Preparing Perl For Mobile Agents - Class::Tom

```
gunzip Class-Tom-2.04.tar.gz
tar xvf Class-Tom-2.04.tar
cd Class-Tom-2.04
perl Makefile.PL
make
make test
su
make install
<ctrl-D>
man Class::Tom
perl -e 'use Class::Tom'
```

Preparing Perl For Mobile Agents - Agent.pm

```
gunzip Agent-3.20.tar.gz
tar xvf Agent-3.20.tar
cd Agent-3.20
perl Makefile.PL
make
make test
su
make install
<ctrl-D>
man Agent
perl -e 'use Agent'
```

The Default Mobile Agent, 1 of 2

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

use strict;
use 5.6.0;

package Agent::OneDefault;

our @ISA = qw( Agent );

sub new {
    my ( $class, %args ) = @_;

    my $self = {};

    foreach ( keys( %args ) )
    {
        $self->{ "$_" } = $args{ "$_" };
    }

    bless $self, $class;
}
```

The Default Mobile Agent, 2 of 2

```
sub agent_main {
    my ( $self, @args ) = @_;
    my $to = delete( $self->{ 'Host' } );

    unless ( $to )
    {
        print "Hello from the onedefault.pa mobile agent.\n";

        return 1;
    }

    my $msg = new Agent::Message(
              'Body'      => [ "OneDefault.\n", $self->store() ],
              'Transport' => 'TCP',
              'Address'   => $to );

    if ( !$msg->send )
    {
        print "onedefault: could not send agent!\n";
    }
}
1;
```

A Launching Mobile Agent Environment, 1 of 2

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

use strict;
use Socket;
use Agent;

my $ma_name = shift || 'onedefault.pa';
my $ma_host = shift || 'localhost';
my $ma_port = shift || 40000;

$ma_host = inet_ntoa( scalar gethostbyname( $ma_host ) );

my $ma_hostport = $ma_host . ':' . $ma_port;
```

A Launching Mobile Agent Environment, 2 of 2

```
my %args = ( 'Name' => $ma_name,
             'Host' => $ma_hostport );

my $perlagent = new Agent ( %args );

my $agentresults = eval { $perlagent->run; };

if ( $@ )
{
    print "onelaunchma: something went wrong: $@\n";
}
if ( $agentresults )
{
    print "onelaunchma: results: ", $agentresults, "\n";
}
```

A One-Shot Location, 1 of 2

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

use strict;
use Socket;
use Sys::Hostname;

use Agent;
use Agent::Transport;

my $mae_name = inet_ntoa( scalar gethostbyname( hostname ) );
my $mae_port = shift || 40000;
my $mae_address = $mae_name . ':' . $mae_port;
```

A One-Shot Location, 2 of 2

```
my $mae = new Agent::Transport( 'Medium'  => 'TCP',
                                'Address' => $mae_address );

my ( $from, @recv_code ) = $mae->recv( 'Timeout' => 120 )
    or die "oneshotloc: timed out waiting: no code.\n";

my $stored_agent = join( '', @recv_code );

my $ma = new Agent( 'Stored' => $stored_agent )
    or die "oneshotloc: could not create agent.\n";

my $res = eval { $ma->run; };

if ( $@ )
{
    warn "oneshotloc: could not run agent: $@\n";
}
```

Testing the Mobile Agent Machinery

```
./oneshotloc 35000

./onelaunchma onedefault.pa pbmac.itcarlow.ie 35000
```

Processing Multiple Mobile Agents

```
while (1)
{
    my ( $from, @recv_code ) = $mae->recv( 'Timeout' => 120 );

    unless ( @recv_code )
    {
        warn "multishotloc: timed out waiting: no code.\n";
        next;
    }

    my $stored_agent = join( '', @recv_code );

    my $ma = new Agent( 'Stored' => $stored_agent );

    my $res = eval { $ma->run; };

    if ( $@ )
    {
        warn "multishotloc: could not run agent: $@\n";
    }
}
```

Identifying Multiple Locations - `multilaunchma.rc`

```
glasnost.itcarlow.ie:35000
149.153.103.5:40000
149.153.103.15:40000
pbmac.itcarlow.ie:35000
```

Identifying Multiple Locations, 1 of 3

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

use strict;
use Socket;

use Agent;

my $mae_name  = shift || 'multidefault.pa';

my %args = ( 'Name' => $mae_name );

open RCFILE, 'multilaunchma.rc'
    or die "multilaunchma: could not open RC file.\n";
```

Identifying Multiple Locations, 2 of 3

```
while ( <RCFILE> )
{
    my ( $mae_host, $mae_port ) = split /:/;

    chomp( $mae_port );

    $mae_host = inet_ntoa( scalar gethostbyname( $mae_host ) );

    push( @{$args{ 'Hosts' }}, ( $mae_host . ':' . $mae_port ) );
}

close RCFILE;
```

Identifying Multiple Locations, 3 of 3

```
my $perlagent = new Agent ( %args );

my $agentresults = eval { $perlagent->run; };

if ( $@ )
{
    print "multilaunchma: something went wrong: $@\n";
}
if ( $agentresults )
{
    print "multilaunchma: results: ", $agentresults, "\n";
}
```

A Multi-Location Mobile Agent, 1 of 3

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

use strict;
use 5.6.0;

package Agent::MultiDefault;

our @ISA = qw( Agent );

sub new {
    my ( $class, %args ) = @_;
    my $self = {};

    foreach ( keys( %args ) )
    {
        $self->{ "$_" } = $args{ "$_" };
    }

    bless $self, $class;
}
```

A Multi-Location Mobile Agent, 2 of 3

```
sub agent_main {
    my ( $self, @args ) = @_;
    my $to = shift @{ $self->{ 'Hosts' }};

    if ( $to )
    {
        $self->do_it;
    }
    unless ( $to )
    {
        return $self->at_end;
    }

    my $msg = new Agent::Message(
              'Body'      => [ "MultiDefault.\n", $self->store() ],
              'Transport' => 'TCP',
              'Address'   => $to );

    if ( !$msg->send )
    {
        print "multidefault: could not send agent!\n";
    }
}
```

A Multi-Location Mobile Agent, 3 of 3

```
sub do_it {
    my ( $self, @args ) = @_;

    print "The is do_it.\n";
}

sub at_end {
    my ( $self, @args ) = @_;

    print "The is at_end.\n";
}

1;
```

The Steve Purkis' agent_main Fix

```
sub agent_main {
    my ( $self, @args ) = @_;
    my $to = \$self->{ 'ToValue' };

    $self->do_it if ( $$to && ($#{$self->{ 'Hosts' }} > -1) );
    $self->{ 'ToValue' } = shift @{ $self->{ 'Hosts' }};
    return $self->at_end unless ( $$to );

    my $msg = new Agent::Message(
              'Body'      => [ "MultiDefault.\n", $self->store() ],
              'Transport' => 'TCP',
              'Address'   => $$to );

    if ( !$msg->send )
    {
        print "multidefault2: could not send agent!\n";
    }
}
```

The Mobile Agent `multiwho`

```
sub do_it {
    my ( $self, @args ) = @_;

    @{$self->{ 'WhoList' }} = ( @{$self->{ 'WhoList' }}, `who` );
}

sub at_end {
    my ( $self, @args ) = @_;

    print @{$self->{ 'WhoList' }}, "\n";
}
```

Running The Mobile Agent `multiwho`

```
./multilaunchma multiwhoma.pa

root      tty1     Apr 19 16:31
cno2031   pts/0    May 31 16:30 (pc310-10.itcarlow.ie)
cno2020   pts/1    May 31 16:14 (pc3-16.itcarlow.ie)
cno2020   pts/2    May 31 16:27 (pc3-16.itcarlow.ie)
cno2026   pts/3    May 31 16:18 (149.153.131.117)
cno2018   pts/4    May 31 16:30 (pc3-20.itcarlow.ie)
cno2019   pts/5    May 31 16:26 (pc3-21.itcarlow.ie)
COM2059   pts/8    May 31 15:20 (pc3-14.itcarlow.ie)
cno2006   pts/7    May 31 13:30 (pc3-13.itcarlow.ie)
com3027   pts/0    May  9 09:52 (pc2-2.itcarlow.ie)
meudecc   pts/0    May 31 16:30 (staff102.itcarlow.ie)
hickeypm  pts/1    May 31 16:14 (staff23.itcarlow.ie)
kinsella  pts/2    May 31 16:27 (akmac.itcarlow.ie)
whyte     pts/3    May 31 16:18 (149.153.100.117)
```

The Mobile Agent `ipdetermine`, 1 of 5

```
package Agent::IpDetermineMA;

use Net::SNMP;
use OIDs qw( sysDescr
             ipRouteTable ipRouteDest
             ipRouteType ipRouteNextHop );
```

The Mobile Agent `ipdetermine`, 2 of 5

```
sub do_it {
    my ( $self, @args ) = @_;

    my $snmp_session = Net::SNMP->session(
                           -hostname     => 'localhost',
                           -community    => 'public',
                           -version      => '2',
                           -port         => 161
                       );

    if ( !defined( $snmp_session ) )
    {
        print "ipdeterminema: error: cannot establish session.\n";

        return;
    }
```

The Mobile Agent ipdetermine, 3 of 5

```
my $responsePDU = $snmp_session->get_table( ipRouteTable );

if ( !defined( $responsePDU ) )
{
    print "ipdeterminema: OID: ", ipRouteTable, " : ",
         $snmp_session->error, "\n";

    return;
}

$snmp_session->close;

my @dests = ();

foreach my $resp ( keys %{ $responsePDU } )
{
    my $oid_pat = ipRouteDest;
    if ( $resp =~ /^$oid_pat\./ )
    {
        @dests = ( @dests, $responsePDU->{ $resp } );
    }
}
```

The Mobile Agent ipdetermine, 4 of 5

```
    my %unique_next_hops = ();

    foreach my $dest ( @dests )
    {
        my $oid_type = ipRouteType . ".$dest";

        if ( $responsePDU->{ $oid_type } eq '4' )
        {
            my $oid_nexthop = ipRouteNextHop . ".$dest";
            $unique_next_hops{ $responsePDU->{ $oid_nexthop } }++;
        }
    }

    foreach my $unique ( keys %unique_next_hops )
    {
        push @{$self->{ 'Routers' }}, $unique;
    }
}
```

The Mobile Agent ipdetermine, 5 of 5

```
sub at_end {
    my ( $self, @args ) = @_;

    print "Determined 'ip' next hop data:\n\n";

    foreach my $router ( @{$self->{ 'Routers' }} )
    {
        print "$router\n";
    }
}
```

Running The Mobile Agent `ipdetermine`

```
./ipdetermine glasnost.itcarlow.ie public

   ...

Requesting 'ip' next hop data for: 149.153.100.67, public, SNMPv2

Next hop: 149.153.100.10

   ...

glasnost.itcarlow.ie:40000
pbmac.itcarlow.ie:44444

   ...

./multilaunchma ipdeterminema.pa

   ...

Determined 'ip' next hop data:

149.153.100.10
```

The Cloning Mobile Agent `ipdetermine`, 1 of 9

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

use strict;
use 5.6.0;

package Agent::IpDetermineCloneMA;

use Socket;
use Net::SNMP;
use OIDs qw( sysDescr
             ipRouteTable ipRouteDest
             ipRouteType ipRouteNextHop );

use constant SENDTO_UDP_PORT   => 40001;
use constant REMOTE_HOST       => 'localhost';
use constant RESULTS_RECV_PORT => 40001;
use constant MAX_RECV_LEN      => 65536;
```

The Cloning Mobile Agent `ipdetermine`, 2 of 9

```
our @ISA = qw( Agent );

sub new {
    my ( $class, %args ) = @_;
    my $self = {};

    foreach ( keys( %args ) )
    {
        $self->{ "$_" } = $args{ "$_" };
    }

    bless $self, $class;
}
```

The Cloning Mobile Agent `ipdetermine`, 3 of 9

```
sub agent_main {
    my ( $self, @args ) = @_;

    my $to = \$self->{ 'ToValue' };

    $self->do_it if ( $$to && ($#{$self->{ 'Hosts' }} > -1) );

    $self->{ 'ToValue' } = shift @{ $self->{ 'Hosts' }};

    return $self->at_end unless $$to;

    my $msg = new Agent::Message(
                  'Body'      => [ "IpDetermineCloneMA.\n",
                                   $self->store() ],
                  'Transport' => 'TCP',
                  'Address'   => $$to );

    if ( !$msg->send )
    {
        print "ipclonema: could not send agent!\n";
    }
```

The Cloning Mobile Agent ipdetermine, 4 of 9

```
else
{
    my $local_port = RESULTS_RECV_PORT;
    my $trans_serv = getprotobyname( 'udp' );
    my $local_addr = sockaddr_in( $local_port, INADDR_ANY );

    socket( MARECV_UDP_SOCK, PF_INET, SOCK_DGRAM, $trans_serv )
        or die "ipclonema: socket creation failed: $!\n";
    bind( MARECV_UDP_SOCK, $local_addr )
        or die "ipclonema: bind to address failed: $!\n";

    my $data;

    my $from_who = recv( MARECV_UDP_SOCK, $data,
                                    MAX_RECV_LEN, 0 );

    close MARECV_UDP_SOCK;
```

The Cloning Mobile Agent ipdetermine, 5 of 9

```
        if ( $from_who )
        {
            print "Determined 'ip' next hop data:\n\n";
            print "$data\n";
        }
        else
        {
            warn "ipclonema: Problem with recv: $!\n";
        }
    }

    return undef;
}
```

The Cloning Mobile Agent ipdetermine, 6 of 9

```
sub do_it {
    my ( $self, @args ) = @_;
}

sub at_end {
    my ( $self, @args ) = @_;

    my $snmp_session = Net::SNMP->session(
                           -hostname     => 'localhost',
                           -community    => 'public',
                           -version      => '2',
                           -port         => 161
                       );

    if ( !defined( $snmp_session ) )
    {
        print "ipclonema: error: cannot establish session.\n";

        return;
    }
```

The Cloning Mobile Agent `ipdetermine`, 7 of 9

```
my $responsePDU = $snmp_session->get_table( OIDs::ipRouteTable );

if ( !defined( $responsePDU ) )
{
    print "ipclonema: OID: ", OIDs::ipRouteTable, " : ",
         $snmp_session->error, "\n";

    return;
}

$snmp_session->close;

my @dests = ();

foreach my $resp ( keys %{ $responsePDU } )
{
    my $oid_pat = OIDs::ipRouteDest;
    if ( $resp =~ /^$oid_pat\./ )
    {
        @dests = ( @dests, $responsePDU->{ $resp } );
    }
}
```

The Cloning Mobile Agent `ipdetermine`, 8 of 9

```
my %unique_next_hops = ();

foreach my $dest ( @dests )
{
    my $oid_type = OIDs::ipRouteType . ".$dest";

    if ( $responsePDU->{ $oid_type } eq '4' )
    {
        my $oid_nexthop = OIDs::ipRouteNextHop . ".$dest";
        $unique_next_hops{ $responsePDU->{ $oid_nexthop } }++;
    }
}

foreach my $unique ( keys %unique_next_hops )
{
    push @{$self->{ 'Routers' }}, $unique;
}
```

The Cloning Mobile Agent ipdetermine, 9 of 9

```
    my $remote = REMOTE_HOST;
    my $remote_port = SENDTO_UDP_PORT;
    my $trans_serv  = getprotobyname( 'udp' );
    my $remote_host = gethostbyname( $remote )
        or die "ipclonema: name lookup failed: $remote\n";
    my $destination = sockaddr_in( $remote_port, $remote_host );

    sleep( 1 );

    socket( MASEND_UDP_SOCK, PF_INET, SOCK_DGRAM, $trans_serv )
        or die "ipclonema: socket creation failed: $!\n";

    my $data = join( '', @{$self->{ 'Routers' }} );

    send( MASEND_UDP_SOCK, $data, 0, $destination )
        or warn "ipclonema: send to socket failed.\n";
    close MASEND_UDP_SOCK
        or die "ipclonema: close socket failed: $!\n";
}

1;
```

Changing `multilaunchma` To Support Cloning Creating `multilaunchma`

```
my $initiating = inet_ntoa( scalar gethostbyname( hostname ) );

my %args = ( 'Name' => $mae_name,
             'Home' => $initiating );
```

Running The Cloning Mobile Agent

```
./multilaunchma2 ipdetermineclonema.pa

   ...

Determined 'ip' next hop data:

149.153.100.10
```