

Process Request Engine

Version 31.00 - PRE-30.2.1.0

PROPRIETARY NOTICE

This document contains proprietary information furnished for evaluation purposes only; except with the express written permission of STGMASTEK, such information may not be published, disclosed, or used for any other purpose. You acknowledge and agree that this document and all portions thereof, including, but not limited to, any copyright, trade secret and other intellectual property rights relating thereto, are and at all times shall remain the sole property of STGMASTEK and that title and full ownership rights in the information contained herein and all portions thereof are reserved to and at all times shall remain with STGMastek. You acknowledge and agree that the information contained herein constitutes a valuable trade secret of STGMastek. You agree to use utmost care in protecting the proprietary and confidential nature of the information contained herein.



Document Information

Current Version:	31.00
Date Last Updated:	06/06/2013
Last Updated By:	Kedar Raybagkar
Author:	Kedar Raybagkar
Date Created:	11/17/2003
Approved By:	
Approval Date:	



Revision History

Version Number	Date Updated	Revision Author	Summary of Major Changes Made
1.0	June 09, 2003	Kedar Raybagkar	Process Request Engine and Scheduler
2.0	Oct.21,2003	Kedar Raybagkar	Changes made in the scheduler part
3.0	Feb.05, 2004	Kedar Raybagkar	Changes made in Monitor Thread and Mailing Facility if the thread is identified as STUCK
4.0	Feb.24, 2004	Kedar Raybagkar	Changes made in scheduler. Few columns that were missing in the document were added
5.0	May 03, 2004	Kedar Raybagkar	Changes made for terminating the process. Added a new interface in the existing package.
6.0	Oct.06, 2004	Kedar Raybagkar	Added details on logging of messages and web server details. Change made in features list and limitations (in the scheduling section).
7.0	Nov.16, 2004	Kedar Raybagkar	Added multiple Week Days for frequency equals to MINUTE, HOUR, and WEEK in the feature list and scheduler table
8.0	Dec.03, 2004	Kedar Raybagkar	Changes made in the Web Server to show the percentage completion of a process being executed. Added a new interface to the existing group of interfaces.
9.0	Dec.20, 2004	Kedar Raybagkar	Overall correction made. Updated the flow chart for scheduling.
10.0	Jun.08, 2005	Anusha Iyer	Formatted the document Reviewed the document to check if grammatically correct
11.0	Jun 22, 2005	Kedar C. Raybagkar	Changes made for Log4J and a new frequency.
12.0	August 25, 2005	Kedar C. Raybagkar	Changes made to add a new interface IStopEvent and to add a new property currenttimestamp in pr.properties.
12.0	September 02, 2005	Anusha Iyer	Formatted the document
13.00	December 20, 2005	Kedar C. Raybagkar	Update the properties section.
14.00	January 20, 2006	Kedar C. Raybagkar	Updated the properties section and the features.



Version Number	Date Updated	Revision Author	Summary of Major Changes Made
15.00	June 30, 2006	Kedar C. Raybagkar	Updated for poolconfig.xml, pr.properties changes, as well as added new feature for Bouncing, Re-Booting, Virtual Queue of PRE.
16.00	December 30, 2007	Kedar C. Raybagkar	Clustering of PRE is now possible. The PRE will make sure that there will be one and only one PRE running at a given time. Corresponding PRE version is V1.0R23.01.001
17.00	March 21, 2007	Kedar C. Raybagkar	Scheduling bug fixed. Also, sample programs are now associated with the bundle that will serve as examples.
18.00	April 11, 2008	Kedar C. Raybagkar	Added new features in PRE.
19.00	May 06, 2008	Kedar C. Raybagkar	Added new frequencies in scheduler.
20.00	Jul 10, 2008	Kedar C. Raybagkar	Added the new changes with respect to the SKIP flag and association of <i>ICalendar</i> functionality in the document.
21.00	Oct 10, 2008	Kedar C. Raybagkar	Changes made for new properties and for associating multiple calendars to create a complex scheduling.
22.00	Apr 08, 2009	Kedar C. Raybagkar	Added new functionality for Context related changes made in PRE. Added System property settings and the new property defined for web server ip or hostname.
23	Nov 19, 2009	Kedar C. Raybagkar	Removed the dependency between the JDBC Pool and PRE. Now PRE can use any datasource. Introduced findbugs in the ant build script. Introduced new properties to handle the datasource factory. Introduced preferences PRE factory implementation.
24	May 05, 2010	Kedar C. Raybagkar	Added clustering features for ACTIVE / PASSIVE mode using Terracotta Server.
25	Aug 19, 2010	Kedar C. Raybagkar	Changed the Terracotta to Hazelcast. Features such as Retry have been added.
26	Dec 20, 2010	Kedar C. Raybagkar	Added a new API in the PREContext.



Version Number	Date Updated	Revision Author	Summary of Major Changes Made
27	04/18/2010	Kedar Raybagkar	Updates made for changes done in PREV1.0R29.xx.xxxx
28	09/19/2011	Kedar Raybagkar	Updates made for changes done in PRE30.0.0 regarding 'keep alive' functionality in the scheduler.
29	11/01/2012	Kedar Raybagkar	New API added. Service and Singleton.
30	06/06/2013	Kedar Raybagkar	License requirement removed and bundling or packaging has changed.
31	05/09/2014	Kedar Raybagkar	Modified API ProcessRequestServicer. getParams() to return Map instead of HashMap. Also the API will now return an unmodifiable map.



1 Table of Contents

Version 29.00 - PRE-30.1.0.....	i
1 Table of Contents.....	vii
2 Specifications for Process Requirement Framework	2-1
2.1 Introduction.....	2-1
2.2 Limitations/Possible Enhancement	2-1
2.3 Features	2-2
2.4 Architectural/Design Considerations	2-7
2.5 Stages/Components Involved	2-7
2.5.1 Submission of Requests.....	2-7
2.5.2 Scanning and Executing Pending Requests.....	2-12
2.5.3 Displaying Status of Requests	2-12
2.5.4 Activity Log.....	2-12
2.5.5 Application Architecture and Design.....	2-13
2.5.6 More about Virtual Queue	2-19
2.6 Scheduler	2-19
2.6.1 Introduction	2-19
2.6.2 Scope	2-19



2.6.3	Schedule Design.....	2-19
2.7	More About E-Mail feature	2-34
3	Installing Process Request Engine (Scheduler)	3-37
3.1	Installation Process.....	3-37
3.2	Configuration XML/Properties File Changes.....	3-38
3.2.1	prinit.properties	3-38
3.2.2	pr.properties.....	3-39
3.2.3	jdbctype.properties	3-48
3.2.4	mail.properties.....	3-48
3.2.5	system.properties	3-51
3.2.6	poolconfig.xml	3-51
3.2.7	hazelcast.xml	3-53
3.3	Shell Scripts	3-55
3.3.1	startEng.sh.....	3-55
3.3.2	stopEng.sh	3-56
3.3.3	killEng.sh	3-57
3.3.4	truncLog.sh	3-58
3.3.5	bootEng.sh	3-58
3.4	Check the Installation	3-60
3.5	Configure <i>PRE</i> in Cluster mode	3-60
3.6	Points to Remember.....	3-61
3.7	More about Logging Messages	3-61
3.8	PRE Process Flow Diagram	3-63
3.9	Sample Screen Shots/ Console Log.....	3-66
3.10	Certified On.....	3-74
3.11	Known Issues.....	3-75



4	Libraries Used.....	4-1
----------	----------------------------	------------



List of Figures

Figure 1 console.out, the PRE Log	3-70
Figure 2 Web Service Monitoring	3-70
Figure 3 Running Process Status	3-71
Figure 4 Terminate Process Warning	3-71
Figure 5 Terminate Process	3-71
Figure 6 Stop Feature has been added	3-72
Figure 7 Stop Warning is displayed	3-72
Figure 8 Message for successfully invoking stop	3-72
Figure 9 Message if invocation of stop is un-successful	3-73



2 Specifications for Process Requirement Framework

2.1 Introduction

In the current architecture, processes are executed online, that is the following steps are performed:

1. User starts the process/report
2. Process starts on the server immediately
3. User waits for the process execution to complete

In such a scenario, time-consuming process leads to browser time-out. However, the process continues execution on the server but the user is not aware of the status of the process.

The idea for the framework evolved to facilitate the execution of time-consuming processes, such as Business Logic Programs and reports, in a user-friendly manner.

In this framework,

1. User can submit a request for the process and move on to carry out some other activity
2. Process runs in the background and the status of the process is periodically updated
3. User can view the status of the process via a report

2.2 Limitations/Possible Enhancement

1. Scheduling of Grouped Requests
2. Internationalizing messages
3. Adding an Event for processing a Request the way Event feature is provided for Scheduling

2.3 Features

The Process Request Engine (PRE) is **Multi-Threaded**, thus taking the maximum advantage of the CPU. As PRE is not bound to any business logic, it is very generic in nature. Thread Pooling is done for specific task execution such as transporting emails and for web services.

**Note**

It can be used across any functionality.

The programmer needs to extend `ProcessRequestServicer` and write a Business Login within one abstract method for the actual execution. And, if necessary, method can be overridden for performing clean-up activities.

The PRE request is divided in two parts:

1. **Grouped Requests**
2. **Stand Alone Requests**

Both the PRE requests are handled by *threads*. The connections to the database are stored in a proprietary Connection Pool mechanism, which has the following features:

1. **Stale connections** that can be closed are available.

**Note**

A stale connection is a connection that is idle for the time specified in 1.7.2 pr.properties.

2. The Pool will not close the initial connections specified in the same property file.
3. For the initial connections, a **periodic check** would be made by the POOL to keep them alive. The self-check time interval can be specified in the 1.7.2 pr.properties file.
4. The Pool checks for **Connection Leaks and Open Statements and Open ResultSets**. Leaked connection is trapped and returned to the pool. Where open statements are statements created in the program using the connection object but not closed explicitly. The Pool guarantees that the open statements are automatically closed without a warning. However, these warnings are clearly marked in the log files.

**Note**

Refer to the JDBCPOOL documentation for further details. This is a separate product of STGMASTEK and is plugged into PRE.

The PRE has a **Scheduler**. The scheduler is intelligent enough to advance the Parameter Dates (if marked as Dynamic) for a requested process with the scheduled frequency.

Example



Consider that we request a report that gives the Account Statement of a Bank. We specify the Account Number, From Period, and To Period.

Assumption	A/C No#	10001912
	From Period	01-JAN-2003
	To Period	31-MAR-2003

Let us consider that this report is requested on 01-APR-2003. We schedule this report with a frequency of three months until the end date as 31-DEC-2003. The date parameters are marked dynamic.

The PRE when executing this queued request schedules another request with the request date as 01-JUL-2003 with the following parameters:

A/C No#	10001912
From Period	01-APR-2003
To Period	31-JUN-2003

The PRE also has an **inbuilt JOB Monitor** that can identify and warn for **STUCK THREAD (ST)**.

Each Stand Alone requests can be associated with the following two time limits:

- **STUCK THREAD LIMIT (STL)**
- **MAX STUCK THREAD LIMIT (MSTL)**

A thread is marked as ST if the time taken to process a thread exceeds STL. A warning is logged. Another warning is logged when a ST crosses the MSTL.

Note These warnings can also be sent via an inbuilt SMTP mailer functionality to the desired personals (administrators).

If the underlying object implements **ITerminateProcess** interface, the PRE will initiate the Terminate process for that object if MSTL is crossed. The PRE will not do a Reboot or Bounce or Shutdown sequence if the JOB implements the **ITerminateProcess** interface upon identification of STUCK job. The PRE will wait for a new cycle of JOB Monitor and if the JOB monitor still finds the same JOB still executing then the PRE will take an appropriate action.

For STL and MSTL actions, mailing details can differ. If all execution threads of the PRE are identified as STUCK and has crossed the MSTL and if the PRE is unable to spawn a new thread then the PRE **bounces** itself. The mail subject, header, recipients, footer, etc. can be driven through the *mail.properties* property file.

Mail is sent if the:

1. Schedule gets cancelled or finished
2. Execution of a scheduled activity fails



Note The Monitor feature is optional and can be stopped if not required. Refer `pr.properties`.

The scheduler has been enhanced to have **multiple weekdays in MINUTE, HOUR, and WEEK frequency**. In other words, if the schedule should run from Monday to Friday, and not on Saturday and Sunday, then the same can be achieved in one single schedule.

The scheduler has been enhanced further to have a **pre-programmed frequency**. This means that any program that implements interface `IPreDefinedFrequency` can now be called to schedule or pro-create new jobs. The implementer class will use its own logic to increment the schedule.

Further, for monitoring process, PRE has an **inbuilt Web server** and shows statistics in the HTML format that would be refreshed after every five seconds. Refer Figure 2 Web Service Monitoring. Web service utilizes Thread pooling. As the usage for this web service is not anticipated the thread pool core threads are allowed to die after a inactivity period of 5 seconds.

The Web server can display the status of a running process if the class implements interface `IProcessStatus`. Refer Figure 3 Running Process Status.

PRE uses *log4j* for logging.

PRE has a **Re-Boot / Bounce** feature. In case of emergencies, like the database link went down or any other IO exception that broke the JDBC TCP/IP link with the queue, the PRE used to go down. Now enhancements have been carried out in PRE so that the PRE will wait for the amount of time specified in `pr.properties` and will attempt a complete self reboot. The number of attempts can be specified in the `pr.properties` file and even after attempting the maximum attempts the engine is unable to start then it gets terminated. For those who want to achieve a programmatic reboot an API is also provided. Bounce is **different** than Re-Boot. In **Re-Boot** a clean shutdown is made. A clean shutdown happens only when all the running processes are complete; the engine waits for the current running processes to complete. In **Bounce**, all the threads cross the maximum stuck thread limit and the PRE is unable to spawn any new thread for processing then the PRE simply dies and boots itself back without a clean shutdown sequence getting fired.

Virtual Queue is introduced in PRE. This enables us to have multiple instances of PRE configured against a single queue and each PRE will service certain types of requests. As a separate instance of PRE is running for specific request types, the administrator can then set different configurations for PRE. Let us assume that there is business need to have a particular type of requests to be serviced by PRE without delays and on priority. Now, PRE will process all requests that are in queued status and thus not therefore distinguish between types of requests. With **Virtual Queue** it is now possible to have a dedicated PRE to service only that particular type of requests which are crucial to the business. This also can be treated as **Thread Profiling**. A virtual queue can be defined in two ways in the `pr.properties`. Property `requesttypefilter` can have two values either `INCLUDE` or `EXCLUDE`. And the actual request type can be defined in the property `processrequesttype` as say `INTERFACE`. So, if the values are `INCLUDE` and `INTERFACE`, then the PRE will service requests of type `INTERFACE` and nothing else. If it was `EXCLUDE` and `INTERFACE` then the PRE will execute all requests whose request type is not `INTERFACE`. One can have comma separated values to the request type. E.g. If filter is `INCLUDE` and type is `INTERFACE, REPORT` then it means that PRE will be service these both types of requests and nothing else.



ACTIVE / PASSIVE clustering is now made available in this version of PRE. For this to happen, PRE makes use of Hazelcast library file, the details can be found here <http://www.hazelcast.com/>.

Case 1: Say we have PRE A and PRE B participating in a cluster and PRE A is currently ACTIVE and B is in PASSIVE mode. In case PRE A gets killed or crashes then the jobs that were being executed by A will fail over to B and B will be made ACTIVE. Please note that the PRE B will re-start the jobs that were failed over. This means that if there is any need of serialization then that needs to be handled by the job itself. The default `tc-config.xml` is made available in the resources folder. In case of *n* PREs participating in the cluster then the PRE that gets registered first on the Hazelcast will be made active and all others will be made PASSIVE. In case the Active PRE is killed or crashes then the subsequent PRE that was registered second on the Hazelcast will be made Active and all others will maintain their current PASSIVE state.

Case 2: Say only one instance of PRE is started and it starts executing jobs. If the PRE is killed then there is no other instance of PRE for fail over. Thus it will simply die and the process will never be taken up again unless the persistence configuration is added to the Hazelcast.

In case of a STOP request processed by the Active PRE and it terminates itself; then all other PASSIVE instances of PRE participating in the cluster will shut themselves down. Refer to the `hazelcast.config` in `pr.properties` for configuring the hazelcast cluster details.

Time elapsed is now stored by PRE in the queue table. Now it is very easy to read the elapsed time that a JOB took for itself to execute. The elapsed time is the time difference between request end date and request start date.

One can **define and associate multiple business calendars** to the schedule.

Calendar should define “Working” and “Non-Working” days (Holidays and Weekends). PRE will not execute a job if reoccurrence of a task falls on “Non-Working” day and will either cancel or pre-pone or postpone the execution date (depending upon set up). This feature can be added on top of the existing frequencies exhibited by PRE that includes pre-programmed frequencies as well.

It is not necessary to associate a calendar with the schedule. If a JOB recurs every month and the scheduled date happens to be a week end (Sat or Sun) then schedule can be associated with a flag that enables to advance or postpone the schedule date to either a Friday or Monday.

Through the Web Service, **one can kill a particular process spawned by PRE**. The process thus killed ensure that the resources are freed from PRE but the underlying process if had called a process on a different server such as a database procedure/function call or given a call to a report on different report server that process may still be active and running. The status of such a process is marked as “K”. Refer to the appendix section for screen shots.

A global context is now made available to all the threads that would be executed by PRE. This context can be used to share objects across executions. All the classes must now extend the abstract class `stg.pr.engine.ProcessRequestServicer` instead of implementing `stg.pr.engine.IProcessRequest` interface.



The job can now be **retried after retry time unit** (MINUTE or HOUR) and specified retry time, in case, the job throws a `CProcessRequestEngineException` or returns false.

The **Business Intelligence and Reporting Tool from Eclipse** have been made available as a startup service. (This though is currently not supported to be on a cluster).

Keep Alive has been introduced as a feature in the schedule. This means that even if the job returns false or throws exception then the schedule will still pro-create the job as per its frequency.



Note

Remember that if there is any SQL or IO exception during persistence of the pro-creation of the scheduled job then in that case the scheduler will fail to pro-create the JOB.

SERVICE, SINGLETON API have been introduced. At times it was necessary to hook certain classes with the lifecycle of PRE. The service and singleton interfaces introduced in this version will enable the developers to hook on custom classes directly within the lifecycle management of PRE.



2.4 Architectural/Design Considerations

Thread pool v/s creating threads every time a task is requested.

Thread pooling is a useful technique when it comes to executing short-lived tasks or threads. Rather than create a brand new thread for each task, you can have one of the threads from the thread pool pulled out of the pool and assigned to the task. When the thread is finished with the task, it adds itself back to the pool and waits for another assignment. The sole purpose of creating PRE was to execute long running tasks; thus the core functionality does not use thread pooling mechanisms. But the in-built Web Server or the SMTP mailer uses thread-pooling techniques. The maximum capacity of the thread pool is set to 5. If more than 5 concurrent requests come to the thread pool for processing 5 threads are created and the 5 tasks are executed while the rest are queued.

Procreating all jobs one at a time v/s procreating jobs as per the schedule recurrence.

It was decided to procreate jobs as and when they get executed instead of procreating jobs based on the schedule recurrence. This was decided so as to keep the records in the queue table very limited to current active schedules and not load the data unnecessarily. Also, terminating a schedule is faster as just one record gets flagged instead of marking multiple records.

2.5 Stages/Components Involved

1. Submission of a request (saved in the database)
2. Scanning the database for pending requests and executing them
3. Displaying the status of requests
4. Activity Log

2.5.1 Submission of Requests



The request is stored in a database. The request table acts as a queue to the PRE. The PRE scans this queue and acts accordingly.

2.5.1.1 Request Attributes


The following attributes are required to define a single-request.

Attribute	Description
Request Id (N12, PK)*	Auto-generated Id for the request
User Id (V8)	Id of the user who submitted the request
Request Date (Date)	Date on which the request is submitted



Attribute	Description	
Request Status (V2)	Status	Description
	I	Process Initiated
	Q	Waiting in the Queue
	P	Processing Started
	S	Completed or Success
	E	Completed with an Error
	C	Processing Cancelled By PRE
	X	User Cancelled Request
	K	Killed Through Web Service
Process Class Name (V500)	Class File Name (including package hierarchy) of the process to be executed	
Group Standalone Indicator (V2)	Indicator	Description
	G	Grouped Request
	S	Standalone Request (Default values is 'S')
Request Start Date Time	Date and time when the request processing starts  Note The Pre will update this.	
Request End Date Time	Date and time when the request processing ends  Note The Pre will update this.	
Group Id (N12)	Auto-generated only if Group Standalone Indicator is 'G' (For Future Use)	
Group Request Sequence No (N12)	Sequence number of the request within a group (For Future Use) Use should define the sequence	
Job Id (V100)	A unique identifier that identifies the job in a system. This is a free text field.	
Job Name (V500)	Name of the job. This is a free text field.	



Attribute	Description
Request Log File Name (V500)	Request Log File Name that will store the path  Note The Pre will update this.
Schedule Date and Time	Schedule date and time when the request would be execution
Schedule Id (N12)	Schedule number identifying the schedule
Stuck Thread Limit	Refer to Features
Stuck Thread Max Limit	Refer to Features
Request Type	Any request must be identified against a request type. The request type can be GENERAL or any business needs. There must be a pre configured PRE to service such a type. Refer to pr.properties section.
Priority	The requests are sorted in ascending order of the priority. All the requests whose priority is 1 will be taken for processing first instead of 99 even though the scheduled time is same for both the request.
Email Ids	The scheduler will sent the status emails to the email ids specified against this field. If the email ids are not associated then it will default to the email ids specified in the mail properties file.
Verbose Time elapsed	<p>The time elapsed between the Request Start time and the Request End time is verbose as follows:</p> <p>y: Year M: Month d: Day H: Hour Format starts from 1 to 23 m: Minutes 0 to 59 s: Seconds 0 to 59 S: Milliseconds 0 to 999</p> <p>Therefore a representation such as 11m 34s 757S means 11 minutes 34 seconds and 757 milliseconds. Another example 24y 9M 22d 23H 59m 59s 887S this means a JOB was running for 24 years 9 months 23 hours 59 minutes 59 seconds and 887 milliseconds. Now this is just an example to show all the attributes and is not actual elapsed time of a job. This column will be populated once the JOB ends its execution and the status is either marked as SUCCESS or ERROR.</p>
Cal scheduled time	Calendar scheduled time is the date and time as per the frequency associated with the schedule. The actual schedule time may differ than that of the Calendar schedule time. This can be null. If it is null PRE will consider the calendar scheduled time same as the actual scheduled time.




Attribute	Description
Text1	Open Text field 1. Can be null.
Text2	Open Text field 2. Can be null.
Num1	Number field 1. Can be null. Java Type double.
Num2	Number field 2. Can be null. Java Type double.
Retry_times N(10)	Defines the number of retries to be done in case of failure.
Retry_time_unit V(10)	Defines the time unit for waiting before the actual retry execution of the job. Valid values are MINUTE and HOUR
Retry_time N(10)	Defines the actual sleep time before retry. To be read in conjunction with Retry_time_unit.
Retry_cnt N(10)	For internal use.

The fields Text1, Text2, Num1 and Num2 are provided for storing project specific reference data.

2.5.1.2 Request Parameter Attributes







The following attributes are required to define parameters for a request.

Attribute	Description
Request Id (N12, PK)	Request Id inserted into the request table
Parameter No (N12, PK)	Auto-generated Id (For each parameter in a request)
Parameter Field Name (V100)	Name of the parameter
Parameter Value (V100)	<p>Value of the parameter stored as VARCHAR data type</p> <div>  Note <ul style="list-style-type: none"> • Format for Date value should be DD/MM/YYYY • Format for Time Stamp should be DD/MM/YYYY HH:MM:SS <p>These formats can be set in the properties file pr.properties.</p> </div>



Attribute	Description																										
Parameter Data Type (V2)	<p>Data type in which the parameters would be available to the business objects while the object is being executed in the background. Refer to paramarrvaluedelim property for delimiter used while specifying array values.</p> <table><tr><th>Data Type</th><th>Description</th></tr><tr><td>I</td><td>Integer</td></tr><tr><td>D</td><td>Double</td></tr><tr><td>L</td><td>Long</td></tr><tr><td>DT</td><td>Date</td></tr><tr><td>TS</td><td>Timestamp</td></tr><tr><td>S</td><td>String</td></tr><tr><td>IA</td><td>Integer Array</td></tr><tr><td>DA</td><td>Double Array</td></tr><tr><td>LA</td><td>Long Array</td></tr><tr><td>SA</td><td>String Array</td></tr><tr><td>DTA</td><td>Date Array</td></tr><tr><td>TSA</td><td>Timestamp Array</td></tr></table>	Data Type	Description	I	Integer	D	Double	L	Long	DT	Date	TS	Timestamp	S	String	IA	Integer Array	DA	Double Array	LA	Long Array	SA	String Array	DTA	Date Array	TSA	Timestamp Array
Data Type	Description																										
I	Integer																										
D	Double																										
L	Long																										
DT	Date																										
TS	Timestamp																										
S	String																										
IA	Integer Array																										
DA	Double Array																										
LA	Long Array																										
SA	String Array																										
DTA	Date Array																										
TSA	Timestamp Array																										



Attribute	Description						
Static Dynamic Flag (V2)	<p>Valid Values S-Static and D-Dynamic</p> <table> <tr> <th>Values</th><th>Description</th></tr> <tr> <td>Dynamic (D)</td><td> <ul style="list-style-type: none"> Indicates whether the scheduler should increment the date parameter with the desired frequency or not Valid for Parameter Data Type as DT, TS, DTA and TSA. <p> Note In case the parameter data type is other than DT, TS, DTA or TSA the engine may either throw exception or simply ignore it.</p> </td></tr> <tr> <td>Static (S)</td><td> <ul style="list-style-type: none"> Indicates that even though the parameter is a date, the value must not be incremented by the scheduler <p> Note Valid for all data types</p> </td></tr> </table>	Values	Description	Dynamic (D)	<ul style="list-style-type: none"> Indicates whether the scheduler should increment the date parameter with the desired frequency or not Valid for Parameter Data Type as DT, TS, DTA and TSA. <p> Note In case the parameter data type is other than DT, TS, DTA or TSA the engine may either throw exception or simply ignore it.</p>	Static (S)	<ul style="list-style-type: none"> Indicates that even though the parameter is a date, the value must not be incremented by the scheduler <p> Note Valid for all data types</p>
Values	Description						
Dynamic (D)	<ul style="list-style-type: none"> Indicates whether the scheduler should increment the date parameter with the desired frequency or not Valid for Parameter Data Type as DT, TS, DTA and TSA. <p> Note In case the parameter data type is other than DT, TS, DTA or TSA the engine may either throw exception or simply ignore it.</p>						
Static (S)	<ul style="list-style-type: none"> Indicates that even though the parameter is a date, the value must not be incremented by the scheduler <p> Note Valid for all data types</p>						

2.5.2 Scanning and Executing Pending Requests

The PRE would be installed as an operating system service or a batch job or cron job that gets executed. It could also be a console-based system, which displays the status of each request being run.

Remember The PRE will automatically stop if there is any JDBC Exception that is raised within the PRE while running. If any other class which implements *IProcessRequestEngine* interface and gives a call to *System.exit (0)*. The PRE in such cases will attempt a reboot sequence. Refer to pr.properties reboot section.

2.5.3 Displaying Status of Requests

A JSP page picks up requests from the database and displays the status. The user would be provided an option to filter requests to be displayed on the status, that is, user can view only running processes or completed processes.

2.5.4 Activity Log

The service maintains a log that logs details such as Start Time, User Id, Request Id, End Time, Success, or Error Status.



The process also maintains a log where the process specific details can be logged or the progress of the process can be logged. However, this logging is responsibility of the business object.

The various attributes of the log file - system and request files - can be configured through a property file. The property file can have attributes such as file size and file path (includes the file name for system log).

2.5.5 Application Architecture and Design

The PRE is written in Java using JDK 1.3.

Ideally, this should be installed on a separate server and not on Application or Web server. It is just a measure to void performance problems on either side.

2.5.5.1 Database Design

Tables defined are:

PROCESS_REQUEST (For fields of the table, see Section 0)

PROCESS_REQ_PARAMS (For fields of the table, see Section 2.5.1.2)

2.5.5.2 Thread Processing

The PRE spawns threads for processing multiple requests at the same time. The PRE itself is concurrency safe. It is not responsible for any requests processed in threads that could cause concurrency. The user would have to ensure that such requests do not go at the same time.

One can limit the number of threads that are spawned and are dependent on the number of connections defined in the properties pr.properties.

Example	If the maximum connections that can be established were defined as two, then one thread would be spawned. One connection is reserved for PRE for its queue.
----------------	---

2.5.5.3 Components (Classes/Interface/Resource Files)

Class CProcessRequestEngine

The Class CProcessRequestEngine is the main PRE that would run continuously, scan for queued requests, and execute them. For the process execution flow, refer PRE Process Flow Diagram.

Abstract Class ProcessRequestServicer

This class implements the interface IProcessRequest. With PRE version V1.0R26 onwards all the class that implemented IProcessRequest now needs to extend this class. This class implements the method *getContext()* from the interface IProcessRequest.



Interface IProcessRequest

The business object should extend abstract *ProcessRequestServicer* class that implements the IProcessRequest interface barring two methods namely processRequest and destroy. The outline of the interface and methods to be implemented are as follows:

```
public interface IProcessRequest
{
    public static enum REQUEST_SOURCE {
        OFFLINE(0, "Request Executed By PRE");
        ...
        ...
    }

    public static enum REQUEST_TYPE {
        STANDALONE("S", "Standalone request"), GROUPED("G", "Group request");
        ...
        ...
    }

    public static enum REQUEST_STATUS {
        QUEUED("Q", "Queued"),
        LAUNCHING("I", "Launching the job"),
        PROCESSING("P", "Processing"),
        COMPLETED("S", "Completed"),
        ERROR("E", "Errored Out"),
        SUSPENDED("C", "Suspended Job"),
        USER_CANCELLED("X", "User Cancelled Job"),
        KILLED("K", "Job Processing Killed");
        ...
        ...
    }

    public Connection getConnection ();
    public Map<String, Object> getParams ();
    public PrintWriter getResponseWriter ();
    public long getRequestId ();
    public String getUserId ();
    public REQUEST_SOURCE getSource ();
    public boolean processRequest () throws CProcessRequestEngineException;
    public void endProcess () throws CProcessRequestEngineException;
    public PREContext getContext();
    public boolean isFailedOver();
}
```

public Connection getConnection ();

The connection passed from the CProcessRequestEngine represents a valid connection to the database. It can be used by the business to interact with the database.




```
public Map<String, Object> getParams ();
```

The Map passed from the CProcessRequestEngine contains all the parameter keys and values that are required by the business object to execute. Also, the returned Map is un-modifiable.

```
public PrintWriter getResponseWriter ();
```

The PrintWriter object passed from the CProcessRequestEngine is a reference to the request log file used by the business object to log messages/errors during execution. If the method is invoked only then the *PrintWriter* object is created. By default the ProcessRequestService class implements endProcess() method which closes the stream. If you happen to override it please make sure to call the super method.

```
public long getRequestId ();
```

The request ID passed from the CProcessRequestEngine is the ID that identifies the request.

```
public REQUEST_SOURCE getSource ();
```

The Source passed from the CProcessRequestEngine is initialized to PRE indicating that the business object is being executed through a request.

```
public boolean processRequest() throws CProcessRequestEngineException;
```

This is the starting point of the business object. The PRE would set the status of the request in Process Request table as "I". The PRE would set the requisite attributes using the above-mentioned methods. The PRE would also set the status of the request in Process Request table as "P". This method is called by the PRE to start execution of the business object. The business object would update the status of the request in the Process Request table as "S" or "E" depending on the return value of this method.

```
public void endProcess() throws CProcessRequestEngineException;
```

This method is called by the PRE when the execution of the processRequest() method is complete. This method will be invoked even if the processRequest() method throws any exception. This method should typically be used do the clean-up activities such as say closing of open streams, etc.

```
public PREContext getContext();
```



The implementer of this class must provide the PREContext that can be used across multiple threads.

```
public boolean isPREActivelyScanning() ;
```

True indicates that the PRE is in a running mode and is actively scanning for new requests for processing. In case of stop request received by PRE, then this method will return false after PRE processes the stop request.

Interface ITerminateProcess

The ITerminateProcess interface may or may not be implemented. The PRE would execute the method if a STUCK Thread scenario occurs and if the property terminateprocess is set to Y in pr.properties.

The PRE would not guarantee the termination of the process. It only gives an indication that the process needs to be terminated. Actual termination of the process is implementation specific.

```
Public void terminateprocess (String strReason)
```

This method would be executed on the class that implements ITerminateProcess.

Interface stg.pr.engine.IProcessStatus

If a class desires to show the status of the ongoing activity, the class will have to implement the interface stg.pr.engine.IProcessStatus. The status returned by the class will be displayed in the web services of the PRE. Refer to status column in Figure 3 Running Process Status.

```
public interface IProcessStatus
{
    public String getStatus();
}
```

Interface stg.pr.engine.startstop.IStopEvent

If a class desires to know the stop event serviced by the Engine then the class must implement the interface stg.pr.engine.startstop.IStopEvent. The Engine will initiate the stop event on the running job if the class that is instantiated implements this interface.

```
public interface IStopEvent {
    /**
```

The Engine will call this method on a running JOB that implements this interface if a stop request is serviced by the Engine. The implementor class must then do accordingly to stop itself. Please note that the Engine will wait till the JOB dies its own death. Therefore it is important to know for the JOB

that the stop request has been requested and the JOB should then take care to terminate itself gracefully.

```
*/  
public void notifyEngineStopEvent();  
}
```

Interface Service

Any custom class that needs to be hooked on to the lifecycle management of PRE must implement the Service interface. The following API is exposed and must be implemented by the custom class.

```
public interface Service {  
  
    /**  
     * Perform initialization routines.  
     * This method is invoked when the PRE starts.  
     *  
     * @param context PREContext implementation.  
     */  
    public void init(PREContext context);  
  
    /**  
     * Perform clean up and destroy actions.  
     * This method is invoked when the PRE shuts down.  
     *  
     * @param context PREContext implementation.  
     */  
    public void destroy(PREContext context);  
  
}
```

Interface Singleton

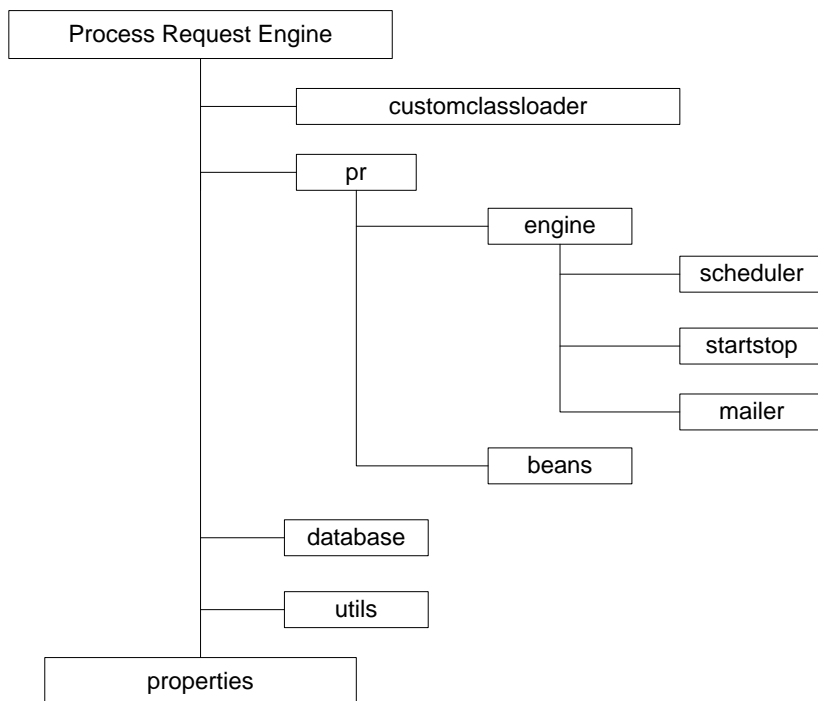
The custom class that implements Service @ time may provide access to the instance. The class therefore must also implement Singleton. The following API is exposed and must be implemented by the custom class. The instances of the classes that implement Singleton are made available through the PREContext.getSingleton(Key); method.

```
public interface Singleton<T> {
```



```
/**  
 * Returns the instance of T.  
 * @return instance.  
 */  
public T getInstance();  
  
}
```

2.5.5.4 Package Hierarchy



2.5.6 More about Virtual Queue

This is with respect to *stopEng.sh* and *bootEng.sh* shell scripts which are given for administrative purpose.

2.5.6.1 Case I

The property *requesttypefilter* is defined as INCLUDE and lets assume that the PRE is configured to service more than one type of requests (say *processrequesttype* is defined as GENERAL,MIS). The *stopEng.sh* and *bootEng.sh* checks for a system property “reqtype” and then generates the stop or reboot request. If the property is not set then default value taken is “GENERAL”.

2.5.6.2 Case II

The property *requesttypefilter* is defined as EXCLUDE. Remember to define the property “reqtype” in both these shell scripts as now default “GENERAL” will not be applicable to such configuration.

2.6 Scheduler

2.6.1 Introduction

The scheduler is independent of functionality and procreates or schedules further requests depending on the schedule. The scheduler would be an integral part of the PRE.

2.6.2 Scope

The scheduler would work only for requests that are yet to be executed by the PRE (queued requests). The scheduler would schedule another request only after the execution of the queued request.


The scheduler itself is not a PRE running independently. Therefore, it would not procreate or schedule requests on its own. The PRE needs to execute a queued request, having a valid schedule, to schedule further requests.

2.6.3 Schedule Design


2.6.3.1 Schedule Attributes

Table Name: process_request_schedule




Attribute	Description										
Schedule Id	Auto-generated ID for a request  Note This can be the Request Id itself.										
Frequency Type	<p>Determines the scheduling frequency. Refer to section 0 Table Name: <code>schedule_event_calendar</code></p> <table border="1"> <thead> <tr> <th>Attribute</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Schedule Id</td><td>Corresponds to the schedule id from the <code>process_request_schedule</code> table.</td></tr> <tr> <td>Serial_no</td><td>Auto-generated Id (For each parameter in a request)</td></tr> <tr> <td>Category</td><td>Currently supported is CALENDAR</td></tr> <tr> <td>Process_class_nm</td><td>Defines the class that implements the interface <code>stg.pr.engine.scheduler.ICalendar</code></td></tr> </tbody> </table> <p>Supported Frequencies Table.</p>	Attribute	Description	Schedule Id	Corresponds to the schedule id from the <code>process_request_schedule</code> table.	Serial_no	Auto-generated Id (For each parameter in a request)	Category	Currently supported is CALENDAR	Process_class_nm	Defines the class that implements the interface <code>stg.pr.engine.scheduler.ICalendar</code>
Attribute	Description										
Schedule Id	Corresponds to the schedule id from the <code>process_request_schedule</code> table.										
Serial_no	Auto-generated Id (For each parameter in a request)										
Category	Currently supported is CALENDAR										
Process_class_nm	Defines the class that implements the interface <code>stg.pr.engine.scheduler.ICalendar</code>										
Recur	<p>Recurrence factor for the schedule</p> <p>Example If <i>Recur</i> factor is set to 2 and the <i>Frequency Type</i> is set to DAY, then it implies that recur every two Days. Similarly, if <i>Frequency Type</i> is set to WEEK, then it implies that recur every two Weeks.</p>										
Start Date	Date when the schedule starts										
Schedule Stat	<p>Status of the Schedule</p> <p>A - Active F - Finished X - Cancelled by the PRE C - Cancelled by the User</p>										
User Id	User who created the schedule										



Attribute	Description
On Week Day	<p>Stored as seven digit string value with values ranging from 0 to 7 with each digit representing one week day</p> <p>Type equals to MINUTE, HOUR, and WEEK where</p> <p>0 - On All Days</p> <p>1 - SUNDAY</p> <p>2 - MONDAY</p> <p>3 - TUESDAY</p> <p>4 - WEDNESDAY</p> <p>5 - THURSDAY</p> <p>6 - FRIDAY</p> <p>7 - SATURDAY</p> <p>Example If <i>Recur</i> is 1, <i>Frequency Type</i> is WEEK, and <i>On Week Day</i> is 0001000, then the schedule will recur the task every one WEEK only on WEDNESDAY.</p> <p>If <i>On Week Day</i> is 0101010, then the schedule will recur the task every 1 WEEK on MONDAY, WEDNESDAY, and FRIDAY.</p> <p>In case of <i>Frequency Types</i> FIRST, SECOND, THIRD, FOURTH, and LAST_IN_MONTH, if the <i>On Week Day</i> is entered as 0110000, then the scheduler would only consider the first occurrence of 1. In this case, MONDAY</p>
End Date	<p>Date when the scheduler would stop scheduling further requests</p> <p>Either End Date or End Occurrences must be specified.</p>
Request Start	For Later Use
End Occurrences	<p>Schedule would execute tasks for the specified times</p> <p>Either End Date or End Occurrences must be specified</p>
Process Class Name	<p>Name of a Java class that has implemented an interface named IScheduleValidator</p> <p> Note This class is instantiated to validate the schedule. The schedule is cancelled if the class returns FALSE.</p>
Start Time	<p>Specified if the <i>Frequency Type</i> is MINUTE and/or HOUR</p> <p>Schedule would be valid between the Start Time and End Time</p>



Attribute	Description
End Time	Specified if the <i>Frequency Type</i> is MINUTE and/or HOUR Schedule would be valid between the Start Time and End Time
Future Scheduling	<p>Yes or No</p> <p> Note Yes indicates that the PRE would schedule requests only for future date and time.</p> <p>Example If the Request was scheduled to Run Daily on 01/Jan/2004 12:00:00 but the PRE was shut down on 31/Dec/2003 23:00:00, this request was not executed on 01/Jan/2004.</p> <p>The PRE was started only on 03/Jan/2003, and if the <i>Future Scheduling</i> flag is Yes, the PRE would execute the request made on 01/Jan/2004 but the next request would be on 03/Jan/2004 12:00:00 or future date.</p> <p>If this flag is set to No, the PRE would run this request of 01/Jan/2004 but the next request would be on 02/Jan/2004 12:00:00</p>
Fixed Date	Fixed Date is to be used when we want a schedule to be exactly on a particular date. E.g. 30 th day of a month is not available in February month so then the scheduler will not schedule a JOB for the month of Feb if the fixed date is true.
Email Ids	Not in Use.
Skip flag	<p>Skip flag identifies if the schedule should check for pre-programmed Calendar or a Weekday check. The valid values can be “NA”, “D+”, “D-” and “SS” where :</p> <p>NA indicates that there is no skipping of the scheduled date and it is okay to accept the scheduled date as arrived by the associated frequency. Example: if a Calendar frequency “MONTH” is associated and say the schedule starts on Jan 01, 2008 and recur every 1 month then the next schedule will be Feb 01, 2008 and after that it would be Mar 01, 2008. Mar 01 2008 is a Saturday. PRE will not do any checks and will schedule a JOB to be executed on Mar 01, 2008.</p> <p>D+ Indicates that the schedule should be postponed by a Day.</p> <p>D- Indicates that the schedule should be advanced by a Day.</p> <p>SS Indicates that the schedule should be skipped to the next recurrence of the schedule.</p> <p>For D+ or D- or SS there must be at least a ICalendar or a Weekday Check Flag or both, otherwise the skip flag must be NA.</p> <p>Please read documentation for the property <i>icalendarinfiniteloopterminationcounter</i> in [3.2.2] pr.properties section</p>



Attribute	Description
Weekday Check Flag	<p>Valid values are Y or N.</p> <p>If the skip flag is set to either D+ or D- and if the weekday check is Y then the schedule date if happens to be on weekend then PRE will advance the date as per the skip flag to a week day. If the skip flag is set to SS then the schedule will be skipped to the next recurrence of the schedule. This will also reduce occurrence counter by 1.</p> <p>Weekday is considered as Monday through Friday. A weekend is considered as Saturday through Sunday.</p>
End_Reason	PRE will update this field upon any termination/closure of the schedule so as to get the reason behind the termination.
Keep_Alive	PRE will use this flag to determine if the schedule should be kept active. if the job fails then to the schedule will try to pro-create the job as per its associated schedule.

Table Name: schedule_event_calendar

Attribute	Description
Schedule Id	Corresponds to the schedule id from the process_request_schedule table.
Serial_no	Auto-generated Id (For each parameter in a request)
Category	Currently supported is CALENDAR
Process_class_nm	Defines the class that implements the interface stg.pr.engine.scheduler.ICalendar

2.6.3.2 Supported Frequencies Table

Frequency	Description
MINUTE	Per Minute
HOUR	Per Hour
DAY	Per Day
WEEK	Per Week
MONTH	Per Month

Frequency	Description
YEAR	Per Year
LDMONTH	Per Last of the Month
FIRST_MTH	Per specific <day> in the first week recurring on Month basis. Where <i>day</i> is one of Sunday through Saturday.
SECOND_MTH	Per specific <day> in the second week recurring on Month basis. Where <i>day</i> is one of Sunday through Saturday.
THIRD_MTH	Per specific <day> in the third week recurring on Month basis. Where <i>day</i> is one of Sunday through Saturday.
FOURTH_MTH	Per specific <day> in the fourth week recurring on Month basis. Where <i>day</i> is one of Sunday through Saturday.
LAST_MTH	Per specific <day> in the last week recurring on Month basis. Where <i>day</i> is one of Sunday through Saturday.
FIRST_YR	Per specific <day> in the first week recurring on Year basis. Where <i>day</i> is one of Sunday through Saturday.
SECOND_YR	Per specific <day> in the second week recurring on Year basis. Where <i>day</i> is one of Sunday through Saturday.
THIRD_YR	Per specific <day> in the third week recurring on Year basis. Where <i>day</i> is one of Sunday through Saturday.
FOURTH_YR	Per specific <day> in the fourth week recurring on Year basis. Where <i>day</i> is one of Sunday through Saturday.



Frequency	Description
LAST_YR	Per specific <day> in the last week recurring on Year basis. Where <i>day</i> is one of Sunday through Saturday.
PRE_DEFINED_FREQUENCY	A pre-defined programmable frequency that will be used by the Engine to schedule or pro-create new jobs.



2.6.3.3 Schedule Process

The PRE would check whether the request just processed has a schedule or not. If the schedule were associated with the request, the Scheduler would be invoiced. The scheduler does the following things in the same order as stated below:

1. Check for schedule End Date or End Occurrences.
 - If the end date is lower than the system date or the end occurrences are reached, simply mark the schedule as 'F' (Finished).
2. Validate the schedule by instantiating a class associated with the schedule.
3. If unable to load the class, the schedule is simply marked as 'X' (Cancelled).
4. If step 3 is successful, this class is executed. The class may return true or false depending on the logic written to validate the respective schedule.



Note If returned false, the schedule is marked as 'X' (Cancelled).

5. If step 4 is successful, the PRE-EVENT, if associated, is invoked and the event is executed. If the event returns false then the schedule is terminated.
6. Advance the scheduled time as per the schedule associated. If the frequency type is PRE_DEFINED_FREQUENCY then the engine calls the implementer `class` of `stg.pr.engine.scheduler.IPreDefinedFrequency` to advance the schedule time.
7. If the step 6 fails and the scheduler finds that the schedule cannot be advanced then the schedule is terminated and the schedule is marked as X.
8. If step 6 is successful then the scheduler checks for the SKIP Flag. If SKIP flag is not equals NA then the following steps occur.
9. The weekday check flag if set to true then the date advanced through step 6 is checked to see if it is a SUNDAY or a SATURDAY. If so then based on the skip flag associated the day is added or reduced. *D+* indicates that day will be added till the day is the immediate next MONDAY. *D-* indicates that day will be reduced from the date till it reaches the immediate previous FRIDAY. If the SKIP flag equals SS then the schedule is skipped to next occurrence of the schedule and the step 9 is repeated.
10. After step 9, if the schedule is associated with the calendar(s) (refer to table `schedule_event_calendar.process_class_nm`), a class that implements `stg.pr.engine.scheduler.ICalendar` interface then the program is called to check if the date returned through step 9 is a WORK day. If it is then control is forwarded to step 11. If it is not then again the date is advanced based on *D+* or *D-* or *SS* logic as described in step 9. The control is sent back to step 9. This is repeated for each calendar associated with the request.



11. Generate the Request ID.
12. If the parameters have a *Date* or *Timestamp* that are identified as dynamic, the scheduler advances the date parameters with the scheduled frequency. If the frequency type is `PRE_DEFINED_FREQUENCY` then the engine calls the implementer class of `Stg.pr.engine.scheduler.IPreDefinedFrequency` to advance the schedule time.
13. Replicate the request and its parameters for the new scheduled time and new scheduled parameters.
14. If step 9 is successful, the POST-EVENT, if associated, is invoked and the event is executed. If the event returns false then the schedule is terminated.
15. Check for the end date or end occurrences, if scheduled date is greater than the end date or end occurrences are reached, mark the schedule as 'F' (Finished).
16. Return to the PRE.

The scheduler would send mail if:

1. PRE is unable to instantiate the associated class
2. Associated class throws some exception
3. Scheduler generates some exception while scheduling
4. Schedule is marked as complete

Thus, the user who has scheduled a job need not worry about the scheduled request, as he would get the mail for the status of the schedule.

**Note**

For a schedule to work there must be an existing queued request. The user must ensure that the schedule time for the request should match with the schedule start date or the frequency. If it does not match the schedule, the PRE would match it from the next scheduled request.

2.6.3.4 Limitations/Possible Enhancements

- The scheduling is currently done for *Stand Alone* requests only. For grouped requests, the scheduling is not done.
- The *No End Date* feature is not yet provided.



2.6.3.5 Components

(Classes/Interface/Resource Files)

Interface `stg.pr.engine.scheduler.ISchedule`

The interface `Stg.pr.engine.scheduler.ISchedule` is the main Interface that is extended by all other interfaces within this package. The structure is as follows:

public interface ISchedule

{

 public static enum FREQUENCY {

 // SECOND("SECOND", "Second"),

 MINUTE("MINUTE", "Minute"),

 HOUR("HOUR", "Hour"),

 DAY("DAY", "Day"),

 WEEK("WEEK", "Week"),

 MONTH("MONTH", "Month"),

 YEAR("YEAR", "Year"),

 LAST_DATE_OF_MONTH("LDMONTH", "Last date of the month"),

 FIRST_DAY_OF_MONTH("FIRST_MTH", "First day of the month"),

 SECOND_DAY_OF_MONTH("SECOND_MTH", "Second day of the month"),

 THIRD_DAY_OF_MONTH("THIRD_MTH", "Third day of the month"),

 FOURTH_DAY_OF_MONTH("FOURTH_MTH", "Fourth day of the month"),

 LAST_DAY_OF_MONTH("LAST_MTH", "Last day of the month"),

 FIRST_DAY_OF_YEAR("FIRST_YR", "First day of the year"),

 SECOND_DAY_OF_YEAR("SECOND_YR", "Second day of the year"),

 THIRD_DAY_OF_YEAR("THIRD_YR", "Third day of the year"),

 FOURTH_DAY_OF_YEAR("FOURTH_YR", "Fourth day of the year"),

 LAST_DAY_OF_YEAR("LAST_YR", "Last day of the year"),

 PRE_PROGRAMMED("PREDEFINED", "Pre programed frequency");

 ...

 ...

 }

 public static enum END_ON {

 DATE,

 OCCURRENCES;

 }



```
public static enum SKIP_FLAG {
    NOT_APPLICABLE("NA"),
    SKIP_SCHEDULE("SS"),
    ADVANCE_BEFORE("D-"),
    ADVANCE_AFTER("D+");
}

/**
 * Identifier for Dynamic Indicator.
 * Comment for <code>DYNAMIC_INDICATOR</code>
 */
public static final String DYNAMIC_INDICATOR = "D";

/**
 * Identifier for Future Scheduling Only.
 * Comment for <code>FUTURE_SCHEDULING_ONLY</code>
 */
public static final String FUTURE_SCHEDULING_ONLY = "Y";

public static enum SCHEDULE_STATUS {
    ACTIVE("A", "Active"),
    COMPLETED("F", "Completed"),
    TERMINATED("X", "Cancelled by PRE"),
    USER_CANCELLED("C", "Cancelled by user");
}

/**
 * Set the PrintWriter object for logging purpose.
 *
 * @param pwOut PrintWriter
 */
public void setPrintWriter(PrintWriter pwOut);

/**
 * Sets the Process Request Entity Bean.
 *
 * @param pObjPREB ProcessRequestEntityBean
 */
public void setProcessRequestEntityBean(ProcessRequestEntityBean pObjPREB);
```



```

/**
 * Sets the Parameters associated with the Request.
 *
 * @param phmParameters HashMap
 */
public void setRequestParameters(HashMap phmParameters);

public void setDatasourceFactory(IDatasourceFactory factory);

}

```

This interface will have to be implemented and associated with the `process_request_schedule.process_class_nm`. This class will be instantiated by the scheduler to validate the schedule. If the implementer class returns `true` then the schedule is considered to be valid and then further processing will be done. If it returns `false` then the schedule will be terminated and the next job will not be pro-created.

**Note**

The implementer class should have a default constructor.

Interface `stg.pr.engine.scheduler.IScheduleValidator`

```

public interface IScheduleValidator extends ISchedule
{
    /**
     * Set the connection object
     *
     * @param pcon Connection
     */
    public void setConnection(Connection pcon);

    /**
     * Validates the schedule.
     * @param plSchId long Schedule Id associated with the Request.
     * @return boolean True if the schedule is valid and false if not.
     */
    public boolean validateSchedule(long plSchId);
}

```

Interface `pr.engine.scheduler.IScheduleEvent`

Any schedule can be associated with an Event Manager. To add a user defined event manager, the class will have to implement the interface



`Stg.pr.engine.scheduler.IScheduleEvent`. This class has the following structure. The class can maintain its state for a particular execution for the Pre and Post events.

```
public interface IScheduleEvent extends IScheduleValidator
{
    /**
     * The method is called before the schedule re-generates the next request
     * as per the schedule associated to the plOldRequestId.
     * If the method returns false the schedule is terminated. Note that commit must
     * not be used in this method.
     *
     * @param plOldRequestId The original Request id that was processed
     * @param plScheduleId The schedule associated for the plOldRequestId.
     */
    public boolean performPreAction(long plOldRequestId, long plScheduleId);

    /**
     * The method is called after the schedule re-generates the next request
     * as per the schedule associated to the plOldRequestId.
     * If the method returns false the schedule is terminated. Note that commit must
     * not be used in this method.
     *
     * @param plNewRequestId The new request id generated after scheduling.
     */
    public boolean performPostAction(long plNewRequestId);
}
```

Interface **pr.engine.scheduler.IPreDefinedFrequency**

The Scheduler is so flexible that one can add his/her own way of scheduling jobs. The class must implement the

`Stg.pr.engine.scheduler.IPreDefinedFrequency`. One can write any business logic within this class that will advance the passed date(s) based on the frequency encoded within it. The parameters associated with a request if has date parameters defined as dynamic then the scheduler will pass those dates along with their field names and will expect the class to return the future dates. PRE may give multiple calls to this class based on the associated SKIP flag. The class can maintain its state for a particular execution.

```
public interface IPreDefinedFrequency extends IScheduleValidator {
    /**
```



* This method is responsible to advance the date parameter with its own pre-defined

* frequency.

* The scheduler will call this method by passing the scheduled date & time that needs

* to be advanced based on the pre-defined frequency. The scheduler will identify the

* parameter `pGivenDate` as a parameter date associated with

* the request by* passing a boolean variable with value equals true. The field name

* associated with the parameter date is also passed for simplicity.

*

* **Note:** If the method returns a date which is less than the scheduled date

* and time then the schedule will be terminated.

*

* @param pGivenDate Date to be advanced.

* @param bParameter True if the date passed is a Parameter Date associated with

* the Request.

* @param pstrFieldName The field name associated with the date.

* @return Date Advanced date.

*/

```
public Date advanceDay(Date pGivenDate, boolean bParameter, String
pstrFieldName);
```

```
}
```

Interface `stg.pr.engine.scheduler.ICalendar`

This class defines the WORK day. A WORK day is a day on which the JOB can be scheduled. It is not necessary that a WORK day should not be Saturday or Sunday. Any day can be a WORK day similarly any day can be a NON-WORK day. The PRE instantiates the class and maintains the same object till the PRE procreates a new request. Thus the class can maintain its state. This class exhibits one method:

/**

* Sets the java.sql.Connection object.

*

* It is important that the class should not do any roll * back or commit on this connection object.

* The method should throw an Exception and the

* scheduler will do the roll back of the transactions.

*

* @param connection

*/

```
public void setConnection(Connection connection);
```



```
/**
 * Returns whether the given Day can be used for
 * scheduling a job.
 *
 * True indicates that the JOB can be scheduled on
 * this particular day.
 * False indicates that this particular day is a non
 * work day.
 *
 * @param day Day to be verified if this is a holiday.
 * @return Boolean
 * @throws SQLException
 */
public boolean isWorkDay(Day day) throws SQLException;
```

Connection object is provided if the underlying class needs to fetch a calendar persisted in database. The underlying implementation should not do any commit or rollback of the transactions nor closing of the connection.

Abstract class **stg,pr.engine.scheduler.DefaultScheduleManager**

Extend this class to create your own implementation of the schedule manager. The class implements the interface ISchedule, IScheduleEvent and IScheduleManager. This class provides the base implementation of the various helper methods but the main methods are kept as abstract or not implemented.



2.7 More About E-Mail feature

An SMTP mailer is provided with PRE. There are two simple ways to transport emails to the email server. One method uses synchronous transportation while the other uses asynchronous email feature. The caller need not wait for the transmission of asynchronous emails and can perform any other task. This mailer utilizes thread pooling mechanism to send asynchronous emails. The configured number of threads is 5 and currently this cannot be changed to any other number. The thread remains active for 10 minutes in the pool before it dies. In case, all the threads are busy in sending asynchronous emails then the subsequent newer emails are stored within a queue. During the shutdown process of PRE, the active threads are allowed to run till they finish their task. No new tasks are added. If there are pending emails waiting to be transported then policy as defined against the property `emailPolicy` is applied (refer to section 3.2.4). For further details about the policy class refer to `stg.pr.engine.mailer.EMailPolicy`. The default implementation provided with PRE is to discard these emails. A sample class `com.myapp.preimpl.SerializeEMailPolicy` is provided in examples directory that serializes the emails to a file on the system. The PRE when started will load such emails that were serialized.

**Note**

One should not use the same EMail object twice. PRE will catch that as a SPAM and will not transport. Such cases are logged and appear in the log file for further analysis.

To send emails in HTML format, set the type as HTML using the method `setEMailType(EMailType.HTML)`.

How to upgrade code from the previous version?

This is demonstrated using the following sample code taken from one of the implementation:

```
CMailer mailer = CMailer.getInstance(CSettings.get("pr.mailtype"));
String strMsgBody = "The Error/Control Report for Automated Document Upload, "+dateString+" is attached.";
String strMessageHeader = "This message is automatically generated by the Renaissance System Process Request Engine. Please do not reply. Contact your help desk or on-site support if you need any assistance.";
String strMessageFooter = "DISCLAIMER: Information contained and transmitted by this E-MAIL is proprietary to the Renaissance System. This is an auto generated mail intended only for certain privileged users of the system. Access to this email by anyone else is unauthorized. Any copying or further distribution beyond the original addressee is not intended.";
mailer.setMsgBody(strMsgBody);
mailer.setMsgBodyHeader(strMessageHeader);
```



```
mailer.setMessageBodyFooter(strMessageFooter);
mailer.setSubject("Error/Control Report for Automated Document
Upload, "+dateString);
mailer.setToRecipient(strToAddress);
if (strCCAddress!=null&&!(strCCAddress.equals("")))
{
    mailer.setCCRecipient(strCCAddress);
}
try
{
    mailer.setAttachment(file);
}
catch(java.io.FileNotFoundException fne)
{
    //do nothing
    System.out.println("File Not Found Exception : sendMail");
}
mailer.sendMail();
```

This code needs to be modified to:

```
EMail email = new EMail();
String strMsgBody = "The Error/Control Report for Automated Document
Upload, "+dateString+" is attached.";
String strMessageHeader = "This message is automatically generated by
the Renaissance System Process Request Engine. Please do not reply.
Contact your help desk or on-site support if you need any
assistance.";
String strMessageFooter = "DISCLAIMER: Information contained and
transmitted by this E-MAIL is proprietary to the Renaissance System.
This is an auto generated mail intended only for certain privileged
users of the system. Access to this email by anyone else is
unauthorized. Any copying or further distribution beyond the original
addressee is not intended.";
email.setMessageBody(strMsgBody);
email.setMessageBodyHeader(strMessageHeader);
email.setMessageBodyFooter(strMessageFooter);
email.setSubject("Error/Control Report for Automated Document Upload,
"+dateString);
email.setToRecipient(strToAddress);
if (strCCAddress!=null&&!(strCCAddress.equals("")))
{
    email.setCCRecipient(strCCAddress);
}
try
{
    email.setAttachment(file);
}
catch(java.io.FileNotFoundException fne)
{
    System.out.println("File Not Found Exception : sendMail");
}
```



```
CMailer.getInstance("SMTP").sendMail(email);  
OR  
CMailer.getInstance("SMTP").sendAsyncMail(email);
```

When to use Asynchronous Emails?

The best way to answer this is if you need a confirmation that the email was transported then go for synchronous email transportation. For all other cases, use asynchronous emails. Again, transportation of email to email server does not mean that it got delivered to the actual intended recipients but it just indicates that the Email server has taken the responsibility of actual delivery and that our email reached it. If the address itself is a non-existent address then there is no way an SMTP mailer will address this issue. As noted, above do not make use of this feature if the email is associated with an attachment which is of type DataSource. You may use it provided the attachment is stored as a reference to the file name or a file.

Interface `stg.pr.engine.mailer.EMailPolicy`

This class defines the EMailPolicy for serializing and/or de-serializing emails. Following methods are exposed.

```
public EMail[] loadEmails() throws IOException,  
ClassNotFoundException;
```

This method is called when the CMailer instance is created. It is expected that this method will de-serialize and load the email messages that were serialized during the shutdown, if any. It is expected that this same method will delete the store once the e-mails are loaded, otherwise the emails will be transported every time the PRE starts.

```
public void serializeEmails(EMail[] emails) throws IOException;
```

This method is responsible to serialize the emails, if any.



3 Installing Process Request Engine (Scheduler)

3.1 Installation Process

To install PRE, follow the steps mentioned below:

1. Unzip the file **packaged zip** on the server in a folder named **pre** or **pre<version number>**.
2. Create two directories inside the pre folder namely **logs** and **requestfiles**.
3. Update the properties located in the **properties** directory as per the guidelines mentioned in the following sections.
4. Update **startEng**, **stopEng** to reflect appropriate directory for scheduler and classpath.

PRE Logs

During the execution, the PRE writes the log of all errors raised/flow messages in a log file specified in the property files. The logging can be enabled or disabled by changing the log4j.properties file.



3.2 Configuration XML/Properties File Changes

For properties documentation, the directories created must have either executable or read/write permissions for the PRE.

Note 1

Red - Indicates that the value should match with the physical settings

Blue - Indicates that the value may be changed

Black - Indicates that the value must not be altered

~~Strike Through~~ - Indicates that the value is deprecated or is placed somewhere else.

Note 2

The properties names or column named Item may have its first character in Upper Case and this is because of the Word Formatting. All the property names start with lower case but may have an upper case letter(s) with some exceptions such as SMTP related properties. Refer to the actual physical properties file that came with the installation for accurate names.

3.2.1 prinit.properties

Item	Data Type	Description	Value
include.pr.source Type	String	Class name that loads the property file specified in the property include.pr.sourceParam	stg.utils.CPRSettings
include.pr.source Param	String	Absolute path and file name that should be loaded by the property include.pr.sourceType Note Only the path should be changed to the appropriate directory in which the pr.properties resides.	\${pre.home}/properties/pr.properties
include.pr.autoload	Boolean	Determines whether the property needs to be loaded during startup or not. Valid values are true or false.	True
include.mail.sourceType	String	Class name that loads the property file specified in the property include.mail.sourceParam	stg.utils.CPRSettings Note Class used for this property is same as that used for font.properties



Item	Data Type	Description	Value
include.mail.sourceParam	String	<p>Absolute path and file name that should be loaded by the property include.mail.sourceType</p> <p>Note Only the path should be changed to the appropriate directory in which the pr.properties resides. The name of the property file can be changed</p>	\${pre.home}/properties/mail.properties
include.mail.autoload	Boolean	Determines whether the property needs to be loaded during startup or not. Valid values are true or false.	true
include.system.sourceType	String	<p>Class name that loads the property file specified in the property include.system.sourceParam</p>	stg.utils.SystemPropertySettings
include.system.sourceParam	String	<p>Absolute path and file name that should be loaded by the property include.system.sourceType</p> <p>Note Only the path should be changed to the appropriate directory in which the system.properties resides. The name of the property file can be changed</p>	\${pre.home}/properties/system.properties
include.system.autoload	Boolean	Determines whether the property needs to be loaded during startup or not. Valid values are true or false.	true

3.2.2 pr.properties

Item	Data Type	Description	Value
# These properties configures the PRE to service a particular request type or set of types.			
Requesttypefilter	String	Determines the method which will determine the request types. Valid values are INCLUDE, EXCLUDE.	INCLUDE



Item	Data Type	Description	Value
Processrequesttype	String	Configures the PRE for a specific type or types of request(s) comma separated values. The values are specific to installations and the application that uses the PRE.	GENERAL
# These properties are used for obtaining connection to the database. ALL The strikethrough properties are discarded. Please refer to poolconfig.xml file for JDBC related properties.			
maximumfetchsize singlefetch	Long	Maximum queued requests to be fetched in a single fetch Note Higher number reduces the query hits to fetch records, thus, increasing the performance.	60
dataSourceFactory	String	Data Source factory implementation. This class must implement stg.pr.engine.datasource.IDataSourceFactory . Default implementation is given and can be used in conjunction with JDBC Pool V16.03; use the class name stg.pr.engine.datasource.defaultimpl.PREDataSourceFactory	stg.pr.engine.datasource.defaultimpl.PREDataSourceFactory
dataSourceFactoryConfigFile	String	Defines the configuration file for the data source. This can either be properties or XML file. This file should define the context name and properties such that the DataSource factory can make use of and return an appropriate datasource based on the names stored against properties dsforstandaloneeng and dsforgroupeng .	\${pre.home}/properties/poolconfig.xml



Item	Data Type	Description	Value
dsforstandaloneeng	String	Data source Name for the StandAlone Engine. PRE will use the JDBC connections from this data source. The number of parallel process that PRE will <u>spawn</u> , is determined by the property <u>standalonemaximumthreads</u> . The maximum number of JDBC connections in this data source should be equal <u>standalonemaximumthreads</u> + 1. If the same data source is being used for Group Engine then the maximum JDBC connections should be equal to <u>standalonemaximumthreads</u> + <u>groupmaximumthreads</u> + 2;	ST
dsforgroupeng	String	Data Source Name for the Group Engine. PRE will use the JDBC connections from this data source. The number of parallel process that PRE will <u>spawn</u> is determined by the property <u>groupmaximumthreads</u> . The maximum number of JDBC connections in this data source should be equal <u>groupmaximumthreads</u> + 1. If the same data source is being used for StandAlone Engine then the maximum JDBC connections should be equal to <u>standalonemaximumthreads</u> + <u>groupmaximumthreads</u> + 2;	GRP
standalonemaximumthreads	String	Defines the maximum thread size for standalone request engine.	6
groupmaximumthreads	String	Defines the maximum thread size for grouped request engine.	2
# These properties are for fetching parameters from the database.			
paramdateformat	String	Determines the way the PRE would read and parse the date from the database (Java Format)	MM/dd/yyyy



Item	Data Type	Description	Value
paramtimeformat	String	Determines the way the PRE would read and parse the time from the database (Java Format)	HH:mm:ss
paramdatetimeformat	Character	Defines the format for date time fields. Default defines the above two parameters separated by a space.	\${paramdateformat} \${paramtimeformat}
paramarrayvaluedelimiter	Character	Determines the array value separator	
# These properties are for Request Log that the PRE writes for every request processed.			
requestlogfilepath	String	Directory in which the request log file would be created Note This directory should represent the directory in the application server so that the same file can be accessed through the application.	\${pre.home}/logs/requestfiles/
requestlogfileurl	String	URL by which the request log file can be accessed Note This is to be kept as the actual directory path. Same as property requestlogfilepath	\${pre.home}/logs/requestfiles/
Requestlogfileextension	String	Extension of the Request Log File Default: html	Html
# These properties are for the PRE.			
waitinterval	Number	PRE would wait for that many seconds before scanning the database for any pending requests to be executed Time in seconds 1 Min = 60 * 1000 ms	15
# These properties are the Custom Class Loader details.			
Objclasspathforclassloader	String	Classes from where the PRE should load Should point at the Application Level classes directory	\${pre.home}/classes
Systemloadedclasses	String	List of classes, delimited by a;; that should not be loaded by the custom classloader	stg.;java.;



Item	Data Type	Description	Value
reloadobjclasses	String	Indicates that the Custom Class Loader should take the latest classes files available on the machine At deployment should be changed to N	N
# These properties are for the Scheduling.			
Schedulerequestidgenerator	String	Class that is used to generate the request id for scheduling Class can be changed to any other class that implements the interface stg.pr.engine.scheduler.IRequestIdGenerator.	stg.pr.engine.scheduler.CRequestScheduler
Icalendarinfiniteooperationcount	Integer	This property avoids infinite looping of the scheduling process. If the job's schedule is associated with a class that implements stg.pr.engine.scheduler.ICalendar interface and this goes into an infinite loop and always returns false then the scheduler will terminate the schedule and will come out of the loop after the said counter is reached. In case, the skip flag is SS then for each check (weekday or ICalendar) the counter will be increased by 1. In case the skip flag is D+ or D- then both the checks (weekday and/or ICalendar) will increase the counter by 1.	20
# These properties are for the group PRE.			
groupengine	String	Specifies that the Group PRE would not be started	OFF
# These properties are for the Stuck Thread Monitor			
stuckthreadmonitorinterval	Integer	Indicates that the Thread Monitor would sleep for that many seconds Time is in minutes Note The monitor thread would run only if the value specified is not equal to zero. The absolute value is taken. In other words, if -1 were specified, the value taken would be 1.	2

Item	Data Type	Description	Value
jobmonitorescalationtimeinterval	Integer	Checks jobs at regular interval after they have crossed the Stuck Thread Maximum Limit Note A mail would be escalated after the specified minutes if the job were still alive. Time is in minutes and the default is 30 minutes.	30
# These properties are for the Mailer			
mailnotification	String	Starts the Mail Notification in case of STUCK threads	ON
maildebug	String	Debugs mails N - No Debugging Y - Debugging ON	N
mailtype	String	Default mail type Currently supported is SMTP	SMTP
# Terminate the process if it crosses STUCK THREAD MAX LIMIT. # This does not necessarily kill that process but indicates that the PRE that if the process being executed has implemented interface IProcessTerminator then the terminateProcess () method would be initiated. # The actual termination of the process is left to the implementer.			
terminateprocess	String	Indicates that the PRE would initiate terminateProcess (String Reason) provided the called class implements ITerminateProcess interface	Y
# Web Service			
webservice	String	Web service ON or OFF	ON
webserverport	Long	Web Service port number. Service will use the port specified for deploying web services. In case the port is in use, then the port is auto-incremented by 1 till it finds a port that is free for use.	80
#Current Timestamp.			



Item	Data Type	Description	Value
<p># NET Properties</p> <p># The following properties are from JDK1.4 onwards. These properties would allow the connection time out or read time out for any class that opens the URLConnection. Thus, ensures that the thread is not stuck forever.</p> <p>Note The properties, sun.net.client.defaultConnectionTimeout and sun.net.client.defaultReadTimeout may not be supported in future releases of JDK though they are supported in JDK1.4.x. These properties are SUN implementation specific. However, JDK1.5.0 supports these properties.</p>			
http.keepAlive	String	<p>Indicates if keep alive (persistent) connections should be supported</p> <p>Note Persistent connections improve performance by allowing the underlying socket connection to be reused for multiple http requests.</p> <p>The default is true and persistent connections would be used with http 1.1 servers.</p> <p>It is set to false to disable the use of persistent connections.</p>	True
http.maxConnections	Long	Number of idle connections that would be simultaneously kept alive per destination in case HTTP keep-alive is enabled	5
sun.net.client.defaultConnectionTimeout	Long	<p>Specifies the default connect and read timeout for the protocol handler used by java.net.URLConnection.</p> <p>sun.net.client.defaultConnectionTimeout specifies the timeout (in milliseconds) to establish the connection to the host</p> <p>Example</p> <p>For http connections, it is the timeout when establishing the connection to the http server.</p> <p>For ftp connection, it is the timeout when establishing the connection to ftp servers.</p>	-1



Item	Data Type	Description	Value
sun.net.client.defaultReadTimeout	Long	Specifies the timeout (in milliseconds) while reading from input stream when a connection is established to a resource	-1
#Current Timestamp.			
currenttimestamp	String	<p>If the value is set to DATABASE then the engine will get the current</p> <p>#timestamp from the Database. Otherwise the current timestamp will be</p> <p>#taken from the server on which the PRE is installed.</p> <p>#NOTE that the current Database supported are Oracle, Microsoft SQL</p> <p>#and IBM DB2. To make the PRE independent of the Database do not set</p> <p>#the value for this property.</p>	DATABASE
<p>#PRE SQL Queries for Generating Sequencies.</p> <p>#If there happens to be different logic to generate the Sequence number and/or an SQL can not be executed to generate the sequence number then kindly write a class that implements Stg.pr.engine.scheduler.IRequestIdGenerator and implement your own logic to generate the sequence number. The default class executes the following query based on the JDBC connection type. Refer to jdbctypes.properties for different types of supported Databases. Ensure that ORACLE, DB2 or for that matter any new type is added in jdbctypes.properties file.</p>			
ORACLE.sequence.sql	String	Defines the query for generating sequence with respect to ORACLE database.	SELECT (TO_NUMBER(TO_CHAR(SYSDATE, 'yyyymmdd')) * 10000) + SEQ_INTERFACE_REQ_ID.NEXTVAL FROM DUAL
DB2.sequence.sql	String	Defines the query for generating sequence with respect to IBM DB2 database.	SELECT nextval FOR seq_interface_req_id FROM sysibm.sysdummy1
#PRE SQL query for getting the system date from database provided the property currenttimestamp points to DATABASE.			
ORACLE.timestamp.sql	String	Defines the query for getting the current system timestamp from ORACLE database.	SELECT sysdate FROM dual



Item	Data Type	Description	Value
DB2.timestamp.sql	String	Defines the query for getting the current system timestamp from IBM DB2 database.	SELECT current timestamp FROM sysibm.sysdummy1
MSSQL.timestamp.sql	String	Defines the query for getting the current system timestamp from Microsoft SQL Server	SELECT getdate()
#PRE Reboot Properties.			
rebootmaxcount	Integer	Defines the number of attempts that the PRE should attempt in case of reboots.	5
Sleepbeforeboottime	Integer	The wait time the PRE will wait after the PRE is terminated and before the PRE is started. Time in Minutes.	5
JVM RUNTIME Properties #Any arguments to the JVM such as -Xms or -Xmx or any boothclasspath etc can be set here. The entire value should not be in double quotes. #The default separator is a space. The value that must have a space can be encapsulated in double quotes denoting that it is a single word. #A double quote within a quoted string is ignored.			
Javaruntimevm args	String	Defines the java runtime arguments to the JVM.	-Xms32M -Xmx64M -Dcom.myapp.preimpl.specificKey="Some Value" -Dpre.home=\${pre.home}
Javaextraclasspath	String	Add your own library files that are not within the lib folder.	
Startup Classes. Classes that needs to be loaded during startup. Fully qualified class name separated with semicolon (;) The class must implement stg.pr.engine.Service (and an optional stg.pr.engine.Singleton) interface. The format of the value is key:className where key is optional and can be skipped. If key is skipped then the class name is used as the key to get access to the instance through the PREContext API. #Refer to pre-plugins project			
startUpClasses	String	Key:FullyQualifiedClassName;Key:FullyQualifiedClassName The class must implement Service (and may implement Singleton) interface. Multiple classes can be added using semi-colon (;) as the separator.	ReportGenerator:com.mmpnc.pre.plugins.ReportService



3.2.3 jdbc.type.properties

Item	Data Type	Description	Value
#JDBC Providers.			
Provider	String	Defines all the JDBC providers whose values will be searched in the String returned by executing method <code>getDatabaseProductName()</code> on <code>DatabaseMetaData</code> . One can define any provider on which the PRE needs to be connected. E.g. ORACLE, DB2, MSSQL. A semi-coma separated value.	ORACLE;DB2;MSSQL
#Define the value against each provider defined above.			
ORACLE	String	This value is case sensitive and must match with what is returned by the method <code>getDatabaseProductName()</code> .	Oracle
DB2	String	This value is case sensitive and must match with what is returned by the method <code>getDatabaseProductName()</code> .	DB2
MSSQL	String	This value is case sensitive and must match with what is returned by the method <code>getDatabaseProductName()</code> .	Microsoft SQL Server
#Value to be searched by using method <i>startsWith</i> or <i>endsWith</i> or <i>indexOf</i>			
Method	String	The value is case sensitive and defines the way the PRE will look for a match of the value defined above against each provider.	indexOf

SQL Connections

The Stand Alone Service Engine and Group Thread Service Engine have two independent connection pool mechanisms. The value specified in DBCON property would create DBCON * 2 number of connections.

3.2.4 mail.properties

Item	Data Type	Description	Value
------	-----------	-------------	-------



Item	Data Type	Description	Value
SMTPserver	String	IP Address of the Mail Server or Name of the mail server	
Senderaddress	String	Address of the PRE (FROM)	
normal.recipientTO	String	To Recipients Mail would be sent to these if the Thread is identified as ST	
critical.recipientTO	String	To Recipients Mail would be sent to these if ST crosses MSTL	
normal.recipientsCC	String	CC Recipients Mail would be sent to these if the Thread is identified as ST	
critical.recipientsCC	String	CC Recipients Mail would be sent to these if ST crosses MSTL	
normal.defaultsubject	String	Default subject of the mail formed once the Thread is marked as ST	STUCK THREAD WARNING
critical.defaultsubject	String	Default subject of the mail formed once ST crosses MSTL	STUCK THREAD WARNING CRITICAL!!
normal.defaultmessageheader	String	Default Message Header of the mail formed once the Thread is marked as ST \n can be used to specify a new line character	This message is automatically generated through ProcessRequestEngine. This process has crossed the STUCK Thread Limit specified. If the specified limit is wrong then kindly, ignore this mail.
critical.defaultmessageheader	String	Default Message Header of the mail formed once ST crosses MSTL \n can be used to specify a new line character	This message is automatically generated through ProcessRequestEngine. This process has crossed the maximum limit specified. If the specified limit is wrong then kindly, ignore this mail.



Item	Data Type	Description	Value
normal.defaultmessagefooter	String	Default Message Footer of the mail formed once the Thread is marked as ST \n can be used to specify a new line character	DISCLAIMER: Information contained and transmitted by this E-MAIL is proprietary to the Renaissance System. This is an auto-generated mail intended only for certain privileged users of the system. \n Access to this e-mail by anyone else is unauthorized. Any copying or further distribution beyond the original addressee is not intended.
critical.defaultmessagefooter	String	Default Message Footer of the mail formed once ST crosses MSTL \n can be used to specify a new line character	DISCLAIMER: Information contained and transmitted by this E-MAIL is proprietary to the Renaissance System. This is an auto-generated mail intended only for certain privileged users of the system. \n Access to this e-mail by anyone else is unauthorized. Any copying or further distribution beyond the original addressee is not intended.
reply-to	String	Reply to address if the user replies to the automated email the email will be directed to this email address.	
defaultemailof type	String	Default Email of type HTML or TEXT. Valid values are HTML or TEXT.	TEXT
htmlmessageheaderstarttags	String	Defines the start tags for an HTML message header.	<HTML><BODY>
htmlmessageheaderendtags	String	Defines the HTML Message header end tags.	<HR color=\"blue\" width=\"100%\" size=\"6\"></BODY></HTML>
htmlmessagefooterstarttags	String	Defines the start tags for HTML message footer.	=<HTML><BODY><HR color=\"blue\" width=\"100%\" size=\"6\">
htmlmessagefootere ndtags	String	Defines the HTML message footer end tags.	</BODY></HTML>



3.2.5 system.properties

Use this property file to define all the properties that you want to set as system properties, that can be accessed using `System.getProperty(..)` method. It is not recommended to define the properties that are provided by jvm in this file. The property defined in this file can be accessed as:

```
CSettings.get("system.mypropertykey");
```

Another example: The following property is not defined in the properties file but this is defined by JVM and therefore it can be accessed as:

```
CSettings.get("system.line.separator");
```

Item	Data Type	Description	Value
hazelcast.logging.class	String	Logging class is by default set to No Log. Comment the key to enable the hazelcast logging.	com.hazelcast.logging.NoLogFactory
hazelcast.config	String	The configuration for hazelcast. The file is placed in the resources folder. To change the values within refer to the documentation on the website www.hazelcast.com .	\${pre.home}/resources/hazelcast.xml

3.2.6 poolconfig.xml

These are provided as sample. Refer to the actual release for JDBC Pool for more details.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--JDBC Pool -->
<jdbcpool>
    <pool capacity-increment="1"
        critical-operation-time-limit="1000"
        driver="com.mysql.jdbc.Driver"
        inactive-time-out="6"
        initial-connections="0"
        maximum-capacity="2"
        name="ST"
        password="W1ZJYzh0K3dMQmh3PV1wakpGZkdWaldYWT0="
        shrink-pool-interval="1"
        url="jdbc:mysql://localhost/pre"
        user="root">
```



```
        vendor="MYSQL">
        <in-use-wait-time>7</in-use-wait-time>
        <load-on-startup>false</load-on-startup>
        <max-usage-per-jdbc-connection>-1</max-usage-per-
jdbc-connection>
        <pool-algorithm>FIFO</pool-algorithm>
        <inmemory-statistics-history-size>1</inmemory-
statistics-history-size>
    </pool>

    <pool
        capacity-increment="1"
        critical-operation-time-limit="1000"
        driver="com.mysql.jdbc.Driver"
        inactive-time-out="6"
        initial-connections="0"
        maximum-capacity="2"
        name="GRP"
        password="W2owMHFycUNPdW9nPVlRMGVjNERzVWxNZz0="
        shrink-pool-interval="1"
        url="jdbc:mysql://localhost/pre"
        user="root"
        vendor="MYSQL">
        <in-use-wait-time>7</in-use-wait-time>
        <load-on-startup>false</load-on-startup>
        <max-usage-per-jdbc-connection>-1</max-usage-per-
jdbc-connection>
        <pool-algorithm>FIFO</pool-algorithm>
        <inmemory-statistics-history-size>1</inmemory-
statistics-history-size>
    </pool>

    <jar product="JDBC Pool" releasedate="20080318 12:45:46"
version="16.00.P040">
    <configsaveddate>Fri Mar 21 14:07:19 EDT 2008</configsaveddate>
    <note>If your application is in use then please do not edit
this file manually.</note>
    <note>Any changes made to the file while the application is
active will not have</note>
    <note>any effect on the Pool's configuration and are likely to
be lost. If your</note>
```



<note>application is inactive then you may edit this file with an XML editor. If</note>

<note>you do so then please refer to the JDBC Pools documentation.</note>

</jar>

</jdbcpool>

3.2.7 hazelcast.xml

<hazelcast>

<group>

<name>biling-qc</name>

<password>dev-pass</password>

</group>

<network>

<port auto-increment="true">5701</port>

<join>

<multicast enabled="false">

<multicast-group>224.2.2.3</multicast-group>

<multicast-port>54327</multicast-port>

</multicast>

<tcp-ip enabled="true">

For clustering add multiple interfaces with the IP address of all participating machines.

<interface>172.16.209.93</interface>

</tcp-ip>

</join>

<interfaces enabled="false">

<interface>10.10.1.*</interface>

</interfaces>

<symmetric-encryption enabled="false">

<!--

If you install multiple PRE's working for different database queues on a single server then it is necessary to change the group names to a unique names across these PREs. If it has the same name then all PREs will work in a cluster mode.

Either you can go with multicast or tcp-ip. One of them must be true. Further documentation refer www.hazelcast.com

```

        encryption algorithm such as
        DES/ECB/PKCS5Padding,
        PBEWithMD5AndDES,
        AES/CBC/PKCS5Padding,
        Blowfish,
        DESede
-->
<algorithm>PBEWithMD5AndDES</algorithm>
<!-- salt value to use when generating the secret key -->
<salt>thesalt</salt>
<!-- pass phrase to use when generating the secret key -->
<password>thepass</password>
<!-- iteration count to use when generating the secret key
-->
    <iteration-count>19</iteration-count>
</symmetric-encryption>
<asymmetric-encryption enabled="false">
    <!-- encryption algorithm -->
    <algorithm>RSA/NONE/PKCS1PADDING</algorithm>
    <!-- private key password -->
    <keyPassword>thekeypass</keyPassword>
    <!-- private key alias -->
    <keyAlias>local</keyAlias>
    <!-- key store type -->
    <storeType>JKS</storeType>
    <!-- key store password -->
    <storePassword>thestorepass</storePassword>
    <!-- path to the key store -->
    <storePath>keystore</storePath>
</asymmetric-encryption>
</network>
<executor-service>
    <core-pool-size>16</core-pool-size>
    <max-pool-size>64</max-pool-size>
    <keep-alive-seconds>60</keep-alive-seconds>
</executor-service>
</hazelcast>

```



3.3 Shell Scripts

3.3.1 startEng.sh

The `startEng.sh` shell script is used for starting the engine. The contents of this script are given below:

```
#!/bin/ksh
#
#
SCHEDULER_HOME=@schedulerhome
#Add your specific libraries that are not inside
$SCHEDULER_HOME/lib/folder below.
CLASSPATH="$SCHEDULER_HOME/@jarfile"
for i in `ls -l $SCHEDULER_HOME/lib/*`; do
CLASSPATH=$CLASSPATH:$i; done;
echo CLASSPATH=$CLASSPATH
echo
if [ ! -f "$SCHEDULER_HOME/engine.pid" ]; then
    if [ -f "$SCHEDULER_HOME/console.out" ]; then
        rm -f $SCHEDULER_HOME/console.out
    fi;
    if [ -f "$SCHEDULER_HOME/error.out" ]; then
        rm -f $SCHEDULER_HOME/error.out
    fi;
    echo "Starting the Engine..."
    nohup java -Djava.awt.headless=true -
Dpre.home=$SCHEDULER_HOME -Djava.compiler=NONE -classpath
$CLASSPATH Stg.pr.engine.startstop.CStartEngine
$SCHEDULER_HOME/properties/prinit.properties
$SCHEDULER_HOME/properties/log4j.properties
$SCHEDULER_HOME/properties/poolconfig.xml
1>>$SCHEDULER_HOME/console.out 2>>$SCHEDULER_HOME/error.out &
    echo $! > $SCHEDULER_HOME/engine.pid
    echo "Engine started. Process Id `cat
$SCHEDULER_HOME/engine.pid`"
elif [ `ps -fp `cat $SCHEDULER_HOME/engine.pid` | wc -l` =
2 ]; then
    echo "Engine already started. Process Id: `cat
$SCHEDULER_HOME/engine.pid`"
elif [ `ps -fp `cat $SCHEDULER_HOME/engine.pid` | wc -l` !=
2 ]; then
    if [ -f "$SCHEDULER_HOME/console.out" ]; then
        rm -f $SCHEDULER_HOME/console.out
    fi;
    if [ -f "$SCHEDULER_HOME/error.out" ]; then
        rm -f $SCHEDULER_HOME/error.out
    fi;
    echo "Starting the Engine..."
    nohup java -Djava.awt.headless=true -
Dpre.home=$SCHEDULER_HOME -Djava.compiler=NONE -classpath
$CLASSPATH Stg.pr.engine.startstop.CStartEngine
$SCHEDULER_HOME/properties/prinit.properties
$SCHEDULER_HOME/properties/log4j.properties
```



```

$SCHEDULER_HOME/properties/poolconfig.xml
1>>$SCHEDULER_HOME/console.out 2>>$SCHEDULER_HOME/error.out &
  echo $! > $SCHEDULER_HOME/engine.pid
  echo "Engine started. Process Id `cat
$SCHEDULER_HOME/engine.pid`"
fi;

```

Scheduler Home

Ensure that the variable `SCHEDULER_HOME` changes to the appropriate directory and includes the directory in which the `pre.zip` is extracted.

3.3.2 stopEng.sh

The `stopEng.sh` shell script is used for clean shutdown of the PRE. The PRE would shut itself down, provided there are no currently running processes. If there were processes running then the PRE would not shut itself down unless all processes are completed. The PRE, will not however, service any new requests. To terminate or kill the process immediately, use `killEng.sh`.

The following is the actual shell script:

```

#!/bin/ksh
#
#
SCHEDULER_HOME=@schedulerhome

#Note if the os user id is too big the request may not get
inserted into the queue table. It is then recommended
#to change the user id to a fix value.
USERID=`whoami`

#Add your specific libraries that are not inside
$SCHEDULER_HOME/lib/folder below.
CLASSPATH=$SCHEDULER_HOME/@jarfile
REQTYPE="GENERAL"
for i in `ls -l $SCHEDULER_HOME/lib/*`; do
CLASSPATH=$CLASSPATH:$i; done;
echo CLASSPATH=$CLASSPATH
echo
if [ ! -f "$SCHEDULER_HOME/engine.pid" ]; then
  echo "Process Id not found"
else
  if [ `ps -fp \cat $SCHEDULER_HOME/engine.pid\` | wc -l`
= 2 ]; then
    java -Djava.compiler=NONE -classpath $CLASSPATH
admin.CStopEngine $SCHEDULER_HOME/properties/prinit.properties
$SCHEDULER_HOME/properties/poolconfig.xml $USERID
  else
    echo "Engine not started."
  fi;
fi;

```



The following output would be displayed on the screen once this shell script is executed: [Classpath and the PRE and JDBC Pool version numbers may change based on the release.]

```
CLASSPATH=/home/hlthchk/pre/pre22.02.M033.jar:/home/hlthchk/pre/lib/activation.jar:/home/hlthchk/pre/lib/avalon-framework-4.1.3.jar:/home/hlthchk/pre/lib/batik-awt-util-1.6.jar:/home/hlthchk/pre/lib/batik-dom-1.6.jar:/home/hlthchk/pre/lib/batik-svggen-1.6.jar:/home/hlthchk/pre/lib/batik-util-1.6.jar:/home/hlthchk/pre/lib/batik-xml-1.6.jar:/home/hlthchk/pre/lib/cewolf-0.12.0.jar:/home/hlthchk/pre/lib/commons-codec-1.3.jar:/home/hlthchk/pre/lib/commons-collections-3.1.jar:/home/hlthchk/pre/lib/commons-configuration-1.2.jar:/home/hlthchk/pre/lib/commons-lang-2.1.jar:/home/hlthchk/pre/lib/commons-logging-1.0.4.jar:/home/hlthchk/pre/lib/crimson-1.1.3.jar:/home/hlthchk/pre/lib/gnujasp-0.0.jar:/home/hlthchk/pre/lib/itext-1.4.jar:/home/hlthchk/pre/lib/jcommon-1.0.0-rc1.jar:/home/hlthchk/pre/lib/JDBC Pool15.00.005.jar:/home/hlthchk/pre/lib/jfreechart-1.0.0-rc1.jar:/home/hlthchk/pre/lib/log4j-1.2.12.jar:/home/hlthchk/pre/lib/mail.jar:/home/hlthchk/pre/lib/ojdbc14.jar:/home/hlthchk/pre/lib/RenChart.jar

NOTICE - Product Name: "Advanced Process Request Engine" Version: "22.02.M033" Packaged On "20060606 11:23:18"
NOTICE - Product Name: "JDBC Pool" Version: "15.00.005" Bundled-On "20060426 10:40:55"
NOTICE - Pool Manager created...
NOTICE - STOP request has been sent to PRE. Request Id # 200606074786
NOTICE - STOP request generated by hlthchk
NOTICE - Please tail the console to find whether the engine is terminated.
NOTICE - Please note that the engine will wait till all the current running processes are executed.
```

3.3.3 killEng.sh

The `killEng.sh` shell script is used for killing the PRE. This needs to be avoided. Sending a STOP request to the PRE through the `stopEng.sh` is available. Stopping the PRE through the `stopEng.sh` would help in clean shutdown of the PRE. The PRE would ensure that all the running processes are completely executed before shutting itself.

The shell script is provided below:

```
#!/bin/ksh
#
#
SCHEDULER_HOME=@schedulerhome
if [ ! -f "$SCHEDULER_HOME/engine.pid" ]; then
    echo "Process Id not found"
else
    if [ `ps -fp \cat $SCHEDULER_HOME/engine.pid\` | wc -l` = 2 ]; then
        kill -9 -cat $SCHEDULER_HOME/engine.pid
        echo "Engine Killed."
    else
        echo "Engine not started."
```



```
fi;
fi;
```

3.3.4 truncLog.sh

The truncLog.sh shell script clears the **console.out** file while the PRE is running. This facility is given so that the PRE need not be stopped to clean the console.out file. This file is not a part of the installation but documented here just in case.

The PRE would continue writing messages to the console. The console.out would keep on increasing until the disk capacity is reached. Therefore, it is necessary to clean this file on a regular basis. Alternative way is to stop and then start the engine

The code is given below:

```
#!/bin/ksh
#
#
SCHEDULER_HOME=@schedulerhome
LOG_FILE=console.out
if [ ! -f "$SCHEDULER_HOME/engine.pid" ]; then
    if [ -f "$SCHEDULER_HOME/$LOG_FILE" ]; then
        rm -f $SCHEDULER_HOME/$LOG_FILE
        echo "Engine not started...Removing the existing log
file $SCHEDULER_HOME/$LOG_FILE"
    fi;
elif [ `ps -fp \cat $SCHEDULER_HOME/engine.pid\` | wc -l` =
2 ]; then
    cat /dev/null > $SCHEDULER_HOME/$LOG_FILE
    echo "Engine in running mode. Truncated the
$SCHEDULER_HOME/$LOG_FILE"
elif [ `ps -fp \cat $SCHEDULER_HOME/engine.pid\` | wc -l` !=
2 ]; then
    if [ -f "$SCHEDULER_HOME/$LOG_FILE" ]; then
        rm -f $SCHEDULER_HOME/$LOG_FILE
        echo "Engine is stopped/killed...Removing the existing
log file $SCHEDULER_HOME/$LOG_FILE"
    fi;
fi;
```

LOG_FILE

Ensure that the variable LOG_FILE is changed to the appropriate file name used in the startEng.sh shell script (1>> *console.out*).

3.3.5 bootEng.sh

The bootEng.sh shell script sends a requests to the PRE for doing a reboot. PRE will initiate a physical reboot of itself.

```
#!/bin/ksh
#
#
SCHEDULER_HOME=@schedulerhome
```



```
#Note if the os user id is too big the request may not get
inserted into the queue table. It is then recommended
#to change the user id to a fix value.
USERID=`whoami`
#Add your specific libraries that are not inside
$SCHEDULER_HOME/lib/folder below.
CLASSPATH=$SCHEDULER_HOME/@jarfile
for i in `ls -l $SCHEDULER_HOME/lib/*`; do
CLASSPATH=$CLASSPATH:$i; done;
echo CLASSPATH=$CLASSPATH
echo
if [ ! -f "$SCHEDULER_HOME/engine.pid" ]; then
    echo "Process Id not found"
else
    if [ `ps -fp \`cat $SCHEDULER_HOME/engine.pid\` | wc -l`
= 2 ]; then
        java -Djava.compiler=NONE -classpath $CLASSPATH
admin.CRebootEngine
$SCHEDULER_HOME/properties/prinit.properties
$SCHEDULER_HOME/properties/poolconfig.xml $USERID
    else
        echo "Engine not started."
    fi;
fi;
fi;
```

The following output would be displayed on the screen once this shell script is executed: [Classpath and the PRE and JDBC Pool version numbers may change based on the release.]

```
CLASSPATH=/home/hlthchk/pre/pre22.02.M033.jar:/home/hlthchk/pre/lib/activation.jar:/home/hlthchk/pre/lib/avalon-framework-4.1.3.jar:/home/hlthchk/pre/lib/batik-awt-util-1.6.jar:/home/hlthchk/pre/lib/batik-dom-1.6.jar:/home/hlthchk/pre/lib/batik-svggen-1.6.jar:/home/hlthchk/pre/lib/batik-util-1.6.jar:/home/hlthchk/pre/lib/batik-xml-1.6.jar:/home/hlthchk/pre/lib/cewolf-0.12.0.jar:/home/hlthchk/pre/lib/commons-codec-1.3.jar:/home/hlthchk/pre/lib/commons-collections-3.1.jar:/home/hlthchk/pre/lib/commons-configuration-1.2.jar:/home/hlthchk/pre/lib/commons-lang-2.1.jar:/home/hlthchk/pre/lib/commons-logging-1.0.4.jar:/home/hlthchk/pre/lib/crimson-1.1.3.jar:/home/hlthchk/pre/lib/gnujasp-0.0.jar:/home/hlthchk/pre/lib/itext-1.4.jar:/home/hlthchk/pre/lib/jcommon-1.0.0-rc1.jar:/home/hlthchk/pre/lib/JDBC Pool15.00.005.jar:/home/hlthchk/pre/lib/jfreechart-1.0.0-rc1.jar:/home/hlthchk/pre/lib/log4j-1.2.12.jar:/home/hlthchk/pre/lib/mail.jar:/home/hlthchk/pre/lib/ojdbc14.jar:/home/hlthchk/pre/lib/RenChart.jar

NOTICE - Product Name: "Advanced Process Request Engine" Version:
"22.02.M033" Packaged On "20060606 11:23:18"
NOTICE - Product Name: "JDBC Pool" Version: "15.00.005" Bundled-On
"20060426 10:40:55"
NOTICE - Pool Manager created...
NOTICE - Reboot request has been sent to PRE. Request Id #
200606074787
NOTICE - Reboot request generated by hlthchk
```



NOTICE - Please tail the console to find whether the engine is rebooted.
NOTICE - Please note that the engine will wait till all the current running processes are executed.

3.4 Check the Installation

1. Change the directory to **pre** where the Process Request Engine is installed.
2. On UNIX prompt, execute the shell script **startEng**
3. Output should be similar to Figure 1. The PRE console log is provided in Annexure 5. The PRE log is created by the **debuglevel=4**. The output will not match if the **debuglevel** is not set accordingly.

3.5 Configure *PRE* in Cluster mode



Note

In the current functionality it limits only one PRE to be active. The other PREs configured in cluster mode if started will die provided they are successfully able to ping with one of the PRE configured in cluster mode. This ensures that no two requests are executed by mistake if the admin starts two PREs unknowingly on both or multiple servers.

To configure the PRE in cluster mode ensure that the following:

1. Pool configuration is same for all PREs that will participate in cluster.
2. PRE configured on two or multiple servers have the same PRE version number.
3. Properties **requesttypefilter** and **processrequesttype** from **pr.properties** are same for PREs configured in cluster mode.
4. Property **clusteraddress** from **pr.properties** defines the all the servers on which PREs are installed hostname or ip address followed by the port. Multiple **<iporhostname>:<port>** can be separated with a semi-colon “;”.
5. Property **webservice** defined in **pr.properties** is set to **ON**.
6. Property **webserveriporhostname** defines the IP/Hostname address of the server on which the PRE is hosted.
7. Property **webserverport** defines the port on which the web server will listen.



3.6 Points to Remember

1. If the engine is killed while it is executing some requests, the consequences would have to be manually verified before starting the engine.
2. While the engine is running and executing a process, and there is an end-of-communication channel on the JDBC side, then also the consequences would have to be manually verified before starting the engine.

3.7 More about Logging Messages

The PRE uses log4j for logging. More help can be found at <http://logging.apache.org/log4j/docs/>. The following loggers are defined in the Engine. Each can have the level defined against it. Please see the log4j.properties for additional details about logging. There are 3 new log levels added. These are FINIE, FINER, FINEST in the class `com.Stg.logger.LogLevel`.

| | |
|--|---|
| <code>Log4j.logger.JDBC</code> | INFO[DEBUG, INFO, WARN, FATAL, NOTICE] |
| <code>Log4j.logger.JDBC.ST</code> | [DEUG, INFO, WARN, NOTICE] |
| <code>Log4j.logger.JDBC.ST.SelfCheck</code> | [DEBUG, INFO, WARN, FATAL, NOTICE] |
| <code>Log4j.logger.JDBC.GRP</code> | [DEUG, INFO, WARN, NOTICE] |
| <code>Log4j.logger.JDBC.GRP.SelfCheck</code> | [DEBUG, INFO, WARN, FATAL, NOTICE] |
| <code>Log4j.logger.JDBC.Query</code> | [INFO, WARN] |
| <code>Log4j.logger.Engine</code> | [DEBUG, INFO, WARN, ERROR, FATAL, NOTICE] |
| <code>Log4j.logger.GroupEngine</code> | [DEBUG, INFO, WARN, ERROR, FATAL, NOTICE] |
| <code>Log4j.logger.JobMonitor</code> | [DEBUG, INFO, WARN, ERROR, FATAL, NOTICE] |
| <code>Log4j.logger.StopEngine</code> | [DEBUG, INFO, WARN, ERROR, FATAL, NOTICE] |
| <code>Log4j.loggerMailer</code> | [DEBUG, INFO, WARN, ERROR, FATAL, NOTICE] |
| <code>Log4j.logger.STMPMailer</code> | [DEBUG, INFO, WARN, ERROR, FATAL, NOTICE] |
| <code>Log4j.logger.CustomClassLoader</code> | [DEBUG, INFO] |



| | |
|-----------------------------|---------------|
| Log4j.logger.PREClassLoader | [DEBUG, INFO] |
|-----------------------------|---------------|

**Note**

PRE displays the TIMINGS of JOB execution and PRE internals with the log level as FINE.

**Note**

To use the loggers FINE, FINER and FINEST associate the root logger with any one of the following property in log4j.properties file.

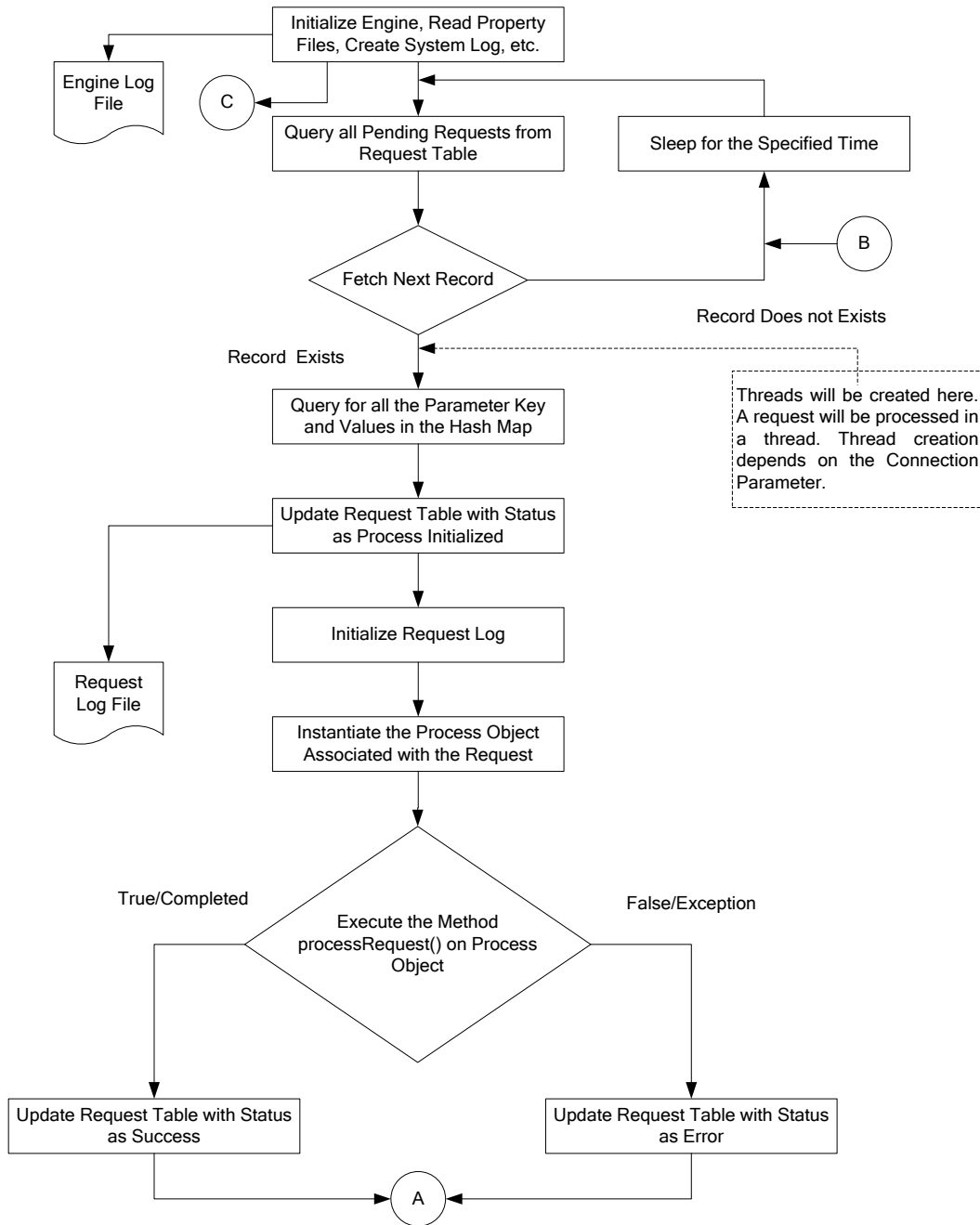
FINE#com.stg.logger.LogLevel

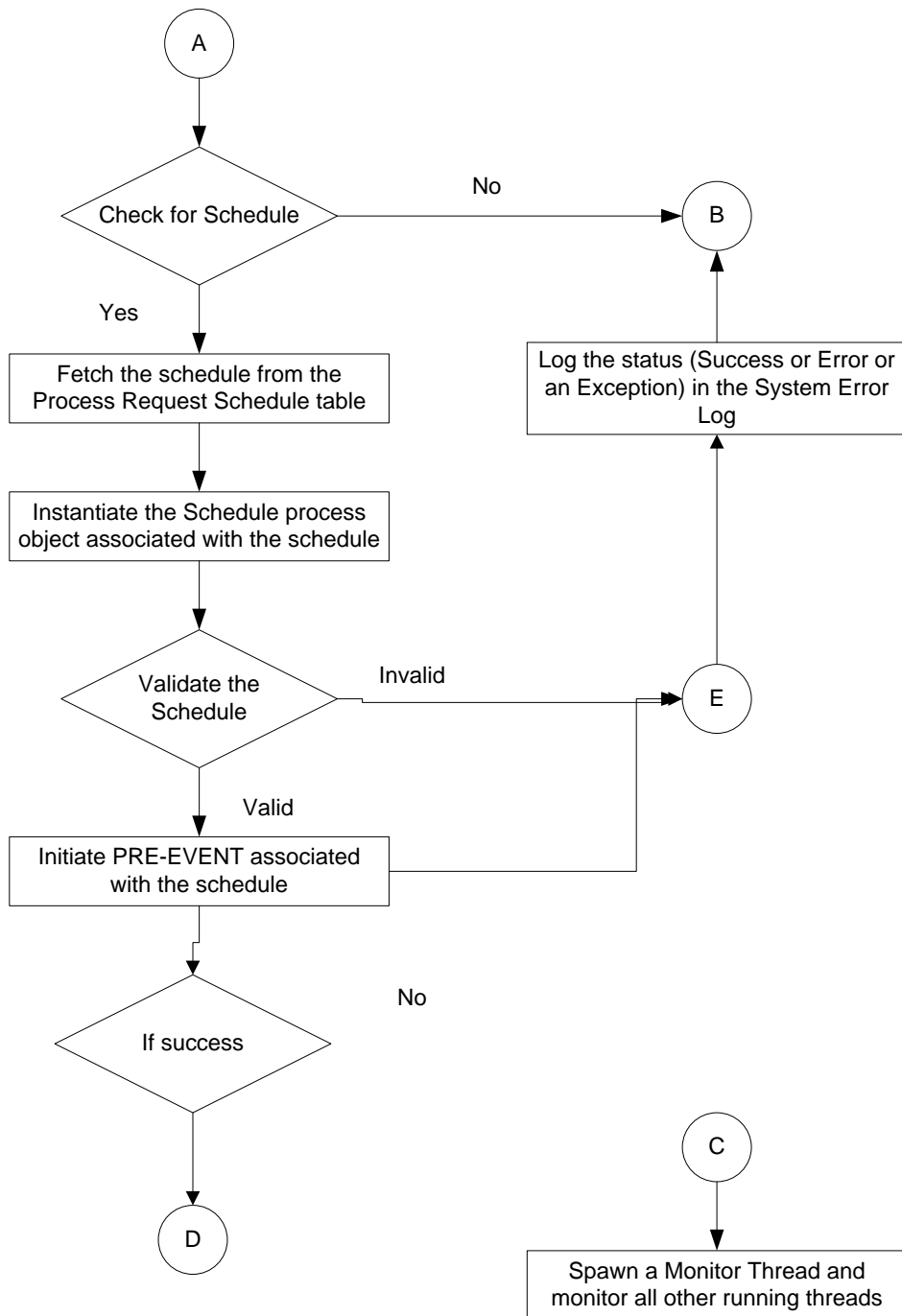
FINER#com.stg.logger.LogLevel

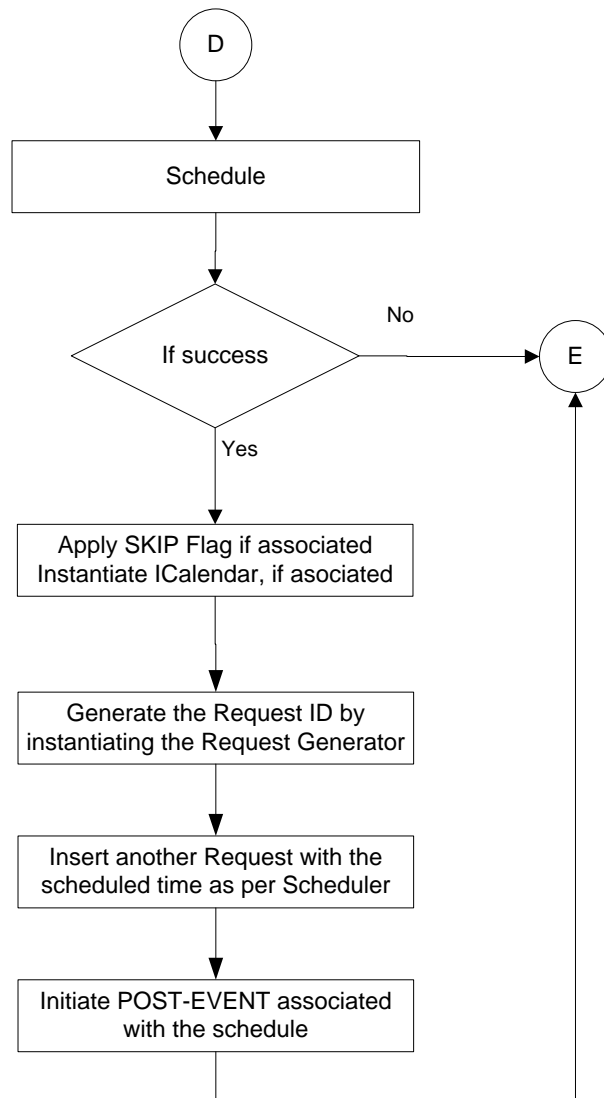
FINEST#com.stg.logger.LogLevel



3.8 PRE Process Flow Diagram







3.9 Sample Screen Shots/ Console Log

Please note that the Version number displayed in the output will depend on the release.

```
<Thursday Jul 10, 2008 16:26:35:159><NOTICE><main><Engine>Product Name:
"Advanced Process Request Engine" Version: "V1.0R24.00.P011" Packaged On
"20080710 16:16:15"

<Thursday Jul 10, 2008 16:26:35:159><INFO><main><Engine>Initializing Engine

<Thursday Jul 10, 2008 16:26:35:175><DEBUG><main><Mailer>Unable to set the CC
Receipient. Cannot accept blank

<Thursday Jul 10, 2008 16:26:35:175><DEBUG><main><Mailer>Getting default
message header

<Thursday Jul 10, 2008 16:26:35:175><DEBUG><main><Mailer>Getting default
message footer

<Thursday Jul 10, 2008 16:26:35:175><DEBUG><main><SMTPMailer>Creating Session

<Thursday Jul 10, 2008 16:26:35:175><DEBUG><main><SMTPMailer>No mail
authenticator specified.

<Thursday Jul 10, 2008 16:26:35:206><DEBUG><main><SMTPMailer>Adding Message
Headers, if any.

<Thursday Jul 10, 2008 16:26:35:206><DEBUG><main><SMTPMailer>Message headers
added...#1

<Thursday Jul 10, 2008 16:26:35:237><DEBUG><main><SMTPMailer>Transporting E-
Mail to the Mail Server...

<Thursday Jul 10, 2008 16:26:35:237><DEBUG><main><SMTPMailer>FROM
[mypersonal.pre@stgil-india.com] TO [kedar.raybagkar@stgil.com], CC [], BCC []
Subject [Engine is starting...]

<Thursday Jul 10, 2008 16:26:35:565><DEBUG><main><SMTPMailer>...E-Mail
Transported

<Thursday Jul 10, 2008 16:26:35:565><INFO><main><Engine>Add ShutDown Hook ....

<Thursday Jul 10, 2008 16:26:35:565><INFO><main><Engine>ShutDown Hook Added....

<Thursday Jul 10, 2008 16:26:35:565><INFO><main><Engine>Initializing Connection
Pool ....

<Thursday Jul 10, 2008 16:26:35:581><NOTICE><main><PoolManager>Product Name:
"JDBC Pool" Version: "16.00.P040" Bundled-On "20080318 12:45:46"

<Thursday Jul 10, 2008
16:26:35:612><DEBUG><main><org.apache.commons.configuration.ConfigurationUtils>
ConfigurationUtils.locate(): base is c:\pre\properties, name is poolconfig.xml

<Thursday Jul 10, 2008
16:26:35:627><DEBUG><main><org.apache.commons.configuration.ConfigurationUtils>
Loading configuration from the path c:\pre\properties\poolconfig.xml

<Thursday Jul 10, 2008 16:26:35:690><DEBUG><main><jdbc.pool.CXMLManager>Reading
attributes for pool #ST

<Thursday Jul 10, 2008 16:26:35:940><DEBUG><main><jdbc.pool.CXMLManager>Pool
Attributes read for pool #ST

<Thursday Jul 10, 2008 16:26:35:940><DEBUG><main><jdbc.pool.CXMLManager>Reading
attributes for pool #GRP

<Thursday Jul 10, 2008 16:26:35:955><DEBUG><main><jdbc.pool.CXMLManager>Pool
Attributes read for pool #GRP

<Thursday Jul 10, 2008 16:26:35:955><INFO><main><PoolManager>Intializing Pool
Manager...

<Thursday Jul 10, 2008 16:26:35:955><INFO><main><PoolManager>Creating start up
Pools...
```



```
<Thursday Jul 10, 2008 16:26:35:955><DEBUG><main><PoolManager>Pool Attributes
loaded Pool Name,GRP, Initial Pool Size,0,, Capacity Increament,1, Maximum
Capacity,2, Idle Timeout,6, Critical Operation Time Limit,1000, Self Check
Interval,1, Wait Interval,7, URL,jdbc:mysql://localhost/pre, Vendor,MYSQL, User
Id,root, Password,*****

<Thursday Jul 10, 2008 16:26:35:955><DEBUG><main><PoolManager>Pool Attributes
loaded Pool Name,ST, Initial Pool Size,0,, Capacity Increament,1, Maximum
Capacity,2, Idle Timeout,6, Critical Operation Time Limit,1000, Self Check
Interval,-1, Wait Interval,7, URL,jdbc:mysql://localhost/pre, Vendor,MYSQL,
User Id,root, Password,*****

<Thursday Jul 10, 2008 16:26:35:971><NOTICE><main><PoolManager>Pool Manager
created...

<Thursday Jul 10, 2008 16:26:35:971><NOTICE><main><Engine>Starting Web Services

<Thursday Jul 10, 2008 16:26:36:002><INFO><Thread-2><WebServer>Acquiring Port
#15000

<Thursday Jul 10, 2008 16:26:36:002><NOTICE><Thread-2><WebServer>Web Server
Started and is listening on 15000

<Thursday Jul 10, 2008 16:26:36:002><INFO><main><Engine>Engine Initialized

<Thursday Jul 10, 2008 16:26:36:002><INFO><main><Engine>Starting the Engine..

<Thursday Jul 10, 2008 16:26:36:002><INFO><main><Engine>Initializing JOB
Monitor...

<Thursday Jul 10, 2008 16:26:36:018><NOTICE><main><Engine>Start Service Engine
for Stand Alone Requests....

<Thursday Jul 10, 2008 16:26:36:018><INFO><main><Engine>Starting StandAlone
Engine ....

<Thursday Jul 10, 2008 16:26:36:018><INFO><main><Engine>Getting JDBC Connection
for the StandAlone Engine ....

<Thursday Jul 10, 2008 16:26:36:018><DEBUG><main><PoolManager>Getting
connection from Pool #ST

<Thursday Jul 10, 2008 16:26:36:018><INFO><main><PoolManager>Creating Pool ST

<Thursday Jul 10, 2008 16:26:36:018><INFO><main><JDBC.Pool[ST]>Using First-In-
First-Out Algorithm

<Thursday Jul 10, 2008 16:26:36:018><INFO><grpeng><Engine>Starting the Group
Engine

<Thursday Jul 10, 2008 16:26:36:018><INFO><grpeng><Engine>Starting Group
Service Engine...

<Thursday Jul 10, 2008 16:26:36:018><INFO><grpeng><Engine>Waiting for the
cluster handshake.

<Thursday Jul 10, 2008 16:26:36:018><NOTICE><Thread-3><JobMonitor>Started...

<Thursday Jul 10, 2008 16:26:36:033><INFO><Thread-3><JobMonitor>Will sleep for
2 minute(s).

<Thursday Jul 10, 2008 16:26:36:174><DEBUG><main><JDBC.Pool[ST]>Attempting
number of #0 new JDBC Connections

<Thursday Jul 10, 2008 16:26:36:174><DEBUG><main><JDBC.Pool[ST]>Initiate Self
Check background process

<Thursday Jul 10, 2008 16:26:36:174><INFO><main><JDBC.Pool[ST]>Pool will not be
shrunked nor self checks will be initiated.

<Thursday Jul 10, 2008 16:26:36:174><NOTICE><main><JDBC.Pool[ST]>Pool Created
and initialized. Statistics Pool Started On,1215721596174, Connections High,0,
Leaked ResultSet,0, Leaked Statements,0, Leaked Connections,0, Current Used
Connections,0, Current Free Connections,0, Bad Connections Found,0, Current
Waiters,0, Waiters High,0, Wait Time High,0, Total Wait Time,0, Average
Connection Delay,0, Connections Total,0, Unavailable Connections,0, Unavailable
Connections High,0, Unavailable Connections High Time,0, Used Memory,4843304,
Total Memory,5177344

<Thursday Jul 10, 2008 16:26:36:189><DEBUG><main><PoolManager>Getting
connection from Pool #ST
```



Design / Installation Document

```
<Thursday Jul 10, 2008 16:26:36:189><DEBUG><main><JDBC.Pool[ST]>Attempting
number of #1 new JDBC Connections

<Thursday Jul 10, 2008 16:26:36:455><DEBUG><main><JDBC.Pool[ST]>New Connection
Created #com.mysql.jdbc.Connection@11a64ed

<Thursday Jul 10, 2008 16:26:36:455><DEBUG><main><JDBC.Pool[ST]>Adding
Connection to the pool #com.mysql.jdbc.Connection@11a64ed

<Thursday Jul 10, 2008 16:26:36:455><DEBUG><main><JDBC.Pool[ST]>Validating the
Connection object.

<Thursday Jul 10, 2008 16:26:36:486><DEBUG><main><JDBC.Pool[ST]>Connection
found OK.

<Thursday Jul 10, 2008 16:26:36:642><NOTICE><main><Engine>Doing handshake with
the cluster PRE.. Address:Port#148.96.103.148:15000

<Thursday Jul 10, 2008 16:26:36:673><INFO><main><Engine>Handshake initiated...

<Thursday Jul 10, 2008 16:26:57:670><ERROR><main><Engine>ConnectException.
Clustered PRE not up.

<Thursday Jul 10, 2008 16:26:57:670><NOTICE><main><Engine>Unable to do the
handshake with the clustered PRE.

<Thursday Jul 10, 2008 16:26:57:670><NOTICE><main><Engine>Stand Alone Engine
Started..

<Thursday Jul 10, 2008 16:26:57:670><DEBUG><main><Engine>Entered infintite
loop, Initializing Request Entity Bean ....

<Thursday Jul 10, 2008 16:26:57:686><DEBUG><grpeng><Engine>Getting Connection
from the pool ....

<Thursday Jul 10, 2008 16:26:57:686><DEBUG><grpeng><PoolManager>Getting
connection from Pool #GRP

<Thursday Jul 10, 2008 16:26:57:686><INFO><grpeng><PoolManager>Creating Pool
GRP

<Thursday Jul 10, 2008 16:26:57:686><INFO><grpeng><JDBC.Pool[GRP]>Using First-
In-First-Out Algorithm

<Thursday Jul 10, 2008 16:26:57:686><DEBUG><grpeng><JDBC.Pool[GRP]>Attempting
number of #0 new JDBC Connections

<Thursday Jul 10, 2008 16:26:57:686><DEBUG><grpeng><JDBC.Pool[GRP]>Initiate
Self Check background process

<Thursday Jul 10, 2008 16:26:57:686><NOTICE><grpeng><JDBC.Pool[GRP]>Pool
Created and initialized. Statistics Pool Started On,1215721617686, Connections
High,0, Leaked ResultSet,0, Leaked Statements,0, Leaked Connections,0, Current
Used Connections,0, Current Free Connections,0, Bad Connections Found,0,
Current Waiters,0, Waiters High,0, Wait Time High,0, Total Wait Time,0, Average
Connection Delay,0, Connections Total,0, Unavailable Connections,0, Unavailable
Connections High,0, Unavailable Connections High Time,0, Used Memory,3804352,
Total Memory,6475776,[ST]Connections Requested,0,[ST]New Connections
Requested,1,[ST]Connections Created,0,[ST]Connections Locked,0,[ST]Connections
UnLocked,0,[ST]Connections Closed,0

<Thursday Jul 10, 2008 16:26:57:717><DEBUG><grpeng><PoolManager>Getting
connection from Pool #GRP

<Thursday Jul 10, 2008 16:26:57:717><DEBUG><grpeng><JDBC.Pool[GRP]>Attempting
number of #1 new JDBC Connections

<Thursday Jul 10, 2008 16:26:57:686><INFO><Thread-
6><JDBC.Pool[GRP].SelfCheck>Background self check process started...

<Thursday Jul 10, 2008 16:26:57:733><DEBUG><Thread-
6><JDBC.Pool[GRP].SelfCheck>In Sleep mode and will sleep for 1 minute(s).

<Thursday Jul 10, 2008 16:26:57:748><DEBUG><grpeng><JDBC.Pool[GRP]>New
Connection Created #com.mysql.jdbc.Connection@15e9756

<Thursday Jul 10, 2008 16:26:57:748><DEBUG><grpeng><JDBC.Pool[GRP]>Adding
Connection to the pool #com.mysql.jdbc.Connection@15e9756

<Thursday Jul 10, 2008 16:26:57:748><DEBUG><grpeng><JDBC.Pool[GRP]>Validating
the Connection object.
```



```
<Thursday Jul 10, 2008 16:26:57:748><DEBUG><grpeng><JDBC.Pool[GRP]>Connection
found OK.

<Thursday Jul 10, 2008 16:26:57:748><DEBUG><grpeng><JDBC.Pool[GRP]>Creating a
new JDBC statement

<Thursday Jul 10, 2008 16:26:57:764><DEBUG><grpeng><JDBC.Pool[GRP]>Creating a
new JDBC statement

<Thursday Jul 10, 2008 16:26:57:764><NOTICE><grpeng><Engine>Group Service
Engine Started....

<Thursday Jul 10, 2008 16:26:57:780><INFO><grpeng><JDBC.Pool[GRP].Query>Con
#com.mysql.jdbc.Connection@15e9756 Time: 16 milis. [SELECT NOW()]

<Thursday Jul 10, 2008 16:26:57:780><INFO><main><JDBC.Pool[ST].Query>Con
#com.mysql.jdbc.Connection@11a64ed Time: 78 milis. [SELECT NOW()]

<Thursday Jul 10, 2008 16:26:57:795><DEBUG><main><JDBC.Pool[ST]>Explicit
close() called on statement. Closing all open ResultSets, if any.

<Thursday Jul 10, 2008 16:26:57:795><DEBUG><grpeng><JDBC.Pool[GRP]>Explicit
close() called on statement. Closing all open ResultSets, if any.

<Thursday Jul 10, 2008 16:26:57:795><DEBUG><main><Engine>Building query ....

<Thursday Jul 10, 2008 16:26:57:795><DEBUG><main><Engine>Querying for queued
requests ....

<Thursday Jul 10, 2008 16:26:57:795><DEBUG><main><JDBC.Pool[ST]>Creating a new
JDBC statement

<Thursday Jul 10, 2008 16:26:57:842><INFO><grpeng><JDBC.Pool[GRP].Query>Con
#com.mysql.jdbc.Connection@15e9756 Time: 47 milis. [Select distinct grp_id from
process_request where grp_st_ind = ? and req_stat = ? and scheduled_time <= ?]

<Thursday Jul 10, 2008 16:26:57:842><INFO><grpeng><Engine>Queued requests does
not exist, Group service will sleep for 15 seconds and scanning will re-
start....

<Thursday Jul 10, 2008 16:26:57:842><INFO><main><JDBC.Pool[ST].Query>Con
#com.mysql.jdbc.Connection@11a64ed Time: 47 milis. [Select scheduled_time,
req_id, sch_id, user_id, grp_st_ind, job_id, job_name, email_ids, req_end_dt,
req_start_dt, process_class_nm, stuck_thread_limit, stuck_thread_max_limit,
req_dt, req_stat, grp_id, grp_req_seq_no, req_logfile_nm, req_type,
verbose time elapsed, cal_scheduled_time, priority From process_request WHERE
scheduled_time <= ? AND grp_st_ind = ? AND req_stat = ? AND req_type IN
('GENERAL') ORDER BY priority]

<Thursday Jul 10, 2008 16:26:57:842><DEBUG><main><JDBC.Pool[ST]>Explicit
close() called on statement. Closing all open ResultSets, if any.

<Thursday Jul 10, 2008 16:26:57:842><INFO><main><Engine>Queued requests does
not exist, StandAlone service will sleep for 15 seconds and scanning will re-
start ....

<Thursday Jul 10, 2008 16:27:12:829><INFO><grpeng><JDBC.Pool[GRP].Query>Con
#com.mysql.jdbc.Connection@15e9756 Time: 0 milis. [SELECT NOW()]

<Thursday Jul 10, 2008 16:27:12:829><DEBUG><grpeng><JDBC.Pool[GRP]>Explicit
close() called on statement. Closing all open ResultSets, if any.

<Thursday Jul 10, 2008 16:27:12:829><DEBUG><main><Engine>Entered infinitite
loop, Initializing Request Entity Bean ....

<Thursday Jul 10, 2008 16:27:12:844><INFO><grpeng><JDBC.Pool[GRP].Query>Con
#com.mysql.jdbc.Connection@15e9756 Time: 15 milis. [Select distinct grp_id from
process_request where grp_st_ind = ? and req_stat = ? and scheduled_time <= ?]

<Thursday Jul 10, 2008 16:27:12:844><INFO><grpeng><Engine>Queued requests does
not exist, Group service will sleep for 15 seconds and scanning will re-
start....

<Thursday Jul 10, 2008 16:27:12:844><INFO><main><JDBC.Pool[ST].Query>Con
#com.mysql.jdbc.Connection@11a64ed Time: 15 milis. [SELECT NOW()]

<Thursday Jul 10, 2008 16:27:12:844><DEBUG><main><JDBC.Pool[ST]>Explicit
close() called on statement. Closing all open ResultSets, if any.

<Thursday Jul 10, 2008 16:27:12:844><DEBUG><main><Engine>Building query ....
```



Design / Installation Document

```

<Thursday Jul 10, 2008 16:27:12:844><DEBUG><main><Engine>Querying for queued
requests ....

<Thursday Jul 10, 2008 16:27:12:844><DEBUG><main><JDBC.Pool[ST]>Creating a new
JDBC statement

<Thursday Jul 10, 2008 16:27:12:844><INFO><main><JDBC.Pool[ST].Query>Con
#com.mysql.jdbc.Connection@11a64ed Time: 0 milis. [Select scheduled_time,
req_id, sch_id, user_id, grp_st_ind, job_id, job_name, email_ids, req_end_dt,
req start dt, process class nm, stuck thread limit, stuck thread max_limit,
req dt, req stat, grp id, grp req seq no, req logfile nm, req type,
verbose time elapsed, cal_scheduled_time, priority From process request WHERE
scheduled_time <= ? AND grp_st_ind = ? AND req_stat = ? AND req_type IN
('GENERAL') ORDER BY priority]

<Thursday Jul 10, 2008 16:27:12:844><DEBUG><main><JDBC.Pool[ST]>Explicit
close() called on statement. Closing all open ResultSets, if any.

<Thursday Jul 10, 2008 16:27:12:844><INFO><main><Engine>Queued requests does
not exist, StandAlone service will sleep for 15 seconds and scanning will re-
start...

```

Figure 1 console.out, the PRE Log

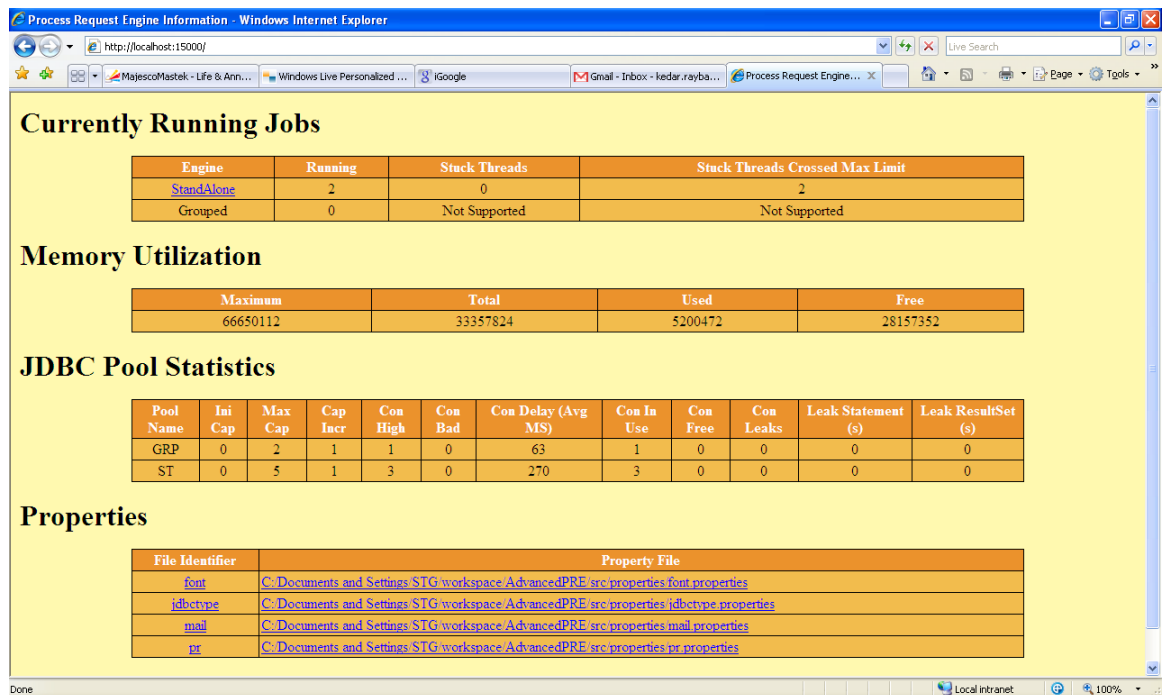


Figure 2 Web Service Monitoring

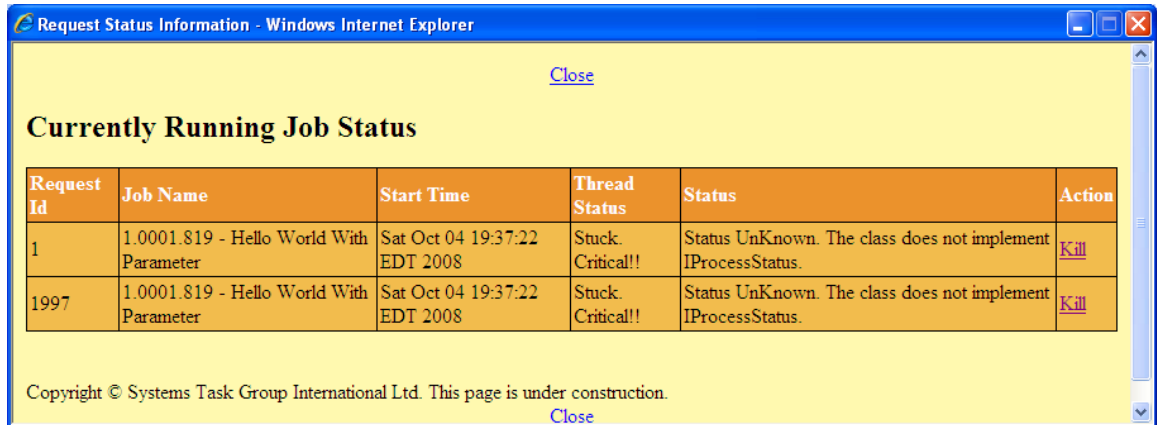


Figure 3 Running Process Status

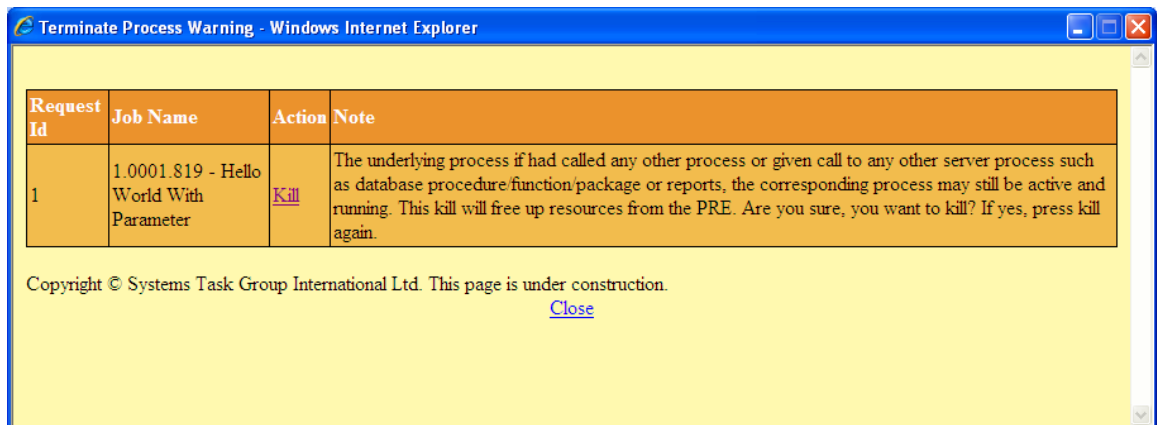


Figure 4 Terminate Process Warning

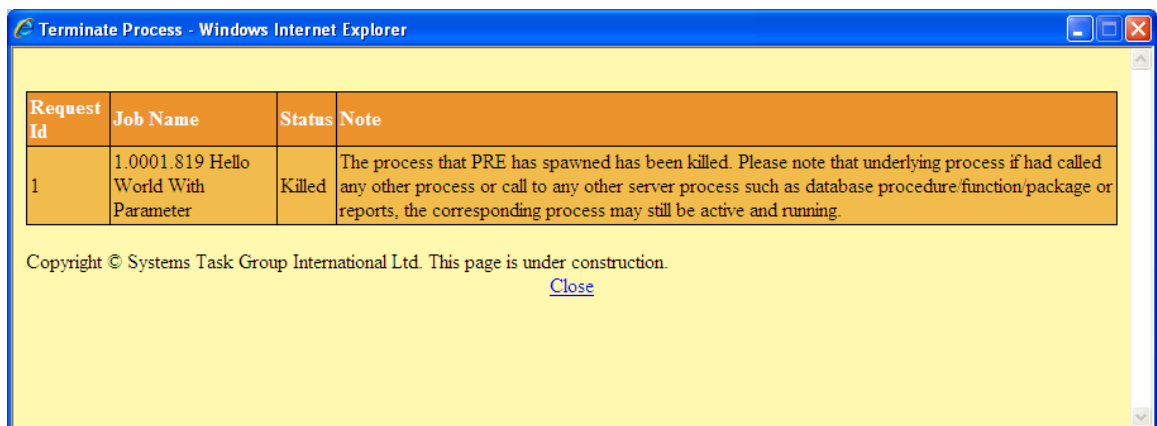


Figure 5 Terminate Process



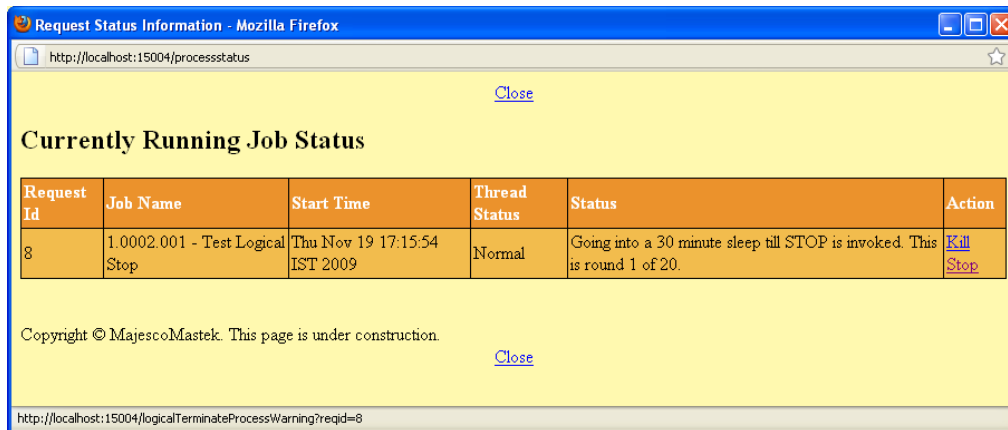


Figure 6 Stop Feature has been added

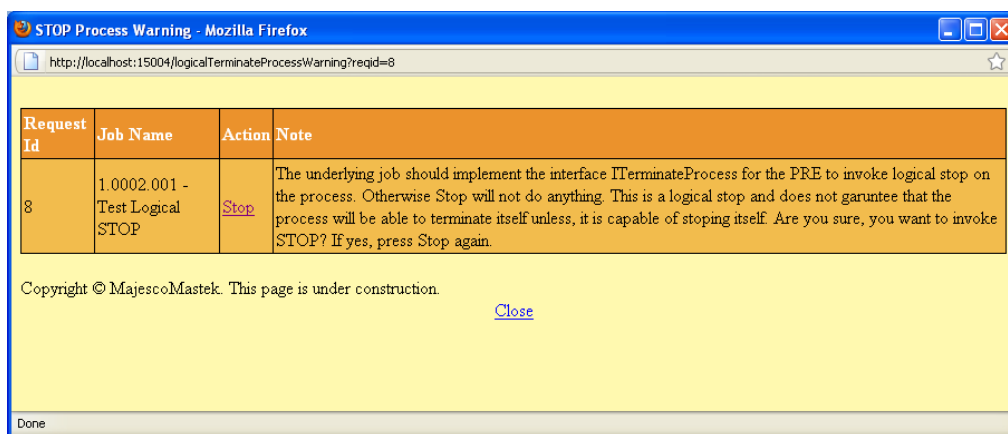


Figure 7 Stop Warning is displayed

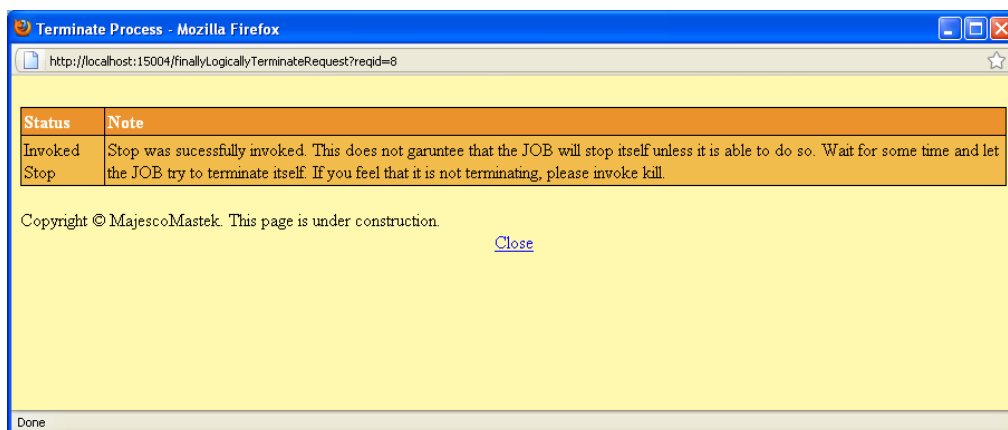


Figure 8 Message for successfully invoking stop

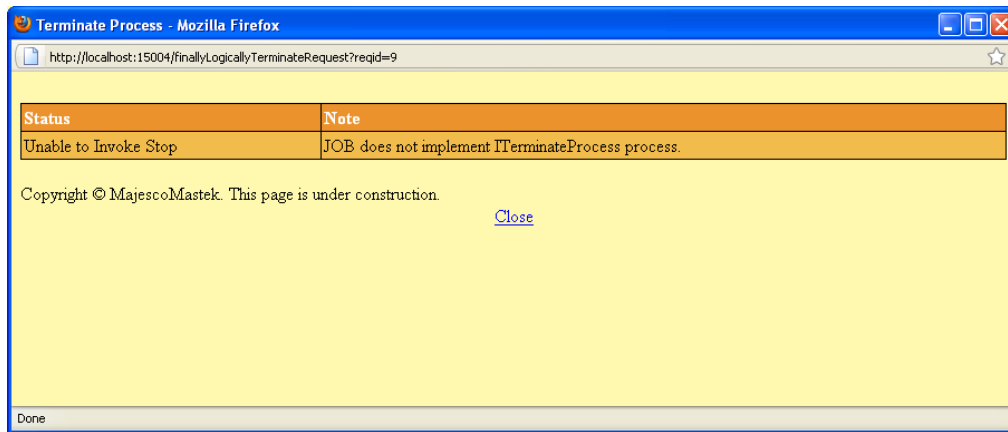


Figure 9 Message if invocation of stop is un-successful



3.10 Certified On

| Server | OS (with version) | JVM Vendor | JVM Version |
|---------------------------------|-------------------|-------------|-----------------------------------|
| Windows | XP SP 2 | SUN JDK | 1.4.x, 1.5.x, 1.6.x |
| Sun Solaris E 450 | Solaris 5.9 | SUN JDK | 1.3.x, 1.4.x, 1.5.x |
| GNU/Linux (Red-Hat 2.6.9-22.EL) | Linux 2.6.9-22.EL | HotSpot(TM) | 1.4.x (Standard Edition) |
| AIX | AIX 3 5 | | 1.4.x |
| Linux | Linux 2.6.9-22 | HotSpot(TM) | 1.4.x (Standard Edition), 1.6.x |
| SunOS | Solaris 5.8 | Solaris_JDK | 1.2.2, 1.6.x |
| Sun Solaris E 25 K | Solaris 5.9 | SUN JRE | 1.4.x, 1.6.x |
| | | | |

| Database | OS (with version) | JVM Vendor | JVM Version |
|---------------------|-----------------------|-------------|----------------------------|
| Oracle 7 and above. | Refer to table above. | | |
| DB2 | Solaris 5.9 | SUN JDK | 1.3.x, 1.4.x, 1.5.x, 1.6.x |
| MySQL | Windows | HotSpot(TM) | 1.4.x, 1.5x |
| MSSQL | Windows | | 1.4.x |
| | | | |



3.11 Known Issues

Operating system: LINUX

It has been found that PRE does not work with the following JVM version on LINUX.

java version "1.4.2"
gcj (GCC) 3.4.4 20050721 (Red Hat 3.4.4-2)
Copyright (C) 2004 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.S

It needs a standard edition of the JVM as mentioned below:

java version "1.4.2_10"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_10-b03)
Java HotSpot(TM) Client VM (build 1.4.2_10-b03, mixed mode)

It has been found that PRE does not work with the following JVM version on LINUX.

java version "1.4.2"
gcj (GCC) 3.4.6 20060404 (Red Hat 3.4.6-3.1)
Copyright (C) 2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.



4 Libraries Used

| Library | Type | Vendor | Comments | Acknowledgement / License Information |
|---|----------------|------------------|---|---|
| bcprov-jdk16-
<x>.jar | Open
Source | Bouncy
Castle | Used for encryption. | http://bouncycastle.org/licence.html |
| commons-codec-
<x>.jar | Open
Source | Apache | General encoding/decoding algorithms (for example phonetic, base64, URL). | http://www.apache.org/licenses/ |
| commons-
collections- | Open
Source | Apache | Extends or augments the Java Collections | http://www.apache.org/licenses/ |

| Library | Type | Vendor | Comments | Acknowledgement / License Information |
|--|-------------|--------------------|--|--|
| <x>.jar | | | Framework. | |
| commons-configuration-<x>.jar | Open Source | Apache | Reading of configuration/preferences files in various formats. | http://www.apache.org/licenses/ |
| commons-lang-<x>.jar | Open Source | Apache | Provides extra functionality for classes in java.lang. | http://www.apache.org/licenses/ |
| commons-logging.jar | Open Source | Apache | Wrapper around a variety of logging API implementations. | http://www.apache.org/licenses/ |
| coreapi.jar | Open Source | Eclipse Foundation | Business Intelligence and Reporting Tools | http://eclipse.org/birt/phoenix/ |
| engineapi.jar | Open Source | Eclipse Foundation | Business Intelligence and Reporting Tools | http://eclipse.org/birt/phoenix/ |
| hazelcast-x.jar | Open Source | Hazelcast | Distributed caching and Clustering | <i>This product includes software developed by the Hazelcast Project (http://www.hazelcast.com).</i>
License http://www.apache.org/licenses/ |
| JDBCPOOL-<x>.jar | Proprietary | Mastek | JDBC Pool library | In case the application pool is to be used then you need to implement the <code>IDataSourceFactory</code> Interface. The default implementation uses the JDBCPOOL. |



| Library | Type | Vendor | Comments | Acknowledgement / License Information |
|---|-------------|--------------------|---|--|
| log4j-<x>.jar | Open Source | Apache log4j | Logging services. | http://logging.apache.org/log4j/1.2/license.html |
| modelapi.jar | Open Source | Eclipse Foundation | Business Intelligence and Reporting Tools | http://eclipse.org/birt/phenix/ |
| mysql-connector-java-<x>-bin.jar | Open Source | MySQL | MySQL JDBC Driver | http://www.oracle.com/technetwork/licenses/distribution-license-152002.html |
| Ojdbc<x>.jar | Open Source | Oracle | Oracle JDBC Driver | OTN license
http://www.oracle.com/technetwork/licenses/distribution-license-152002.html |
| stg-birt-report-<x>.jar | Proprietary | Mastek | Library to execute BIRT reports. | The setting can be turned off by changing the property <code>reportService</code> pr.properties in case not needed. |
| stg-commons-<x>.jar | Proprietary | Mastek | Common library for common routines. | Some common routines used by many other products. |

