - turns on the HTTP-based Monitoring Service running on port 8080 (HTTP_PORT), thus enabling remote monitoring of the Locations current status. It also logs interactions with this Location into 'location.log' (LOGFILE). Set to 0 to disable this behaviour.

Note that any received mobile agent executes in a directory called "Location", which will be created (if needs be) in the directory that houses this Location. Any "logs" are also created in the "Location" directory.

A constructor example is:

```
my $place = Mobile::Location->new( Port => 5555, Debug => 1 );
```

creates an object that will display all STDERR status messages, and use protocol port number 5555 for connections. Logging of received agents to disk is off. The standard Opcode mask is in effect. And logging to disk is on, as is the HTTP server.

When the Location is constructed with **new**, a second network service is created, running at protocol port number **Port+1**. In the example above, this second network service would run at protocol port number 5556. When sent the names of a set of Perl classes (e.g., Data::Dumper, HTTP::Request, Net::SNMP and the like), this service checks to see if the classes are available to the locally installed Perl installation. This allows **Devel::Scooby** to determine whether or not relocation is worthwhile prior to an attempted relocation. The **Devel::Scooby** module tries to determines the list of classes used by any mobile agent and communicates with this second network service "in the background". This all happens automatically, so the mobile agent programmer does not need to worry about it, as **Devel::Scooby** only complains when a module does not exist on a remote Location. That said, the administrator of the Location does need to be aware of this second network service. To confirm that the Location and the second network service are up-and-running use the **netstat -an** command-line utility (on Linux). The two "listening" services should appear in netstat's output.

Note: If a Location crashes (or is killed), the second network service can sometimes keeps running. After all, it is a separate process (albeit a child of the original). Trying to restart the Location results in an "bind to address failed" error message. Use the **ps -aux** command to identify the Perl interpreter that is executing and kill it with **kill -9 pid**, where **pid** is the process ID of the child process's Perl interpreter.

Class and object methods

start_concurrent

Start the location as a passive server, which operates concurrently. Once connected to a client, the server forks another process to receive and continue executing a mobile agent. This is the preferred method to use when there exists the potential to have an agent execute for a long period of time.

start_sequential

Start the location as a passive server, which operates sequentially. Once connected to a client, the server sequentially processes the receipt and continued executing of a mobile agent. This is OK if the agent is quick and not processor intensive. If the agent has the potential to execute for a long period of time, use the **start_concurrent** method instead. This may also be of use within environments that place a restriction on the use of **fork**.

Internal methods/subroutines

The following list of subroutines are used within the class to provide support services to the class methods. These subroutines should not be invoked through the object (and in some cases, cannot be invoked through the object).

_generate_label

Takes a filename and line number, then combines them with the current time to produce a random, unique label.

_check_for_modules

Given a list of module names, checks to see if the Location's Perl system has the module installed or not.

_spawn_network_service

Used by the **new** constructor to spawn the Port+1 network service which listens for a list of modules names from a mobile agent, then checks for their existence within the locally installed Perl system.

_service_client

Given a socket object (and the instances init data), service the relocation of a Scooby mobile agent.

_register_with_keyserver

Creates a PK+ and PK- value for the server, storing the PK+ in the keyserver, and the PK- in the object's state.

_logger and _logger2

Logs a message to the LOGFILE.

_build_index_dot_html

Builds the INDEX.HTML page for use by the HTTP-based Monitoring Service.

_build_clearlog_dot_html

Builds the CLEARLOG.HTML page for use by the HTTP-based Monitoring Service.

_start_web_service

Starts a small web server running at port 8080 (HTTP_PORT), and uses the two "_build_*" routines just described.

_spawn_web_monitoring_service

Creates a subprocess and starts the web server.

SEE ALSO

The **Mobile::Executive** module (for creating mobile agents), as well as **Devel::Scooby** (for running mobile agents).

The Scooby Website: **http://glasnost.itcarlow.ie/~scooby/**.

AUTHOR

Paul Barry, Institute of Technology, Carlow,
paul.barry@itcarlow.ie,
http://glasnost.itcarlow.ie/~barrypi/.

COPYRIGHT