CMT Release Manager

Overview

The CMT Release Manager is split into several components. The Batch submission system, the Workflow manager, and the Release Manager. These components work together to perform the automated builds in CMT. The Batch submission is at the lowest level. It accepts input from the layers above, such as the Workflow manager, and submits the jobs to lsf. Optionally, it notifies the layer above when jobs have started and when they finished, along with any information such as return code or output. The Workflow manager is a simple rule engine. It submits jobs to the Batch submission system and, based on rules and the output from a job submitted to lsf, it determines the next job to execute. Finally, the Release Manager is a set of scripts that are registered in the Workflow system. The are executed based on the rules set in the Workflow.

Cron system

The cron system is a simple script designed to execute a script at the specified time on the specified host. It's supposed to supplement trscron by allowing us to execute the cron job on several computers at once while only having one computer actually execute the script. The goal is to allow for failover when a host crashes and becomes unavailable for a prolonged period. The information of the cron system is stored in a MySQL database on glastDB.slac. stanford.edu (aka glastlnx01.slac.stanford.edu). The information is used by a script to perform the actions described in it.

Database

The cron system's database contains two tables. The first table is called **crontab** and contains the same information as what would be stored in the cron command configuration file. The settings table contains information such as which host is the primary host for executing the cron jobs. All other hosts, except the primary one, will not execute the job.

The **crontab** table is organized as follows:

```
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| user       | varchar(255) |      |     | glast   |       |
| minute     | varchar(255) |      |     | *       |       |
| hour       | varchar(255) |      |     | *       |       |
| dayOfMonth | varchar(255) |      |     | *       |       |
| month      | varchar(255) |      |     | *       |       |
| dayOfWeek  | varchar(255) |      |     | *       |       |
| command    | varchar(255) |      |     |         |       |
+------------+--------------+------+-----+---------+-------+
```

The user field contains the user under which the cron job should be executed. All other fields are identical to the cron command and will not be explained here.

The **settings** table is organized as follows:

```
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| name  | varchar(255) |      |     |         |       |
| value | varchar(255) |      |     |         |       |
+-------+--------------+------+-----+---------+-------+
```

The **settings** table contains only name/value pairs to describe the settings. Currently supported name/value pairs are:

- **master** - The value field for this name field will contain which host is considered the master. Cron jobs only execute on the master.
- **runTime** - This field controls how long the cron script will remain running before it shuts down.

The **runTime** field is meant as a workaround to afs tokens. Since afs tokens expire, this shuts down the cron script. A new instance of the script is started shortly after by trscron with a new afs token.

Batch Submission

The Batch submission system consists of two a few simple scripts and a simple database system. Its only purpose is to submit jobs to lsf and notify the caller of the job when it has finished. Additionally, it provides the caller with the return code information as well as any output produced. Due to technical difficulties, the Batch submission system does not guarantee that it can provide the output or the return code. The central information about the Batch submission system is stored in a MySQL database. This database is accessed by a bunch of scripts to perform the tasks described in it. Additionally, the information in the database is displayed visually on a webpage.

Database

The database table for the Batch Submission system is stored on glastDB.slac.stanford.edu (aka glastlnx01.slac.stanford.edu). The database name is **bsub** and contains a single table entry named **jobs**.

The **jobs** table is structured as follows:

```
+----------------+---------------------------------------------------+------+-----+---------+----------------+
| Field          | Type                                              | Null | Key | Default | Extra          |
+----------------+---------------------------------------------------+------+-----+---------+----------------+
| jobId          | bigint(20) unsigned                               |      | PRI | NULL    | auto_increment |
| lsfId          | bigint(20) unsigned                               |      |     | 0       |                |
| command        | varchar(255)                                      |      |     |         |                |
| args           | varchar(255)                                      | YES  |     | NULL    |                |
| queue          | varchar(255)                                      |      |     | short   |                |
| batchOpts      | varchar(255)                                      | YES  |     | NULL    |                |
| status         | enum('waiting','submitting','pending','running') |      |     | waiting |                |
| tries          | int(11)                                           |      |     | 0       |                |
| onSuccess      | varchar(255)                                      | YES  |     | NULL    |                |
| workingDir     | varchar(128)                                      |      |     |         |                |
| outputLocation | varchar(255)                                      | YES  |     | NULL    |                |
| user           | varchar(128)                                      | YES  |     | NULL    |                |
+----------------+---------------------------------------------------+------+-----+---------+----------------+
```

The fields in the table are used as follows:

- **jobID** - A unique ID that is automatically created when a new job is registered with the batch submission.
- **lsfId** - Initially contains a value of 0 and will be filled with the ID provided by lsf once the job is submitted.
- **command** - The full path of the executable to be submitted to lsf.
- **args** - The arguments to be passed to the executable.
- **queue** - The queue the job is submitted to.
- **batchOpts** - A string containing the options to pass to bsub while submitting.
- **status** - Field containing what the status of the job is.
  - **waiting** - The job is waiting to be submitted.
  - **submitting** - The bsub command is in process of executing.
  - **pending** - The job has been submitted to lsf but hasn't started executing yet.
  - **running** - The job has started execution.
- **tries** - A counter indicating how many times the job has attempted to execute the bsub command (and failed).
- onSuccess* - A string indicating what to do when the job is submitted, starts executing, or finishes.
  - The format of the string is either **script:/path/to/script** or **email:valid@email.address**.
- **workingDir** - Path to the working directory to use when running the job.
- **outputLocation** - Full path to the file where the output of the job should be saved to.
- **user** - The user to run this job under. Currently only supports the glast user accounts **glastrm**, **glast**, etc.

Scripts

The Batch submission system is controlled by a few scripts:

- **/u/gl/glast/infraCron/batchSub.pl** - This script is executed by the cron system to submit jobs that are in the waiting stage.
- **/u/gl/glast/infraBin/bsubChange.pl** - This script is executed every time the job changes state (pending to running to finished).

Workflow

The Workflow system is a rule based engine which submits jobs to lsf via the batch submission system. The jobs are submitted to lsf based on rules defined in the database. The rules can evaluate conditions based on the output or return code of previous jobs in the same run. The workflow system is controlled by a MySQL database and scripts. There is a web interface for viewing the current rules set in the Workflow system but it is broken as of this writing.

Database

The database is stored on glastDB.slac.stanford.edu under the database name **Workflow**. The database contains the following tables:

- **action** - This table contains the rules that determine which script to execute based on the rules set and the current finished script.
- **forceTrigger** - This table contains entries about manually started runs in the workflow which were not triggered by the trigger script.
- **forceTriggerSettings** - Options to pass to the workflow when forcing a run via the forceTrigger table.
- **main** - The main table containing all the workflows known and the scripts for that workflow.
- **running** - Instances of the registered workflows that are currently running.
- **settings** - Settings for the workflow system.
- **trigger** - Trigger scripts for defined workflows.

Scripts

The workflow system consists of a single script located in **/u/gl/glast/infraBin/workflowComplete.pl**. This script is called by the batch submission every time the script finishes. The script will evaluate the rules of the finished script and determine which scripts to execute next. Those scripts are submitted to the batch submission system. Once finished, those scripts will call this script again. The cycle continues until the rules no longer allow for the execution of new scripts.

Webpage

The webpage for the Workflow system is https://www.slac.stanford.edu/www-glast-dev/cgi/Workflow. The scripts to display this webpage are located in /afs /slac.stanford.edu/g/www/cgi-wrap-bin/glast/ground/cgi/Workflow. As mentioned earlier, these pages currently do not work due to graphviz installation problems.

Release Manager

Scripts

The Release Manager is controlled by a bunch of scripts that are located in **/u/gl/glastrm/ReleaseManager/**, **/u/gl/glast/perl-modules**, **/u/gl/glast /ReleaseManager**, and **/u/gl/glast/infraCron**. The list of these script that require explanation is:

- **trigger.pl** - This script is executed by the Workflow system to determine when a new job should be started.
- **rmTodo.pl** - This script is executed to perform user initiated functions such as erasing builds, triggering builds, etc.
- All other scripts are fairly self explanatory and require arguments of the form **package~~version~~tag**.

For windows the **trigger.pl** and **rmTodo.pl** don't exist. Otherwise all other scripts exist in the windows equivalent in **V:\Glast_Software\Toaster\tools\ReleaseManager**.

Many of the scripts in /u/gl/glastrm/ReleaseManager/src/linux can be run by hand when there are problems:

compile.pl GlastRelease-v20r0p1-rhel4_gcc34

test.pl GlastRelease-v20r0p1-rhel4_gcc34

createUserRelease.pl GlastRelease-v20r0p1-rhel4_gcc34

Web interface

The web page for the Release Manager is https://www.slac.stanford.edu/www-glast-dev/cgi/ReleaseManager. It is controlled by the SLAC web server. The information is displayed by a bunch of perl scripts located in /afs/slac/g/www/cgi-wrap-bin/glast/ground/cgi/ReleaseManager.

Release Manager paths.

The Release Manager stores its data in several locations. These locations are:

- **/afs/slac/g/glast/ground/GLAST_EXT/tag** - Location of the external libraries to use while compiling.
- **/nfs/farm/g/glast/u30/builds/** - Location the builds performed by the Release Manager.
- **/nfs/farm/g/glast/u05/extlib/** - Location of the external libraries tarred up for the installer to use.
- **/nfs/farm/g/glast/u09/binDist** - Location where the installer files of the Release Manager build are located.
- **/nfs/farm/g/glast/u09/builds** - Old location where the Release Manager used to store builds.
- **/nfs/farm/g/glast/u09/documentation** - The location where the doxygen documentation is stored.
- **/nfs/farm/g/glast/u09/html** - The location where the output from checkout, build, and unit tests are stored.

For windows these paths are slightly different:

- **V:\Glast_Software\Toaster\GLAST_EXT** - Location of the external libraries.
- **V:\Glast_Software\Toaster\tools\builds** - Location of the builds.
- The other paths are identical but are accessed via the windows path \\slaccfs\...

Release Manager database

The Release Manager database is stored in glastDB.slac.stanford.edu (aka glastlnx01.slac.stanford.edu). The database name is **ReleaseManager**

The database contains these tables:

- **checkout** - Contains checkout information about the packages built for a particular build.
- **checkoutPackage** - Contains information about a particular build.
- **compile** - Contains compile information about the packages built for a particular build.
- **exclude** - Contains builds that should not be built.
- **settings** - Contains settings for the Release Manager.
- **status** - Contains the status of builds currently in progress.
- **test** - Contains unit test information about packages build for a particular build.
- **todo** - Contains pending tasks for the Release Manager to perform.

The **checkoutPackage** is the main table which is populated with the information about a build. The main field is the **cpId** field which all other tables use to identify which build they represent. The **chechkout**, **compile**, and **test** tables contain the information for individual packages in a build. They tie to the **chec koutPackage** table with the cpId field.

When a build is erased, it is entered into the **exclude** table to prevent the Release Manager from rebuilding the build.

Archiver

The archiver is another system that works on top of the Workflow system. Just like the Release Manager, it has a trigger script registered with the workflow system that triggers when a new archive job is started. Similarly, the archiver contains database entries and a web page for viewing the information.

Database

The database for the Archiver is stored on glastDB.slac.stanford.edu with the database name of **Archive**. It contains the following tables:

- **archiveContent** - The files archived.
- **pending** - Contains pending archive entries.
- **tarFiles** - Contains a list of tar files that belong to a single archive job and their locations in mstore/gstore/astore.
- **task** - The main table containing the archive job.

All the tables are tied together via an id field defined by the **task** table. Tar files are split at a size defined by the user at archive creation time and the list of tar files belonging to a single task is stored in the **tarFiles** table.

Scripts

The scripts for the Archiver are stored in **/u/gl/glast/infraBin/Archiver**. The scripts located there are:

- **archive.pl** - The script invoked by the workflow when archiving.
- **archiver.pl** - This script is the controller script with which users can archive, restore, delete jobs.
- **delete.pl** - This script is invoked by the workflow when deleting archived files.
- **determineTask.pl** - This script is invoked to determine if a job is for archiving, deleting, restoring, etc.
- **finish.pl** - This script is invoked as the last script by the workflow to cleanup the database and mstore/gstore/astore.
- **restore.pl** - This script is invoked for restore operations.
- **trigger.pl** - This script is invoked by the workflow to determine if new archiving tasks are pending.
- **verify.pl** - This script is invoked for verify operations.

All of these scripts are mostly frontends to the mstore/gstore/astore applications. The use the expect perl module to programmatically control tar which expects interactive input for creating and splitting tar files.

Here is an example how to archive a file CLHEP-1.9.2.2.tar.gz in directory /nfs/farm/g/glast/u05/extlib/tiger_gcc33:

/afs/slac.stanford.edu/u/gl/glast/infraBin/Archiver/archiver.pl --module "Manual" --callback "user:kiml" --path "/nfs/farm/g/glast/u05/extlib/tiger_gcc33" --user glast --name "u05.extlib.tiger_gcc33.CLHEP-1.9.2.2.tar.gz" --file CLHEP-1.9.2.2.tar.gz --method mstore archive

And to restore it to /nfs/farm/g/glast/u30/tmp:

/afs/slac.stanford.edu/u/gl/glast/infraBin/Archiver/archiver.pl --module "Manual" --callback "user:kiml" --path "/nfs/farm/g/glast/u30/tmp" --user glast --name "u05.extlib.tiger_gcc33.CLHEP-1.9.2.2.tar.gz" --file CLHEP-1.9.2.2.tar.gz --method mstore restore

Notification

The RM scripts notify the users of problems encountered during checkout, compile, or unit tests. This notification is done by the **finish.pl** script. It checks the database for every package that belonged to a build to determine if the package had any errors during checkout, build, or unit test compiles. If there were errors, the script checks who the author is for the failure and notifies the author. Additionally, all the failures are accumulated and sent to a general failure mailing list as specified by the settings page.

Webpages

The web page for the Archiver is https://www.slac.stanford.edu/www-glast-dev/cgi/DiskArchive. It is controlled by the SLAC web server. The information is displayed by a bunch of perl scripts located in /afs/slac/g/www/cgi-wrap-bin/glast/ground/cg/archive.

Installer

The installer consists of a single backend script, a database, and a perl/java frontend script. The database is populated by the backend script which is located along with all the other ReleaseManager scripts. The frontend scripts read the database and display choices of files to download and install for the end user.

Database

The database is located on glastDB.slac.stanford.edu and is called **dependency**. The database consists of several tables:

- **externals** - lists the external libraries, their versions, and the path for a particular package
- **packages** - lists packages and their dependencies. Top level packages have the **mainPkg** entry set to true (1).
- **pkgdep** - creates the dependency list between packages in the **packages** table
- **tags** - a list of supported tags that can be downloaded.

The **tags** table creates a list of tags. A checkout package is listed in the **packages** table with the name, and version as well as the **mainPkg** field set to true. All packages contained in the checkout package are listed in the **packages** table as well with the appropriate tag but with **mainPkg** set to a value of false (0). The **pkgdep** table creates a list of dependencies between packages. The package whose dependencies are being listed is specified by the field **p kgId** which refers to a package in the **packages** table. The **needPkgId** is then an ID that refers to an entry in the **packages** table with the same value in **pk gId** as **needPkgId**. This is considered the dependency of the package.

Backend script

The backend script is installed in ~glastrm/ReleaseManager/src/linux/createInstaller.pl for linux and V:\Glast_Software\toaster\Tools\ReleaseManager\src\windows for windows. This package gets a list of packages for a particular checkout package and inserts the appropriate entries into the database described above based on the output from cmt's dependency graph (cmt show uses). Additionally, the script creates tar.gz files or .zip files of the packages and stores them in /nfs/farm/g/glast/u09/binDist.

This script occasionally fails to execute and simply re-running the script will most of the time fix the problem. The script has been setup not to allow duplicate entries in the database or re-tar/zip files that already exist. Should a tar/zip file be corrupt, the file needs to be erased first prior to re-running this script. Otherwise the tar/zip files will not be re-generated.

Frontend script

The frontend script is what is known by others as the Installer. It is installed in ~glast/infraBin/installer.pl. This script reads the contents of the database using the glastreader account and displays choices for the user to download. The script works both on windows and linux provided that unzip and perl are available on windows.