xCAT 2 Developer's Guide

01/07/10, 10:39:11 AM

Table of Contents

1.0 Introduction

The Developer's Guide intends to give you enough information that you can start writing code for the xCAT project. It's intent is to be a starting point but by no means will provide all the information you need because each new function developed has it's own problems to solve.

Before starting to write code for xCAT, you should post your intentions to the xcat mailing list . First this will allow the xCAT architects to evaluate the function to see if it fits into the future plans of xCAT; and also to determine, if the function is already in plan and being developed by someone else.

To contribute code: To **contribute code**, please create a SourceFourge account, and send your id to xcat-user@lists.xcat.org.

Go to the xCAT SourceForge wiki. Read and follow the instructions in the contribution guidelines.

You may also check the xCAT list of requested features and submit your request there.

2.0 xCAT Architecture

The heart of the xCAT architecture is the xCAT daemon (xcatd) on the management node. This receives requests from the client, validates the requests, and then invokes the operation. Function developed for xCAT should be designed to operate in this environment.
You need to review and understand the xCAT Architecture before developing your commands.

3.0 Developing xCAT Code

Code written for xCAT should be in Perl except under the condition that the code will run in environments where Perl may not be installed (e.g compute nodes), or Perl is not appropriate for the function being developed. Non-use of Perl should be reviewed by the architecture committee. With the many OS's and architectures that xCAT must support, when a language like C or C++ is used, it causes additional packaging work. More rules for code development are listed at contribution guidelines.

Recommended good programming practices should be followed as outlined in our Programming Tips page.

1.1 Client/Server Model

Review the Client/Server flow in the xCAT Architecture. When developing your command, there will be two major parts to the command. The Client Code and the Server Code (plugin).

We have provided a very simple example **xCATWorld** client code and the **xCATWorld.pm** plugin in the release.

1.1

1.1.1 Client code

First you will develop the client front-end. Many commands can use the two available client front-ends provided (xcatclient or xcatclientnnr). The command can either be a sym link to xcatclient/xcatclientnnr or a thin wrapper that calls xCAT::Client::submit_request() .

The xcatclient client front end supports commands that will provide a noderange and flags to the new function. The xcatclientnrr client front end supports commands that do not need a noderange. If your command has a more complex interface than is supported by these two routines, you can write your own client front-end. You can go to the /opt/xcat/bin directory to check which commands are links and which wrote their own clients.

1.1.2 Server code (plugins)

Next you will develop your plugin that will support processing the request from your client front-end. Review the xCATd Plugins flow in the xCAT Architecture. It tells you how xcatd processes the plugins and is very important for your design. All xCAT plugins are installed in /opt/xcat/lib/perl/xCAT_plugin.

Each Plugin is divided into three major sections:

1. handled_commands() - returns list of command(s) handled by this plugin. If commands are related we tend to put them in one plugin.
2. preprocess_request() - Needed if your command supports hierarchy, that is the plugin will be run on a service node to process the compute node.
3. process_request () - This is where the function of you command is placed.

Implementation note for handled_commands:

For many of the xCAT commands, the same command can be implemented in different plugins and the correct plugin is invoked based on a table value. For example, the rpower command is implemented in the hmc.pm for pSeries nodes, in the blade.pm for blades, and so on, and the correct plugin is invoked based on the value of nodehm:power and nodehm:mgt attributes for that node. This is coded in the plugin as follows:

```
sub handled_commands {
  return {
    findme => 'blade',
    getmacs => 'nodehm:getmac,mgt',
    rscan => 'nodehm:mgt',
    rpower => 'nodehm:power,mgt',
    getbladecons => 'blade',
    getrvidparms => 'nodehm:mgt',
    rvitals => 'nodehm:mgt',
    rinv => 'nodehm:mgt',
    rbeacon => 'nodehm:mgt',
    rspreset => 'nodehm:mgt',
    rspconfig => 'nodehm:mgt',
    rbootseq => 'nodehm:mgt',
    reventlog => 'nodehm:mgt',
    switchblade => 'nodehm:mgt',
  };
}
```

When the xCAT daemon loads all of the plugins, it builds an internal table of handled commands. When the same command is listed in more than one plugin, the value for the last plugin that is loaded is the one that the daemon will use. Therefore, ALL plugins must code the identical value for a table-driven command. For example, all plugins that implement the rpower command MUST code the handled_commands entry as

```
rpower => 'nodehm:power,mgt',
```

If you are coding one of these database-driven commands, you should search all existing plugins for entries that match the command you are coding and use the same value.

1.1.1 Debugging Commands (XCATBYPASS mode)

Debugging you new client/server command through the xcatd can be difficult, so we have put in place a debug mode that bypasses the daemon. This allows you to run the perl debugger from the client invocation through the plugin.

If the **XCATBYPASS** environment variable is set, the connection to the server/daemon will be bypassed and the plugin will be called directly by Client.pm. If it is set to a directory, all perl modules in that directory will be loaded in as plugins. This allows you to add or change new xCAT perl module libraries for test without disrupting other users on the system. If it is set to any other value (e.g. "yes", "default", whatever string you want) the default plugin directory will be used.

1.2 Hierarchy

Commands that are going to be run on the compute nodes, need to support hierarchy, because xCAT can be configured for hierarchy. This means the preprocess_request function in the plugin for the command must be provided. A simple example, is in the **xCATWorld.pm** plugin.

1.1

1.1 Calling Plugins from other Plugins

If you xCAT client/server command needs to call another xCAT client/server command from your plugin (e.g. Xdsh), you are in the situation that your plugin must call the other commands plugin. You cannot succesfully just call the other command client from your plugin.

There is a special interface defined in Utils.pm (runxcmd) for doing this. If it does not support the returns you need from the command you can write your own.
See the General Utilitiesprovided by xCAT.

1.2 Remote Commands (ssh, rsh)

We desire that all the xCAT commands to be using the same remote shell method to the nodes. The default is ssh on Linux and on AIX, it is determined by the site table attribute "useSSHonAIX". If this attribute is set to "no" then rsh will be used on AIX. IF set to "yes", ssh is used on AIX. We need the attribute on AIX, because OpenSSH does not come with the AIX operating system and must be installed.

If you are developing commands that will use remote shell, you may want to discuss this with the xCAT architects. One way to always use the appropriate remote shell is to not call ssh or rsh (scp/rcp) directly but to use xdsh and xdcp- api or plugin (for hierarchy support) which will check the remote shell setup for xCAT and use the appropriate remote shell. It is not recommended to call the xdsh or xdcp command line interface from another xCAT command.

2.0 Common Perl Libraries for xCAT code

2.1 General Utilities

There are a many of xCAT Perl libraries available that contain utility functions that we have found useful to share across the xCAT code. Here are some of the more useful ones. You should look in the xCAT Perl library (/opt/xcat/lib/perl/xcat) when you need function to see if it already exists.

- Utils.pm – Contains a large number of Utilities. Some of the more commonly used are the following.
  a) isLinux
  b) isAIX
  c) isMN
  d) isServiceNode
  e) Version
  f) runcmd
  g) runxcmd
- DbobjUtils.pm – A set of Utilities that handle xCAT data objects at a more abstract level. They are used by commands like lsdef which will access many database tables to return all the information for a node.
- NodeRange.pm – Routines to figure out the list of node based on an input group or noderange.
- MsgUtils.pm – Contains the messaging, logging interface for xCAT. All message should use these utilities. This ensure that the message do work in a client/server and hierarchical architecture. It also records messages to syslog appopriately for xCAT. You can see many examples in the existing code.
- Table.pm – all the xCAT database access routines. These routines use a Perl DBI to access the database that xCAT is currently running. All accesses to the xCAT database should be through one of these routines. This ensure sthat your code will support all the databases supported by xCAT (e.g. Sqlite, mysql,postgresql, etc), and be unaware of what database we are using. If new routines are needed, the need should be submitted as a feature request. Changes to the existing routines must be done by the xCAT core development team to ensure existing code is not broken.
- Client.pm – this routine is the primary interface from your command to the xcatd daemon. No changes should be make without careful review with the xCAT architects.
- Schema.pm - Database Schema. Any change or addition requests to that Schema must be submitted to the xCAT architects. Currently in plan, is a being designed a way for an individual to extend the Schema and not affect the basic schema that everyone uses.

1.0 Adding Postscripts

Scripts that run on the installed nodes after installation, for further node setup, can be added to xCAT. When they are added to the /svn/xcat/xcat-core/trunk/xCAT/postscripts directory in SVN, they are automatically copied to /install/postscripts during the xCAT Management Node install. This makes them available to be added to the xCAT postscripts table to be run post node installation. Normally we do not write these script in Perl but in shell, so as to not require Perl to be installed on the compute nodes.

2.0 Documentation

All new command must be documented:

1. There must be a man page. See Creating ManPages.
2. There must be a link added in xCAT2top doc. This will be done by the xCAT development team, but they must be made aware of new function. Posting to the xcat forum is a good method.
3. The code must be documented. See contribution guidelines for coding standards.

1.0 Creating Manpages

XCAT automatically creates manpages out of pods during the build process. To create a manpage for a new command a pod must be created and checked into SVN. The pods are located at /svn/xcat/xcat-core/trunk/xCAT-client/pods. You can get good examples of pods to copy there. Most of commands manpages go in the man1 directory, but if you have any questions, contact the xCAT architects.

2.0 Packaging new Code

In most cases it will be obvious where your client code will be checked into SVN (the xCAT-client path) and the plugin code (the xCAT-Server path) , but it would be good to review with the xCAT architects any new code that will be packaged with the xCAT code. For the most part, putting the code in the appropriate directory in SVN will automatically have it packaged with xCAt when it is built.

3.0 Changes to xCAT *.spec files

All changes to the xCAT *.spec files should be done by the core xCAT team. Submit any change requests through a feature or directly to the team on the xCAT mailing list.

4.0 Setting up your SVN development Environment

SVN is a open source sw that you can download from http://subversion.tigris.org/project_packages.html#binary-packages. It supports a variety of OS including RedHat Linux, Windows and IBM OS/400. The documentation can be found here: http://svnbook.red-bean.com/nightly/en/svn-book.html

The concept of SVN is you extract all the files under a directory to your workspace by *svn checkout* command. After modifying files, you then use *svn commit* command to merger you changes back into the repository. Multiple people can checkout (copy) the same files at the same time.

xCAT 2.xsource files are stored under http://xcat.svn.sourceforge.net/svnroot/xcat.

- xcat-contrib - contributed code
- xcat-core – source code for all releases
  1. branches
     1. 2.0 - 2.0 release source tree
     2. 2.1 – 2.1 release source tree
  2. trunk - 2.x (next) release source tree
- xcat-deps – xcat dependency code

Usually all new development is checked into the trunk. Only fixes go into the branches.

1.1 Setting up SVN for windows

1. Download svn-1.5.1-setup.exe from http://subversion.tigris.org/servlets/ProjectDocumentList?folderID=91
2. Install it on PC by running *svn-1.5.1-setup.exe*.
3. Create directories for my sandbox (workspace in SVN's term) C:\Ling\xCAT\sb\xcat-core\trunk\bin
4. Extract files. From command line window:

cd C:\Ling\xCAT\sb\xcat-core\trunk\bin
svn checkout http://xcat.svn.sourceforge.net/svnroot/xcat/xcat-core/trunk
5.Add a new file by first creating a file called test, then run
svn add test.
6. Check in the changes
svn --username linggao -m "my comments " ci (It asks my password).
7. undo what I have done
svn del ling_test
svn --username linggao -m "my first experience on SVN" ci

1.1 Setting up SVN for RedHat

1. Download and install the following from http://the.earth.li/pub/subversion/summersoft.fay.ar.us/pub/subversion/latest/rhel-3/bin/.
   apr-util-0.9.5-0.3.i386.rpm
   subversion-1.3.2-1.rhel3.i386.rpm
2. Create directories for workspace: /svn
3. Extract xCAT source code using command-line:

   cd /svn

   svn checkout http://xcat.svn.sourceforge.net/svnroot/xcat/xcat-core/trunk
4. Use same commands as above (svn add, svn del, svn .....ci).
5. To update you development tree later

   cd /svn

   svn update

1.1 Building xCAT rpms

To get the latest level of the xCAT code and build the rpms on your machine:

```
cd /svn/xcat/xcat-core/trunk
svn update
./makeperlxcatrpm
./makeclientrpm
./makeserverrpm
./makexcatsnrpm
./makexcatrpm
rm -f /usr/src/redhat/RPMS/noarch/xCAT-nbroot-core-x86_64*.rpm
./makenbrootrpm x86_64
```

```
rm -f /usr/src/redhat/RPMS/noarch/xCAT-nbroot-core-ppc*.rpm
./makenbrootrpm ppc64
```