**PMT Framework Document**

**Important Paths**

The full directory structure of PMT framework can be found under /opt/PMT/, this directory is known as PMTROOT.

/opt/PMT/etc

Configuration of each flow and common configuration between all flows (pmt.conf)

/opt/PMT/pmtsys/bin

Libraries related to libxml ( dbxml )

/opt/PMT/pmtsys/var/collection\_history

Stores the collection history, i.e. files that are retrieved already.

/opt/PMT/pmtsys/lib/perl5/site\_perl/5.18.1/

Perl Lib installation

/opt/PMT/pmtsys/lib/pmt/

Framework modules.

/opt/PMT/pmtsys/lib/pmt/AppModules

Modules which help parsing the files ( SaxParsers )

/opt/PMT/pmtsys/lib/pmt/templates/

Ctl and par files for sqlldr to load the files into oracle database.

/opt/PMT/pmtsys/lib/xquery/

Xquery function

/opt/PMT/pmtsys/scripts/

Scripts to handle flow xml loading , dumping basically to handle environment around the framework.

/opt/PMT/logs

Contains logs generated by flows running.

/opt/PMT/tmp

Contains standard error logs generated by sub processes running for a flow.

/opt/PMT/var/db

Contains dbxml database used to flow definition.

/opt/PMT/bin

Stores executables like dbxml, sqlplus, perl etc.

Important specification in .bash\_profile in home dir of owner user.

PMT\_ROOT\_DIR="/opt/PMT"

export PMT\_ROOT\_DIR PMT\_INSTALL\_DIR

PATH=/usr/bin:$PATH

PATH=$PATH:$PMT\_INSTALL\_DIR/util:/var/opt/oracle:/opt/oracle/product/11.2.0.3/bin

PATH=/opt/PMT/binary/perl/bin:$PATH

PATH=/opt/PMT/pmtsys/bin:/opt/PMT/pmtsys/scripts:/opt/PMT/bin:$PATH

export PATH

ORACLE\_SID=PMT01P ; export ORACLE\_SID

ORACLE\_HOME=/opt/oracle/product/11.2.0.3 ; export ORACLE\_HOME

ORACLE\_BASE=/opt/oracle ; export ORACLE\_BASE

NLS\_LANG=American\_america.WE8ISO8859P1 ; export NLS\_LANG

NLS\_DATE\_FORMAT="DD-MON-YY HH24:MI:SS" ; export NLS\_DATE\_FORMAT

export PMTROOT=/opt/PMT

export PMTCONFIG=/opt/PMT/etc/pmt.conf

export TNS\_ADMIN=${ORACLE\_HOME}/network/admin

export DB\_HOME=/opt/PMT/var/db/

Terminology used.

Flow

A particular type of data is fetched in PMT and parsed and loaded into an oracle table, the processes that does all these activities are grouped into one name called a flow. A flow consists of set of sets that are performed in a specific order for the run to be successfully.

Role

A flow consists of roles, a role within a flow defines a high level task at hand. It could be either collection, parsing or loading or combination of these.

Group

A role consists of groups. Group further states what has to be performed within a flow. In a role called “load” group can be “load” and “after\_loader”.

IC

IC is called InitialContext, its purpose is to store configuration data and partial functions which can be called at any point from the code to get specific tasks like resolution of a URI done.

Helper

As name stands there are processes in PMT names as helpers to help achieve certain tasks. Like there is a Monitoring Helper named PMTMonitoringHelper.pm , thre is a Log helper named PMTLogHelper.pm etc. For example , PMTPLHelper helps manage parsers and loaders roles.

Interceptor

An interceptor is added into the IC when a job starts as a plugin. It has certain job that it can perform like PMTConfigInterceptor : if in any part of the program a config is needed and then PMTConfigInterceptor is invoked, another example : PMTXPathInterceptor is called when xpath function is used to find something in xml definition of the flow.

JOBDEF

JLOG

PMTRunner

Location : /opt/PMT/pmtsys/scripts/

This script is responsible for starting any mentioned role in a flow.

Takes input as

Runid

Role : role of a flow to be run

Help : Print out help message.

Debug : Run is debug mode.

Jlogfile : Final Jlog to be dumped to.

Flowcd : Flow to be started

Param : command like params ( like )

Algorithm:

* Load SysConfig ( Configuration PMTCONFIG set in the .bash\_profile , then it reads the PMTCONFIG file and loads configurations into a hash – this is achieved by getSysConfig function from PMTUtilities module)

Check if Flow is in runnable state in dbxml

If Yes, then check if role is enabled in dbxml for the particular flow

( isFlowRunning, roleIsEnabled – both functions defined in PMTFlowRepo modules where statement for checking if the flow is in running state or not is

/status/flow[@name=<flow\_name> and @status="running"] – by connecting to pmt\_log.dbxml

For checking role is enabled or not for a flow using xquery

/status/flow[@name=<flow\_name> and @status="running"]/roles/role[@name=<role\_name>])

* Load All plugin [PMTENVInterceptor, PMTConfigInterceptor, PMTXPathInterceptor, PMTMonitoringHelper, PMTLogHelper ]
* Set up Logging
* Reads the Job definition from dbxml
* Starts the driver
* Waits for the driver to end
* Check if an error occurred in the run of driver , if yes set the exit code to 1 and end.

PMTJobManager [ driver ]

Location : /opt/PMT/pmtsys/lib/pmt/

This script has two packages inside (ProcessGroupManager and PMTJobManager ).

Its purpose is to start the group(s) within a role and keep looping over till the group(s) run is finished.

Algorithm

This module is invoked from PMTRunner , hence always first run function is called.

* Module run
  + Sets up the setup steps from the Job Definition within node param with name=’process’ and data\_type=’xnode’
    - Calls execLocalSteps with node <setup> from the job definition
      * If conditions are okay , it loops on individual step under steps/step node and calls execLocalStep
        + execLocalStep function

Extract all the nodes under node <step> that is not pre\_reqs

Loop over the above result

If the name of the node is ic\_update call xfind, parseParamset function on it and if all is okay merge the return value to IC

Return

* + - * Return
    - Find all the group under groups definition in the job definition and Loop over it
    - Check if all the conditions are met for the group if not then skip the group else
    - Request ProcessGroupManager to create a new group
      * New method is called which in turns calls appendJLOG from PMTXPATHInterceptor to create a xml structure for the group.
    - Loop over all the created groups
      * Check if the group isfinished
        + Isfunished function – setup is done for the group then start is called by this function.
      * If the job is finished then exit
      * Repeat above task after .05 mil sec.
    - End loop, return.

ProcessGroupManager

Location : /opt/PMT/pmtsys/lib/pmt/

( within file PMTJobManager.pm )

Function List

* + new
  + setup
  + start
  + execLocalCallStep
  + execLocalIcUpdateStep
  + execLocalStep
  + end
  + run
  + isFinished

Algorithm

This module is invoked by PMTJobManager , for setup, start and to check if the group is finished.

* New is called by PMTJobManager to setup the group for a role
  + It creates a new execution context ( IC )
  + All initialization is done for the new IC
  + New JLOG xml entry is created (calls the function from PMTXPathInterceptor plugin).
  + Returns
* Then isFinished function is called by PMTJobManager to check if the group process is finished what it was doing.
  + Upon calling isFinished function it first calls the run method on itself
    - Run method calls the start method in turn
      * Start method calls the setup method
        + Setup method checks if there is any wait condition for that group to start

If there is then it sets certain fields to notify parent methods that wait condition is enforced.

If the wait condition are okay, then it calculates the number of process that can be started for the group ( either by checking CPU core or by the specification in Job Definition xml )

Finds the loopdriver for the group [ what is a loop driver ?? ]

On the basis of loopdriver , it sets up some policy. [ what policy ? ]

Returns with \_setup\_=1

* + - * + When \_setup\_ is 1 then it checks for wait condition, if okay then it logs the message and then some data into JLOG.
        + Return with \_started\_ =1
      * If there is no wait condition then start the helper process ( helper process is determined from Job definition ).
      * If there is wait condition then return
      * Return

PMTExecContext

Location: /opt/PMT/pmtsys/lib/pmt/

The purpose of this module is to create an execution context for the processes in a flow to run and perform its job.

This module uses operator overloading to achieve its task. It provides processes with the variable using which processes can call functions ( shared, anonymous, etc ) or can call modules to handle specific tasks. This variable is called initialcontext.

All the parameter values are held in variable “stacked\_parameters”. One can see what parameters are currently held in the variable by statement

print Dumper $ic->handleparameters();

All named helpers are held in variable \_namedhelpers\_

All named interceptors are held in variable \_namedinterceptors\_

Function list

New

Asstring

Handleparamerters

getRawHash

merge

addParameterStack

getCurrentLevel

getLayerData

getRootLevelParameters

getRootParameter

setRootParameter

getOutParameter

setJobReturnParameter

setJobParameter

getJobParameter

getAllOutRunJobParameters

popParameterStack

addHelper

getNamedHelper

getNamedPlugin

getNamedPlugins

hasNamedPlugin

addGarbageCleanupHandler

handleGarbageCleanupHandlers

clearGarbageCleanupHandlers

startInterceptors

stopInterceptors

addInterceoptors

addInterceptor

addPlugin

getHelpersByMethodName

freeze

exportProcessImage

getNumberOfLevels

coalesce

resolve

expand

createfuncUnitSpec

AUTOLOAD

Upon calling the new function of this module it returns a bless object with hash value storing anonymous functions for each of the methods mentioned above.

When a method is called above using the class object then the method in turn calls that anonymous method using the hash key which stores the reference to that method.

Brief Description of some of the anonymous methods

addPlugin

Calls anonymous method of key ADDPLUGIN within hash $operations

It takes following parameters

Name,module,initparams ( initparams can contain config\_file, name,mountpoint, mode )

This method instantiate the plugin and calls addhelper or addinterceptor methods to add them to the list of helpers and interceptors. [ it uses PMTStackedDataHandler object for this purpose ]

Adding this plugin helps when a call is made using the IC variable and if those calls are to be handled by any of the pre defined helper or interceptor then the object of that helper or interceptor is invoked to complete the task.

addHelper

Calls anonymous method of key ADDHELPER.

It adds a helper

addInterceptor

Calls anonymous method of key ADDINTERCEPTOR

It adds an interceptor

startInterceptor

After adding , the interceptor object needs to be started

stopInterceptor

After all the processing is done, interceptor needs to be stopped properly.

Expand

Throughout the programming , the config used is reference to a configuration value in the conf file or run time generated value or command line parameter, so this function helps in expand a line meaning it will get the referenced value ( it could be another reference to another place )

Resolve

Resolve also helps in resolving reference. It uses PMTUtilities expand function to resolve a reference.

PMTXPathInterceptor

Location: /opt/PMT/pmtsys/lib/pmt/

This module is an interceptor which helps with parsing or resolving anything which starts with JOBDEF or JLOG

PMTHelperHelper

Location: /opt/PMT/pmtsys/lib/pmt/

This module is called by ProcessManagerGroup from run function , to start the processing of a role. It also uses overloading hence all that processgroupmanager has to do is to use this statement to call the run function of PMTHelperHelper

$h->();

Its purpose is to run PMTPLHelper ( what does it do ? see below ) and send the service request to the invoked child processes, get log messages from child processes and log it into the file. Check if the child process is complete or not. Check if there is any thing pending in the worklist to be done or not.

Execute all the steps in a group.

PMTPLHelper

This script is responsible for starting the processing, in processing it does following steps,

1. Create listener thread to listen to PMTHelperHelper for adding anything into the wok queue.
2. Create runner thread to do the actual processing.
3. Runner thread reads the work queue and decides what to do with that item on basis of type of item

It uses packages which is written within itself

LogRedirector

To send log messages to the master process (PMTHelperHelper)

Controller

To put control on all the tasks that are to be performed by the runner thread.

Runner

To perform the actual tasks

PMTUtilities

This module provides some the methods for the framework to run properly.

Modules

Icdefined :- checks if the parameter is defined in the IC or not

mergeRecursiveHash :- This function merges two hash, this is used when ic is to be merged with another hash with new data.

h2a :- converts hash to array

expand :- It is responsible for expanding the config value.

For example :- {{\_NOW\_|formatDateTime %Y%m%d}}

getPMTSysConfig :- returns PMT system configuration like database details, resource detail usually from pmt.conf file in /opt/PMT/etc.

loadClass :- load a perl module dynamically.

partial :- returns a sub reference which executes the sub that is passed to it along with parameters passed to it.

evalBoolean :- this methods evaluates a Boolean condition to true or false.

serializeTo, deserializeFrom :- This two subs are used between handshake of parent and child processes.