Logtrend’s agent developper’s guide

LogTrend

Logtrend’s agent developper’s guide

by LogTrend

Revision History

Revision 1.0July, 2001

First Docbook version

Table of Contents

Chapter 1. Introduction

Well, let’s introduce the good way how to create a new agent from the generic agent available in the LogTrend’s software. First be sure to have learnt a really strange language called Perl ;-).

There are two steps in creating a new agent :

- building a new agent’s class that inherits from the generic agent’s class.
- building a perl script that will be used to instantiate the newly created class.

We will now see in details all the work that must be done to create a new agent. For the needs of this explanations, we will create an agent called SimpleAgent that will collect three datas and could generate an alarm.

Chapter 2. Creation of an agent's class that inherits from the generic agent's class

Every LogTrend agent inherit from the LogTrend::Agent perl module. This module provides needed features to read XML description file ('Generic' section), to send data, to schedul actions.

2.1. First lines of code

First create a file named `SimpleAgent.pm` and copy the following code into it :

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

package LogTrend::Agent::SimpleAgent;

use strict;
use vars qw( @ISA );

use LogTrend::Agent;
use LogTrend::Common::LogDie;

@ISA = ("LogTrend::Agent");

sub new
{
   my ( $classname ) = @_;

   my $self = $classname->SUPER::new( $name, $version );
   bless($self, $classname);
   srand( time() );
   return $self;
}

sub ParseXMLConfigFile
{
   my ($self,$file,$agentstate) = @_;
   $self->SUPER::ParseXMLConfigFile( $file );

   ## Agent-specific configuration parameters :
   ## use $self->{FOO} to stock information ( all $self->{_FOO} are reserved )

   ## Tag 'Configuration'
   my ($list,$node,$valueref,$attributes,$attrnode);
   my $parser = new XML::DOM::Parser() || Die($!);
   my $doc = $parser->parsefile( $file ) || Die($!);

   my $rootlist = $doc->getElementsByTagName("Configuration") ||
                                                Die("No \"Configuration\" tag.");
```

```
   my $rootnode = $rootlist->item(0) || Die("No \"Configuration\" tag.");

   ## Tag 'Specific'
   $rootlist = $rootnode->getElementsByTagName("Specific") ||
                                        Die("No \"Specific\" tag.");
   $rootnode = $rootlist->item(0) || Die("No \"Specific\" tag.");

   ## Tag 'Yellow'
   $list = $rootnode->getElementsByTagName("Yellow") || Die("$file: No \"Yellow\" tag.");
   $node = $list->item(0) || Die("No \"Yellow\" tag.");
   $attributes = $node->getAttributes() || Die("$file: Error in \"Yellow\" tag.");

   $attrnode = $attributes->getNamedItem("Limit") ||
                  Die("$file: No 'Limit' field in \"Yellow\" tag.");
   $self->{"Cfg_Limit"} = $attrnode->getValue();

   $attrnode = $attributes->getNamedItem("Interval") ||
             Die("$file: No 'Interval' field in \"Yellow\" tag.");
   $self->{"Cfg_Interval"} = $attrnode->getValue();

   $parser = undef;
}

sub CreateAgentDescription
{
   my $self = shift;
   my ($d,$a) = (1,1);

   $self->AddADataDescription(  $d++, "Integer", "bytes", "blue foo" );
   $self->AddADataDescription(  $d++, "Real",    "none",  "red foo",    "red_foo");
   $self->AddADataDescription(  $d++, "Text",    "none",  "green foo" );
   $self->AddAnAlarmDescription($a++, "Warning",          "yellow foo", "yellow_foo");
}

sub CollectData
{
   my $self = shift;

   $self->AddDataInteger( $self->{"blue foo"}, 333 );
   $self->AddDataReal(    $self->{"red_foo"}, 4.5 );
   $self->AddDataText(    $self->{"green foo"}, "foo" );

   if( rand($self->{"Cfg_Interval"}) < $self->{"Cfg_Limit"} )
   {
      $self->AddAlarm( $self->{"yellow_foo_Warning"} );
   }
}
```

2.2. Explanations

2.2.1. Starting

To define the new agent you must create a new package of the new agent's class with the code :

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

package LogTrend::Agent::SimpleAgent;

use strict;

use vars qw( @ISA );

use LogTrend::Agent;
use LogTrend::Common::LogDie;

@ISA = ("LogTrend::Agent");
```

After that, you will have to redefine the generic agent's constructor that will be used during the instantiation of a new agent. This will be done with the following code :

```
sub new
  {
   my ( $classname ) = @_;

   my $self = $classname->SUPER::new( $name, $version );
   bless($self, $classname);
   srand( time() );
   return $self;
}
```

Note: the redefinition of the constructor is mandatory in order to allow polymorphism to work properly. So don't forget it.

2.2.2. XML parsing

You could define a `ParseXMLConfigFile` method for your agent, if you have to take information for the XML file ('Specific' tag). Your method must call the Agent's one and do its work with the file. In the case of you dont need to parse the XML, you can defined no such method, and then only the Agent's one will ce called.

Here's an example of the `ParseXMLConfigFile` redefinition for our SimpleAgent example :

```
sub ParseXMLConfigFile
{
```

```
    my ($self,$file,$agentstate) = @_;
    $self->SUPER::ParseXMLConfigFile( $file );

    ## Agent-specific configuration parameters :
    ## use $self->{FOO} to stock information ( all $self->{_FOO} are reserved )

    ## Tag 'Configuration'
    my ($list,$node,$valueref,$attributes,$attrnode);
    my $parser = new XML::DOM::Parser() || Die($!);
    my $doc = $parser->parsefile( $file ) || Die($!);

    my $rootlist = $doc->getElementsByTagName("Configuration") ||
                                                    Die("No \"Configuration\" tag.");
    my $rootnode = $rootlist->item(0) || Die("No \"Configuration\" tag.");

    ## Tag 'Specific'
    $rootlist = $rootnode->getElementsByTagName("Specific") ||
                                          Die("No \"Specific\" tag.");
    $rootnode = $rootlist->item(0) || Die("No \"Specific\" tag.");

    ## Tag 'Yellow'
    $list = $rootnode->getElementsByTagName("Yellow") || Die("$file: No \"Yellow\" tag.");
    $node = $list->item(0) || Die("No \"Yellow\" tag.");
    $attributes = $node->getAttributes() || Die("$file: Error in \"Yellow\" tag.");

    $attrnode = $attributes->getNamedItem("Limit") ||
                    Die("$file: No 'Limit' field in \"Yellow\" tag.");
    $self->{"Cfg_Limit"} = $attrnode->getValue();

    $attrnode = $attributes->getNamedItem("Interval") ||
              Die("$file: No 'Interval' field in \"Yellow\" tag.");
    $self->{"Cfg_Interval"} = $attrnode->getValue();

    $parser = undef;
}
```

2.2.3. Creating the agent's description

The next step is the redefinition of the method that is used to create the agent's description. In this method, you can use all the code that you need but there is only two methods of the generic agent's class that you can call :

- `AddADataDescription` : to add a data description. Parameters are :
  - a number which is a unique key that is identifying the data,
  - the data type which can be one of the following values :
    - Integer

  - Real
  - Text
  - Date
  - Time
  - DateTime
- an information text about the data,
- the identier optional of the data (by default, the information text will be the identifier).

- `AddAnAlarmDescription` : to add an alarm description. Parameters are :
  - a number which is a unique key that is identifying the alarm,
  - the alarm level which can be one of the following values :
    - Info
    - Warning
    - Error
  - an information text about the alarm,
  - the identier optional of the alarm (by default, the information text will be the identifier).

Here's an example of the `CreateAgentDescription` redefinition for our SimpleAgent example :

```
sub CreateAgentDescription
{
   my $self = shift;
   my ($d,$a) = (1,1);

   $self->AddADataDescription(  $d++, "Integer", "bytes", "blue foo" );
   $self->AddADataDescription(  $d++, "Real",    "none",  "red foo",    "red_foo");
   $self->AddADataDescription(  $d++, "Text",    "none",  "green foo" );
   $self->AddAnAlarmDescription($a++, "Warning",          "yellow foo", "yellow_foo");
}
```

In our example, the creation of the agent's description is a bit static but it can also be dynamic if you need it. A good example can be found with an agent that would have to collect datas about partitions on a Linux Box. In this case, you can not presume of the number of partitions available, so you will have to build the description dynamically.

> **Note:** the redefinition of the `CreateAgentDescription` method is mandatory in order to make the agent work properly.

2.2.4. Collecting data

The last thing to do is to redefine the method used to collect datas. In this method, you can use all the code that you need but there is only two types of methods of the generic agent's class that you can call :

- the methods used to add datas which are :
  - `AddDataText`
  - `AddDataInteger`
  - `AddDataReal`
  - `AddDataDate`
  - `AddDataTime`
  - `AddDataDateTime`

  The parameters of this methods are :

  - a number that is identifying the data ; this number is $self->{"identifier"} where identifier is the identifier of the data,
  - the data value itself.
- the method used to add an alarm called `AddAlarm`.

  The parameter of this method is the number that is identifying the data ; this number is $self->{"identifier"} where identifier is the identifier of the data.

Now let's have a look at our example for the redefinition of the `CollectData` method :

```
sub CollectData
{
   my $self = shift;

   $self->AddDataInteger( $self->{"blue foo"}, 333 );
   $self->AddDataReal(    $self->{"red_foo"}, 4.5 );
   $self->AddDataText(    $self->{"green foo"}, "foo" );

   if( rand($self->{"Cfg_Interval"}) < $self->{"Cfg_Limit"} )
   {
      $self->AddAlarm( $self->{"yellow_foo_Warning"} );
   }
}
```

Our example just set three foo data, the first beeing an integer, the second a real and the last one a text. Data values are collected using the `AddData*` methods from DataAlarmsSet. We also can see an example of setting an alarm : if the value is greater than the threshold, we set the Alarm using the `AddAlarm`

> **Note:** the redefinition of the `CollectData` method is mandatory in order to make the agent work properly.

2.3. Summary

In summary, the steps are :

1. create a new package for the new agent's class inherited from the generic agent's class.
2. redefine the constructor,
3. optionaly redefine the `ParseXMLConfigFile` method,
4. redefine the `CreateAgentDescription` method,
5. redefine the `CollectData` method.

2.4. Running actions on alarms

You can easily run actions on agent's alarm with the package `LogTrend::Action::ActionSet`. While parsing XML, you can create an ActionSet with :

```
my $actionset LogTrend::Action::ActionSet->new($alarm_node)
```

Where `$alarm_node` is the XML::DOM::Node of the alarm configuration tag. This methods parse *Action* tag described in agents installation guide.

The last code lines to added for alarm's actions support, is :

```
$actionset->Run($alarm_level);
```

This line should be add near `$self->AddAlarm` when the agent add an alarm.

Chapter 3. Creation of a Perl script used to instantiate an agent's class

Now, we will interest us to the creation of a Perl script that will be used to instantiate the new agent.

We will now look at the things we have to do to create a Perl instantiation file of our SimpleAgent. In fact there's only two things really important : firstly the instantiation of the agent and secondly the call of the `Run` method on this newly created agent's object. But in order to allow the user of the agent to configure easily how the agent is working we have to supply a configuration file and all the mechanisms needed to read and interpret it. We also have to manage the command line parameters.

Here's an example of a configuration file, which is in xml format (DTD is available in appendix):

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE Configuration SYSTEM "Configuration.dtd">
<Configuration>
   <Generic>
      <AgentDescriptionFile>simpleagentdescription.xml</AgentDescriptionFile>
      <Source>3</Source>
      <Agent Number="4" Version="5" /> <!-- Used for -d mode only -->
      <Time Between_Collections="5m"
            Between_Deliveries="10m"
            Before_Warn_If_Serveur_Not_Responding="1h" />
      <DataFuture>
         <!-- Make you choice : -->
         <!-- <Save FileName="linuxagentcache.xml" /> -->
         <!-- <Send Host="serveur" Port="9999" Password="spa"
MailForBridge="mailbridge@mydomain.com" /> -->
         <Send Host="spa" Port="9999" Password="spa" />
      </DataFuture>
   </Generic>
   <Specific>

   <!-- Insert agent-specific configuration parameters here : -->
   <Yellow Limit="1" Interval="6" />

   </Specific>
</Configuration>
```