Oracle Utilities Customer Care And Billing

Batch Operations and Configuration Guide

Ver 2.3.1

E18372-01

August 2010

Oracle Utilities Customer Care And Billing Batch Operations and Configuration Guide, Ver 2.3.1

E18372-01

Table of Contents

Contents

Introduction

Welcome to the Oracle Utilities Customer Care And Billing Batch Operations and Configuration Guide. This guide outlines the concepts applicable to operating and configuration of the batch component of the product on its platforms in association with the operations and configuration steps outlined in the Oracle Utilities Customer Care And Billing Operations and Configuration Guide. It is highly recommended that readers of this guide familiarize themselves with that guide before reading this guide.

Note: All examples and screen captures are used for publishing purposes only and may vary from the actual values seen at your site.

Note: All utilities in this guide are multi-platform (unless otherwise indicated). For publishing purposes the commands will be in the format **`command[.sh]`** which indicates that the command can be used as "**`command`**" on the Windows platform or "**`command.sh`**" on the UNIX/Linux platforms.

Note: The Batch component of the may not be applicable to all Oracle Tax and Utilities Global Business Unit products. Refer to the provided user and framework documentation for clarification.

Note: This document includes documentation of fixes up to an including Oracle Utilities Application Framework Service Pack 6. Any new material is marked with NEW. Individual fixes are listed in the documentation as required.

Note: For publishing purposes, Oracle Utilities Customer Care And Billing will be referred to as "product".

Contents

Updates to this documentation

This documentation is provided with the version of the product indicated. Additional and updated information about the operations and configuration of the product is available from the Knowledge Base section of ORACLE MetaLink (http://metalink.oracle.com). Please refer to MetaLink for more information.

Other documentation available

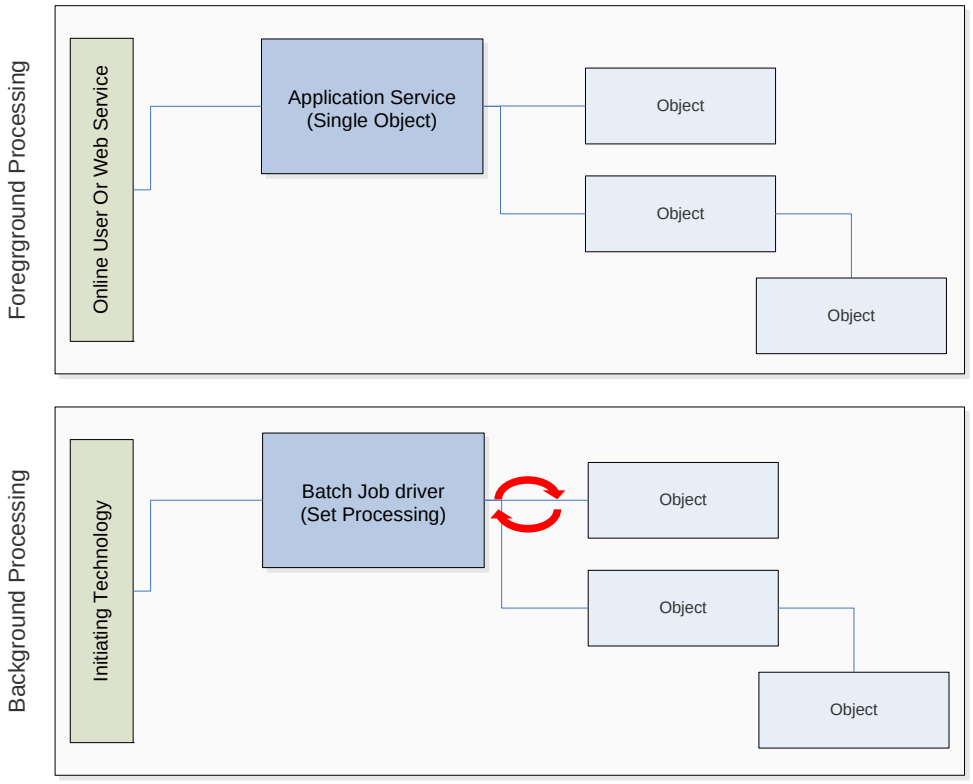
This document is part of the product technical documentation. There are groups of manuals that need to be also read for additional specific advice and information:

Title	Content Summary	Location
Oracle Utilities Customer Care And Billing Installation Guide	Installation instructions for the product	http://edelivery.oracle.com
Oracle Utilities Customer Care And Billing Quick Installation Guide	Licensing and installation overview for the product	http://edelivery.oracle.com
Oracle Utilities Customer Care And Billing Operations and Configuration Guide	General Configuration and Operations for the product	http://edelivery.oracle.com

Batch Architecture

The product is known for its online (or foreground) processing (a.k.a. "online" processing) but one of the major features of the product is its set of background processes. Background processing is a major part of the product with numerous background processes supplied as "standard".

The easiest way to understand the concept behind background processing is to think that background processing is like a "super efficient user" that operates on a batch of objects. That is why background Processing is commonly called "Batch". Online typically operates on one object at a time, initiated by an online user or a Web Service call, where batch can operate on one or more objects (also known as a set of objects) at a time, initiated using a number of technologies.



The main reasoning behind the "super efficient user" is that each background process consists of a driver object that identifies the set of valid objects to process and then processes each object through the same business objects that the online uses. For example, the BILLING driver determines which accounts are eligible to be billed according to business calendar and then passes each account to the rate object to produce a bill. Contrast this with online bill generation, where the user identifies the account manually, and then that single account is passed to the same rate object to be billed. The background process can call more than one object during the duration of the background process.

For the batch process, all of the database access and object access (including access to both COBOL and Java business objects, algorithms, user exits (server side only) etc is done through the Oracle Utilities Application Framework.

Background processing is an integral part of the operation of the product.

Background processing and the Architecture

The Background Processing component is run within the Oracle Utilities Application Framework and is associated typically with the Business Application Server. It is not associated the Web Application Server and does not require the Web Application Server to be active to operate. The only component other than product that the background processing component requires is the database server (or tier).

Depending on the initiation method employed the background processing component uses a standalone copy of the Oracle Utilities Application Framework to perform access to the database and business objects and its own copy of the same business objects used by the Business Application Server.

Essentially the background processing has its own resources (Java Virtual machines (JVMs), connection pools) independent of the rest of the architecture and can therefore be run on the same hardware as the rest of the architecture or on dedicated hardware.

Concepts

Before you attempt to configure or operate the product, there are important concepts that you should understand. These concepts are addressed in this document as a basis for the other documents in the Technical Documentation set.

Contents

Process Types

The product ships with a set of predefined background processes that are grouped into the following process types:

- **Process What's Ready Processes** - These are the main set of processes that run the majority of the major processing jobs for the business.
- **Extract Processes** - These are background processes that extract information out of the product for various purposes such as interfaces or extracts for business processes such as bill printing etc.
- **To Do Processes** - These are a specialist set of regularly run processes to create, update or remove "To Do" entries from the product depending on outcomes of other processes in the system.
- **Object Validation Processes** - These are processes that are used to perform object validation during conversion and migration activities.
- **Conversion Processes** - These are processes that are used to perform conversion activities.
- **Purge Processes** – These processes purge inactive data from interface or staging objects.
- **Ad-hoc Processes** - These are processes that do not fit into any of the above categories and may be run at any time to process the data available or for special tasks.
- **Archive and Purge Processes** – These process archive and/or purge data from key entities.
- **Configuration Lab** – These processes migrate and/or synchronize data from environment to environment.

Contents

Process What's Ready Processes

Some background processes create and update records that are "ready for processing". The definition of "ready" differs for every process. For example:

- The bill cycle process produces bills for all accounts belonging to open bill cycles.
- The account debt monitor process analyzes the debt associated with all accounts whose review date is on or before the business date.

Processes of this type tend to use a business date in their determination of what's ready. For example, the bill cycle process creates bills for all bill cycles whose bill window is open (i.e., where the business date is between the bill cycle's start and end date). If the requester of the process does not supply a specific business date, the system assumes that the current system date should be used. If you need to use a date other than the current date, supply the desired date when you request the batch process.

Extract Processes

Some background processes extract a batch of information (to be interfaced OUT of the system). For example:

- Bill print extract.
- Letter print extract.

Processes of this type extract records marked with a specific batch number. If the requester of the process does not supply a specific batch number, the system assumes that the latest batch number should be extracted. If you need to re-extract an historical batch, you can supply the respective batch number when you request the batch process.

To rerun extracts it may be possible to simply rerun using a rerun number (if rerun number "re-runable") or by running the staging process that is associated with the extract then running the extract again. Refer to individual processes for more details.

Note: Default file formats for all supplied extracts are documented in the relevant business process documentation supplied with the product.

Note: The **FILE-PATH** and **FILE-DIR** additional parameters used in all extract processes are limited to two hundred and fifty-four (254) characters each fully expanded.

Ad-hoc Processes

There is a specific background process that doesn't fit into the any other categories. This process backs out bills that were created during the bill cycle process. You must supply specific parameters to this job in order to tell it which batch of bills to remove.

Conversion Processes

A number of processes are available when converting or migrating data from external applications into the product. These processes may or may not be used as part of an implementation depending on your conversion strategy.

Refer to the Conversion Toolkit Utilities documentation for further information about conversion.

Object Validation Processes

A number of processes are available to perform general validation for conversion or upgrade purposes. Each of the major objects in the database must be validated using the respective object validation program.

We strongly recommend validating each object in the following steps:

- Execute each object's validation program in random-sample mode to highlight pervasive errors. When you execute a validation in random-sample mode, you are actually telling it to validate every X records (where X is a parameter that you supply to the job).
- View errors highlighted by validation programs using the Conversion Error Summary transaction.
- Correct the errors using SQL. Note, you can use the base package's transactions (e.g., Person Maintenance, Premise Maintenance, etc.) to correct an error if the error isn't so egregious that it prevents the object from being displayed on the browser.
- After all pervasive errors have been corrected; re-execute each object's validation program in all-instances mode to highlight elusive, one-off errors.

In addition to validating your objects after conversion or an upgrade, the validation programs have another use. For example, you may want to experiment with changing the validation of a person and want to determine the impact of this new validation on your existing persons. You could change the validation and then run the person validation object - it will produce errors for each person that fails the new validation.

Refer to the Conversion Toolkit Utilities documentation for further information about conversion.

To Do Processes

To Do processes are processes that feed off all the other processes in the system and create, update or delete To Do as defined in the system tables for the product. The number of records created will depend on the values in the system tables and the number of records satisfying those criteria.

For example, after a **BILLING** run the **TD-BIERR** and **TD-BSERR** To Do process create To Do entries for all bills or segments, respectively, in error to groups defined in the system as handling those errors.

If the To Do functionality is not used at this site then the To Do jobs are not required to be run and should be removed from the schedules.

Refer to the "Defining General Options" and "To Do Business Process" documentation for further details.

Archive and Purge Processes

During the life of a product implementation at your site the data in the database will build up. Historical records will remain in the product until they are archived and/or purged. There are a set of background processes that execute the necessary components of the archiving engine to archive and/or purge data from an environment. They are usually scheduled in accordance with business requirements. Configuration of the archive engine must be performed before executing these processes.

Refer to the Archiving Engine Utilities documentation for further information.

Configuration Lab Processes

To migrate or synchronize data between environments a set of processes must be executed to initiate components of the Configuration Lab component of the product. These background processes are run only when synchronizing or comparing/apply changes between two environments.

Refer to the Configuration Lab Utilities documentation for further information.

Interface Processes

Some of the processes implemented by the product are in fact interfaces that may need to be updated during an implementation. Refer to the individual process register in the IT Supplemental Background Process Register for details of each process.

Batch Controls

In the product the concept of Batch controls are implemented to act as control points for a background process and have the following purposes:

- For those processes that extract information, the product batch control record defines the next batch number to be assigned to new records that are eligible for extraction. For example, the batch control record associated with the process that extracts bill print information defines the next batch number to be assigned to recently completed bill routings. When this bill print extract process next runs, it extracts all bill routings marked with the current batch number (and increments the next batch number).
- Each background process' batch control record organizes audit information about the historical execution of the background process. The system uses this information to control the restart of failed processes. You can use this information to view error messages associated with failed runs.
- Many processes have been designed to run in parallel in order to speed execution. For example, the Payment Process can be executed so that payments are processed in multiple "threads" (and multiple threads can execute at the same time). Batch control records associated with this type of process organize audit information about each thread in every execution. The system uses this information to control the restart of failed threads.

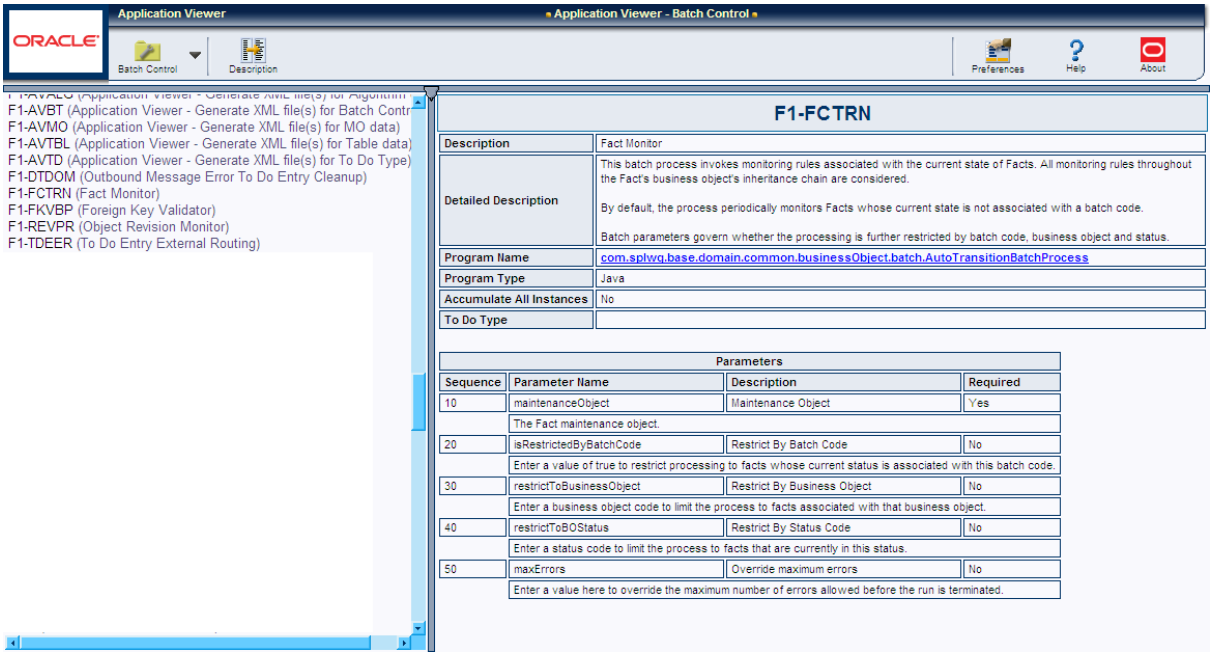
Refer to the online Administration Guide for more details of the screens to define Batch Control.

Note: The system is delivered with all necessary batch controls for the supplied base background processes.

Contents

Viewing Batch Controls Using the Application Viewer

While the Batch Controls can be viewed using the online system it is possible to view batch control information from the Application Viewer application supplied with your product. It can be accessed from the menu **Admin** → **A** → **Application Viewer** → **Batch Control**. A sample of the output that can be seen is shown on the following diagram:



This information is only available if the **F1-AVBT** background process has been executed or the **genappviewitems[.sh]** command is executed.

Adding your own batch controls

In any implementation Batch Controls may need to be added for new custom processes. This needs to be done in a manner so that they are consistent with the base product as well as be supported for upgrades. The following guidelines can assist in ensuring that Batch Controls are implemented correctly:

- Every custom process should have its own batch control. While it is possible to share batch controls, there may be concurrency and restart issues if the multiple processes are executed at the same time.
- Every instance of a particular process needs to have its own batch control. If you need to run an interface multiple times, once for each supplier for example, then a batch control records needs to be assigned to each instance so that they can be tracked and managed individually. This is also important because in an environment running multiple instances of a process, there is a far more likely chance the instances will be executing at the same time according to your schedule (see point above).
- All custom batch controls should be prefixed by **CM** to avoid conflicts with possible future processes introduced into the batch schedule. If this rule is not obeyed then there is a risk that when an upgrade is introduced it may cause concurrency and restart issues.
- Avoid using batch controls with any special characters (i.e. characters other than letters and numbers) as it may cause intermittent or operational errors. Avoid embedded blanks and characters such as !@#$%^|\?><,.~`"'{}[]&*()/:;.

Standard parameters

To standardize all the batch processes, the product uses a number of common standard parameters to uniformly provide functionality across all processes. The table below lists all the standard parameters:

Parameter	Usage
Batch code	Code is the unique identifier of the background process
Batch thread number	Thread number is only used for background processes that can be run in multiple parallel threads. It contains the relative thread number of the process. For example, if the billing process has been set up to run in 20 parallel threads, each of the 20 instances receives its relative thread number (1 through 20).
Batch thread count	Thread count is only used for background processes that can be run in multiple parallel threads. It contains the total number of parallel threads that have been scheduled. For example, if the billing process has been set up to run in 20 parallel threads, each of the 20 instances receives a thread count of 20.
Batch rerun number	Rerun number is only used for background processes that download information that belongs to given run number. It should only be supplied if you need to download an historical run (rather than the latest run).
Batch business date	Business date is only used for background processes that use the current date in their processing. For example, billing using the business date to determine which bill cycles should be downloaded. If this parameter is left blank, the system date is used.
Commit Interval	Override maximum records between commits. This parameter represents the number of transactions that are committed in each unit of work. This parameter is optional and overrides the background process's Standard Commit between records (each background process's Standard Commit between records is documented in the product documentation). You would reduce these values, for example, if you were submitting a job during the day and you wanted more frequent commits to release held resources. You might want to increase these values when a background process is executed at night (or weekends) and you have a lot of memory on your servers.
Timeout	Override maximum minutes between cursor re-initiation (also known as Cursor Reinitialization). This parameters are optional and override each background process's Standard Commit Records and Standard Cursor Re-Initiation Minutes (each background process's Standard Commit Records / Standard Cursor Re-Initiation Minutes is documented in individual process registers in the product documentation). You would reduce these values, for example, if you were submitting a job during the day and you wanted more frequent commits to release held resources (or more frequent cursor initiations). You might want to increase these values when a background process is executed at night (or weekends) and you have a lot of memory on your servers. *Note: The Maximum minutes between cursor re-initiation is for ORACLE implementations only.*
User ID	This is the userid that is used to access objects. It must be defined to the security component of the product.
Password	This parameter is not applicable (it is provided for backward compatibility).
Language	This is the language code used to retrieve messages and format output from background processes.

Parameter	Usage
Traces	Trace program at start (Y/N), trace program exit (Y/N), trace SQL (Y/N) and trace output. If trace program start is set to Y, a message is displayed whenever a program is started. If trace program at exist is set to Y, a message is displayed whenever a program is exited. If trace SQL is set to Y, a message is displayed whenever an SQL statement is executed. If trace out is set to Y, message are output from the program at execution points. This facility should only be used in QA and benchmarking.

Contents

Explanation of Timeout and Commit Interval

Note: Timeout only applies to COBOL based background processes.

The Timeout and Commit interval parameters are tunable parameters to affect the impact of the background processes on the other processes running and prevent internal database errors. In most cases using the defaults will satisfy your site requirements. It is also important to understand their impact to ascertain whether any change is required.

During processing of any background process a main "object" is used to drive the process. For example in Payment the main object is Payment Event. The Payment process loops through the payment event objects as it processes. For other processes it is other "objects" that are considered the main object. This main object type is used to determine when a transaction is complete.

For both Timeout and Commit interval this is important as:

- When a certain number of main objects have been processed then a database commit is issued to the database. This number is the Commit Interval. The larger the commit interval the larger the amount of work that the database has to keep track of between commit points.
- The Timeout parameter is used to minimize issues in ORACLE where the unit of work is so large it causes a "Snapshot too old". ORACLE stores undo information on the Rollback Segment and the read consistent information for the current open cursor is no longer available. This is primarily caused when ORACLE recycles the Rollback Segment storage regularly. The product is prevented by reinitializing the cursor on a regular basis to prevent an error. When this timeout, known as the Cursor Reinitialization, is exceeded then at the end of the current transaction a commit will be issued.
- At any time in a process a commit for objects processed may be caused by the reaching the Commit Interval or the time limit set on Timeout, whichever comes first.

Explanation of Thread Limit and Thread Number

One of the features of the Oracle Utilities Application Framework is the ability to run background processes using multiple threads.

The threading concept in the product is simple. Each thread takes a predetermined slice of the data to work on. The last thread checks if all other threads are finished and updates the status of the batch control records. For example, if you have 10 threads, then each thread takes 1/10th of the work. As each thread is executing it processes its workload and then completes, the last thread executing is responsible for updating the overall process status to indicate completion.

Implementing threading means you have to execute a number of jobs with an ascending thread number up to the thread limit. For example, if you have a job with 10 threads, you must run 10 jobs each with an unique thread number between 1 and 10 to complete the job. Threads can be located on the same machine or different machines. For example, you can run threads 1 to 5 on one machine and threads 6 – 10 on another.

Note: If there is limited "data skew" in the data then the threads should finish around the same time. If there is some data skew then some threads may finish later than others.

To implement multi-threading when you submit a process:

- Specify a thread limit greater than 1 as a parameter.
- Execute a process for every thread with a sequential thread number up to an including the thread limit. There are a couple of implementation guidelines with threading:
- Make sure the number of threads is not excessive. You do not want to flood the CPU's. A good rule of thumb is to have up to four (4) times the number of CPU's (this will vary).
- You must submit a process per thread. In some submission methods this is done automatically and in some it is done manually.
- Threading will increase throughout BUT it will cause higher than usual resource usage (CPU, Disk etc) as well as higher contention. Excessive threading can in fact cause performance degradation in online as well as background processing. Therefore the number of threads should not be excessive.

Almost all background processes within the product support multiple threads (the only processes typically single streamed are extracts and data loads as they involve sequential files).

Explanation of Restart and Rerun

The product allows all background processes to be restarted or rerun as required. During the execution of the background process, restart information per thread is stored within framework, like a "checkpoint". This checkpoint is performed at the last commit point as dictated by the Commit Interval and/or Timeout value (*Time out only applied to ORACLE implementations only*). When a "commit" is performed, the last commit point is recorded for the execution. If a thread of a background process fails, the database automatically rolls back to the last commit point. The thread can then be restarted from that point automatically or from the start of the data. To indicate the restart, the thread is executed with the same parameters as the original.

Additionally, processes are "re-run able". Re-run able means that a specific run number can be re-run as required or a process at a specific date. Using a rerun number or a previously used business date are all that is required to rerun a process.

Note: Not all background processes use Run number as a run indicator. Refer to the online documentation for which jobs are re-runable.

Monitoring Background Processes

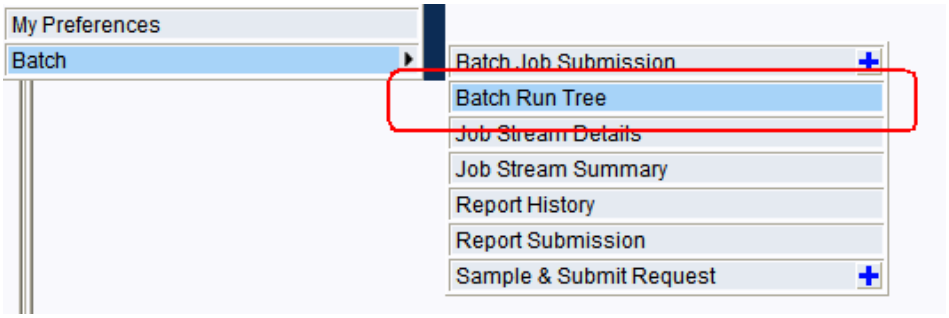
When a background process is initiated the product records information about the progress of the execution using a number of methods. These methods can be used to provide feedback to the operations personnel on the health and progress of individual processes.

Contents

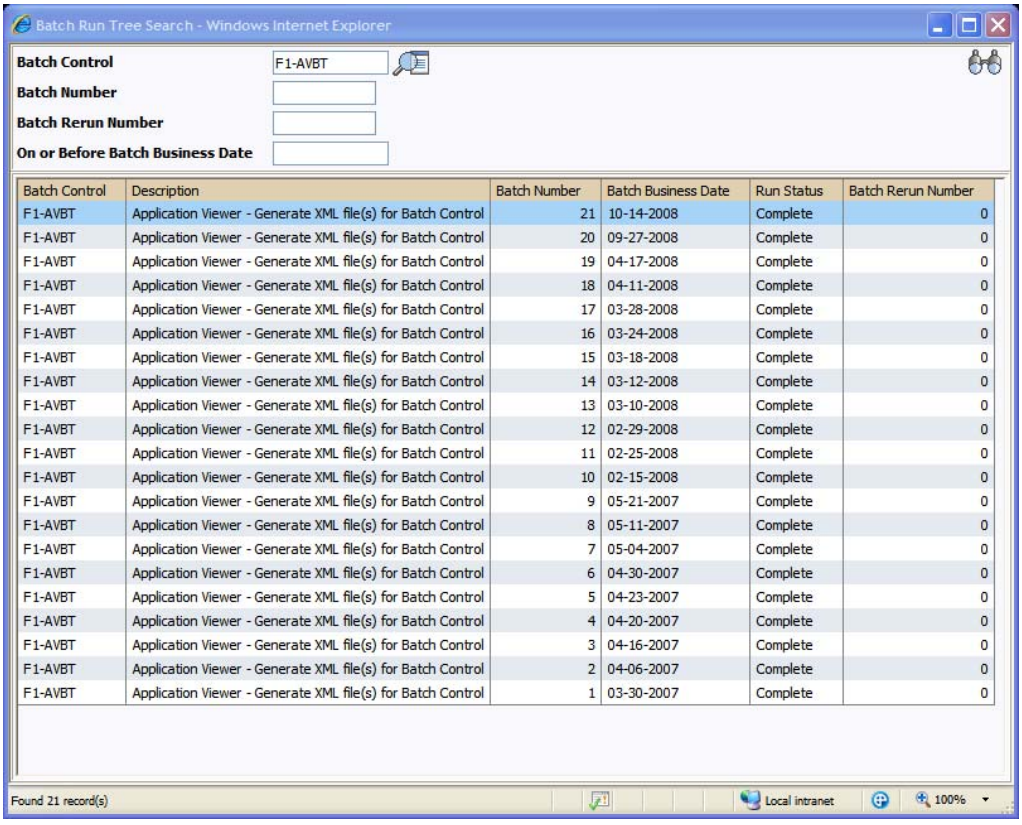
Batch Run Tree

Within the product browser interface there is an ability to monitor the status and outcomes of individual processes. This can be useful for finding out what actually occurred if an error condition occurred. To access the screen:

- Acquire a logon to the browser interface. It may be necessary to setup a special userid that operators can use to access the online.
- Select the **Batch** → **Batch Run Tree** option from the side menu. A sample is displayed below:

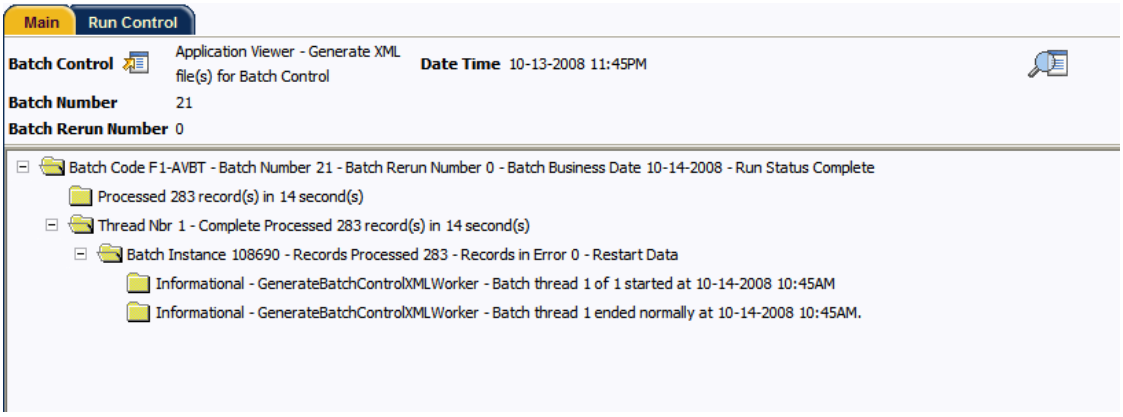


- A batch search window will appear to allow select of the individual execution of the process. It is possible to search on batch number, batch Control Id or rerun number. A sample is illustrated below.

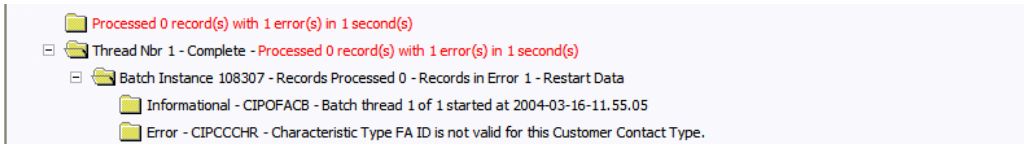


Batch Control	Description	Batch Number	Batch Business Date	Run Status	Batch Rerun Number
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	21	10-14-2008	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	20	09-27-2008	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	19	04-17-2008	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	18	04-11-2008	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	17	03-28-2008	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	16	03-24-2008	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	15	03-18-2008	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	14	03-12-2008	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	13	03-10-2008	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	12	02-29-2008	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	11	02-25-2008	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	10	02-15-2008	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	9	05-21-2007	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	8	05-11-2007	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	7	05-04-2007	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	6	04-30-2007	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	5	04-23-2007	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	4	04-20-2007	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	3	04-16-2007	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	2	04-06-2007	Complete	0
F1-AVBT	Application Viewer - Generate XML file(s) for Batch Control	1	03-30-2007	Complete	0

- Select the appropriate batch run to monitor. This will then open a portal with the appropriate run information:

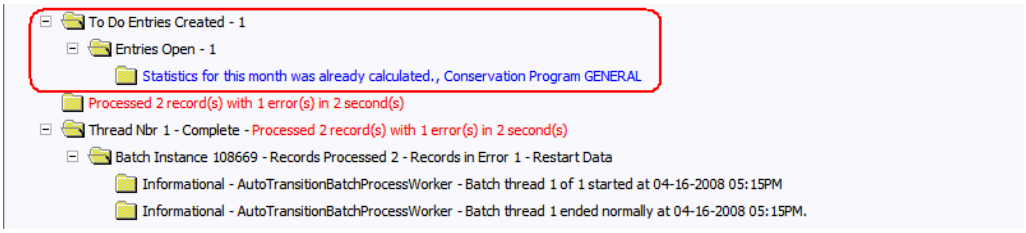


- In the case of the sample the process ended successfully. Additionally the following additional elements may be displayed:
  - If the processed ended unsuccessfully then the error message would be indicated.



Note: Technical Errors (e.g. SQL Errors) are indicated using this method.

- Business errors that are generated as To Do's are indicated separately.



- If the program was restarted, each restart would be displayed in the tree individually.

- To get more information about the error click on the error message on the tree.

The Batch Run tree is available to any valid user and is a method to communicate the execution information to the relevant business representatives.

Using SQL Queries to monitor background processes

The Batch Run Tree displays information within the database that is collected by the Oracle Utilities Application Framework for every background process execution, regardless of the method used to initiating the process.

While it is possible to use the Batch Run Tree as a "spot" check on particular processes, it is possible to create views on the underlying to extract the data for long term analysis of batch performance. These views can be then used to analyze or extract the data for further investigation.

The details of the views that can be created and types of analysis that can be performed are located in the "**Performance Troubleshooting Guide – Batch Troubleshooting**" whitepaper available on MetaLink (http://metalink.oracle.com).

Monitoring using JMX classes

The product supports management and monitoring using Java Management eXtensions (JMX) For example, a user may want to see exactly which processes are busy running in a worker JVM at any particular point, and may want to be able to cancel runaway tasks. Refer to the Java Management Extensions (JMX) Technology site for more information.

Java Management Extensions (JMX) is a technology that specifically addresses this requirement to introspect information within the Oracle Utilities Application Framework. By employing Management Beans (MBeans), the batch framework can implement management interfaces for the various monitoring and management instrumentation points. A remote client, such as Sun's JConsole or other JMX consoles/browsers, can then communicate with the active MBeans to query and modify the behavior of the batch node.

This section will outline the basic facilities available using JMX. Configuration of the JMX capability is discussed within each submission method outlined in Submission Methods.

Contents

Jconsole

Jconsole is a GUI application provided with the Java JDK installed. It can be invoked with the connection information configured with the product as a parameter, for example:

```
jconsole service:jmx:rmi:///jndi/rmi://server-name:9999/spl/fw/jmxConnector
```

The "server-name", port number (9999) and the "/spl/fw/jmxConnector" string correspond with the property values specified for the batch node. These values are specified in configuration files outlined in the relevant subsection of the "Submission Methods" section of this document.

Refer to http://java.sun.com/j2se/1.5.0/docs/guide/management/jconsole.html for more information on using **jconsole**.

Mbeans

The MBeans that expose the batch processes are divided into categories. The name of the MBean is constructed to indicate the type of batch process, the name of the batch process, and the thread number and count. For sake of uniqueness, the name also includes the Java thread number.

The name therefore is constructed as follows:

```
CCC_BBB_t_of_c.jjj
```

Where:

CCC	The type of MBean. This can be either "BatchJob" or "BatchThread".
BBB	The Batch code from the Batch Control.
t*_of_*c	The batch thread number and count. For "BatchJob" types, this will just be "0". For "BatchThread" types, the ***t*** is the thread number, and ***c*** the thread count.
jjj	the Java thread number, which will be unique within a batch node.

BatchJob Mbeans

Mbeans associated with "BatchJob" are created when any background process is initiated and are used to control any individual threads associated with the job. The Mbean exposes a number of attributes:

BatchNumber	The current batch number.
DateTimeStarted	The date and time the job was started.
DistThreadPool	The thread pool to which this job belongs.
ElapsedTime	How long the job has been running

ProgramName The program name

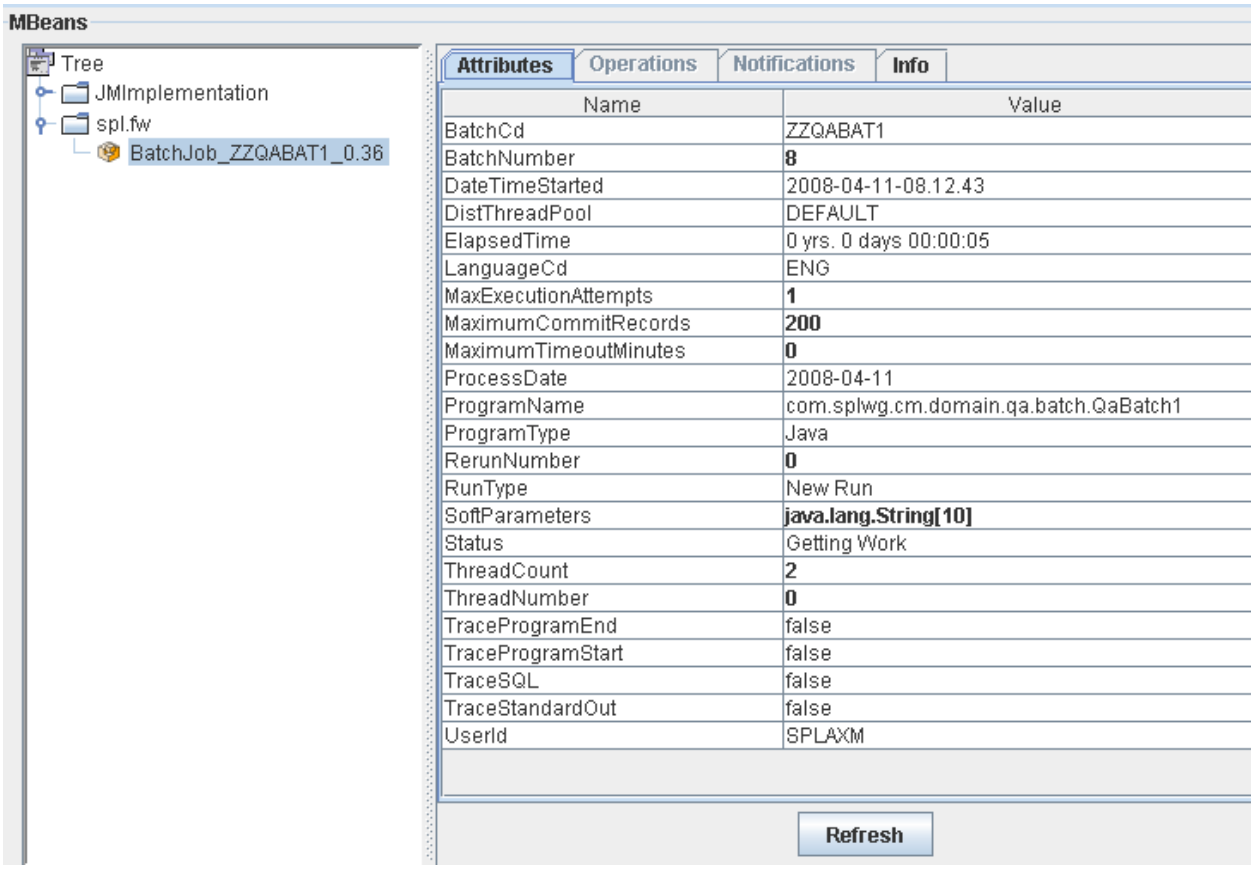
ProgramType The program type: "Java" or "COBOL".

RunType The type of run: "New Run", "Restart" or "Rerun".

Status Current status of the job. Valid values are "Initializing" (*very briefly in the beginning – prior to the call to getJobWork in the application class*); "Getting Work" means it is currently in the process of selecting the work units for the job; "Got Work" means it is has successfully selected the work and is in the process of initiating the threads.

Standard parameters The Job parameters are also displayed. Refer to Standard parameters for details of the valid values.

For example:



Note: Refreshing this information will dynamically update the values.

Note: JMX information is only displayed at active runtime and calls process can happen very fast – depending on the amount of data to be selected for the run – so the JMX console may not even detect this MBean.

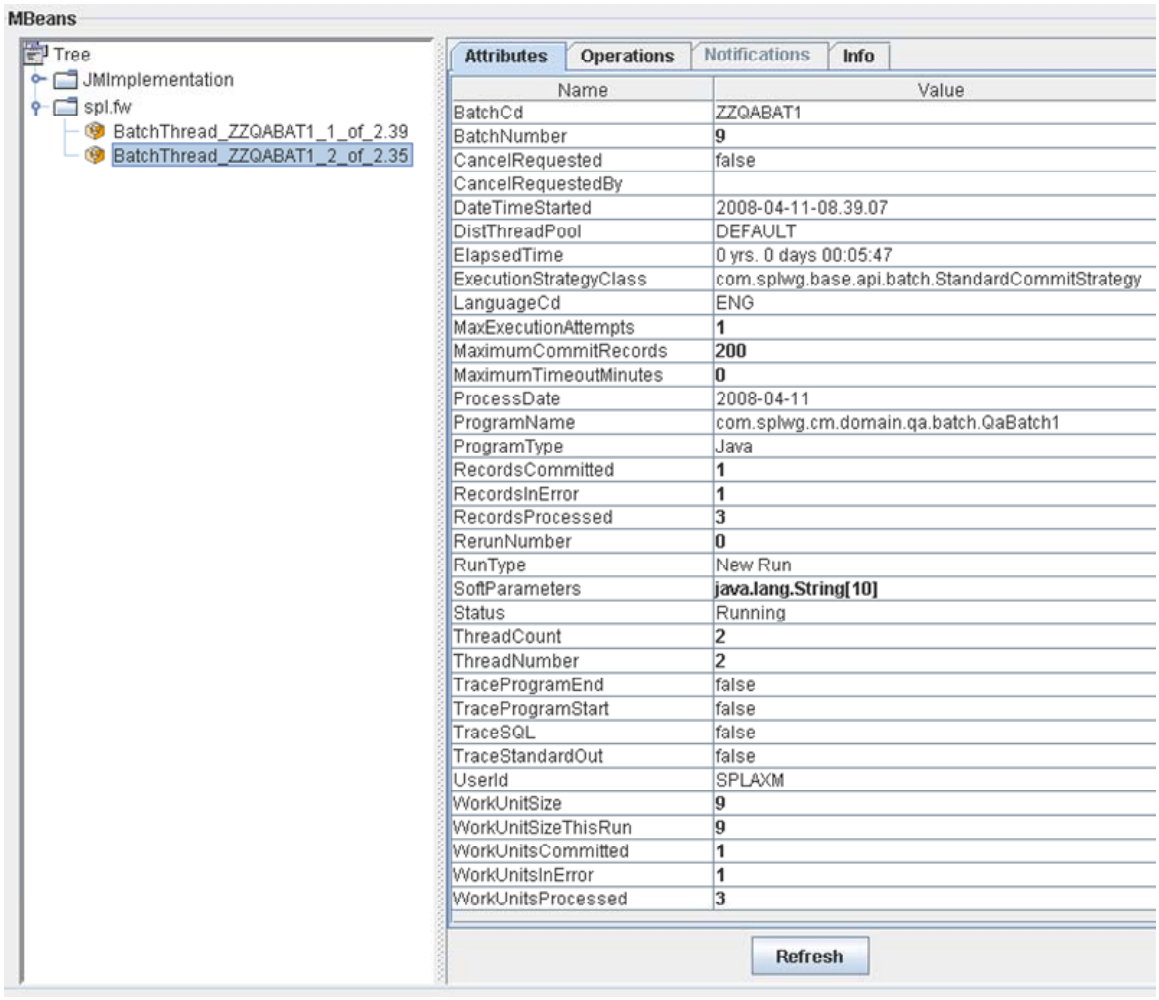
BatchThread Mbeans

Mbeans associated with "BatchThread" are created once the "getJobWork" method for a Java program has successfully completed. Each thread, as requested by the threadCount parameter for the job, will have its own MBean. A BatchThread MBean for a thread is alive for as long as it takes for the thread to complete, and automatically destroyed when the thread ends.

The batch thread number, as indicated in the MBean name, will be the current thread's thread number.

- In the case of Java, these MBeans expose the "running values" for a thread. The records/units processed, in-error and remaining, are provided as the thread runs and updates the MBean internally.
- For a COBOL thread, if COBOL used, the values are not as detailed, since COBOL does not work in terms of "work units", but some valuable information can still be obtained (e.g. elapsed time).

The Java BatchThread example below shows two threads running for job ZZQABAT1. The MBean name contains the thread number and count, and they show to be running in Java threads 39 and 35 respectively. The Java thread number is for uniqueness only.



Name	Value
BatchCd	ZZQABAT1
BatchNumber	9
CancelRequested	false
CancelRequestedBy	
DateTimeStarted	2008-04-11-08.39.07
DistThreadPool	DEFAULT
ElapsedTime	0 yrs. 0 days 00:05:47
ExecutionStrategyClass	com.splwg.base.api.batch.StandardCommitStrategy
LanguageCd	ENG
MaxExecutionAttempts	1
MaximumCommitRecords	200
MaximumTimeoutMinutes	0
ProcessDate	2008-04-11
ProgramName	com.splwg.cm.domain.qa.batch.QaBatch1
ProgramType	Java
RecordsCommitted	1
RecordsInError	1
RecordsProcessed	3
RerunNumber	0
RunType	New Run
SoftParameters	java.lang.String[10]
Status	Running
ThreadCount	2
ThreadNumber	2
TraceProgramEnd	false
TraceProgramStart	false
TraceSQL	false
TraceStandardOut	false
UserId	SPLAXM
WorkUnitSize	9
WorkUnitSizeThisRun	9
WorkUnitsCommitted	1
WorkUnitsInError	1
WorkUnitsProcessed	3

Note: Refreshing this information will dynamically update the values.

Note: JMX information is only displayed at active runtime and calls process can happen very fast – depending on the amount of data to be selected for the run – so the JMX console may not even detect this MBean.

The Mbean exposes a number of attributes:

BatchNumber	The current batch number.
CancelRequested	True if the thread has been asked to stop running. See CancelReqeustedBy.
CancelRequestedBy	If CancelRequested=true, this will be a string indicating the workstation from where the cancellation was requested. This value will also be logged to the Batch Run Tree.
DateTimeStarted	The date and time the job was started.
DistThreadPool	The thread pool to which this job belongs.
ElapsedTime	How long the job has been running.
ExecutionStrategyClass	This indicates the commit strategy followed by the program
ProgramName	The program name executed.
ProgramType	The program type: "Java" or "COBOL".
RecordsCommitted	The number of record updates that have been committed to the database. See note below.
RecordsInError	The number of records so far in error. This is what will be logged to the Batch Run Tree. See note below.
RecordsProcessed	The number of records processed so far. This is what will be logged to the Batch Run Tree. See note below.
RunType	The type of run: "New Run", "Restart" or "Rerun".
Status	Current status of the thread. Valid values are: "Initializing" (very briefly in the beginning – prior to the call to getJobWork in the application class); "Getting Work" means it is currently in the process of selecting the work units for the job; "Got Work" means it is has successfully selected the work and is in the process of initiating the threads.
WorkUnitSize	The total number of work units for this job. For new and restarted runs, this will always contain the total number of work units as selected in the getJobWork method when the job was originally started.
WorkUnitSizeThisRun	This is the number of work units for this particular run. For a restarted run, this value will typically be less than the above value; otherwise they will be the same.

WorkUnitsCommitted	The number of work units that have had their work committed.
WorkUnitsInError	The work units that have been found to be in error so far.
WorkUnitsProcessed	The work units that have been processed so far.

Note: The "Records…" numbers are what will be used to log to the Batch Run Tree, and they are usually in step with the "WorkUnits…" values. The reason they are shown separately is because some Java batch programs manually manipulate the record counts for the Batch Run Tree. The true progress status of a thread is reflected in the "WorkUnits…" counts.

Cancelling jobs using JMX

While JMX can be used to obtain monitoring information it is possible to cancel threads of jobs using the operations component of JMX. To cancel a thread the following process must be performed:

- Start the JMX console of your choice and connect to the relevant JMX port configured for the batch.
- Select the thread and job to be cancelled from the JMX console.
- Select the "Cancel" operation from the operations component of the console. The console may recognize the operations of the JMX classes and allow the actions to be processed. For example, `jconsole` will generate "cancelThread()" button. Issue the action.

Note: Depending on the JMX console used, a confirmation dialog may NOT be displayed and cannot be undone once issued. Ensure that the correct thread for the job is selected. To cancel a job, ALL threads must be cancelled.

- The job will be marked as cancelled and stopped. The IP address of the requestor is logged in the Batch Run Tree for auditing purposes.

Miscellaneous Operations

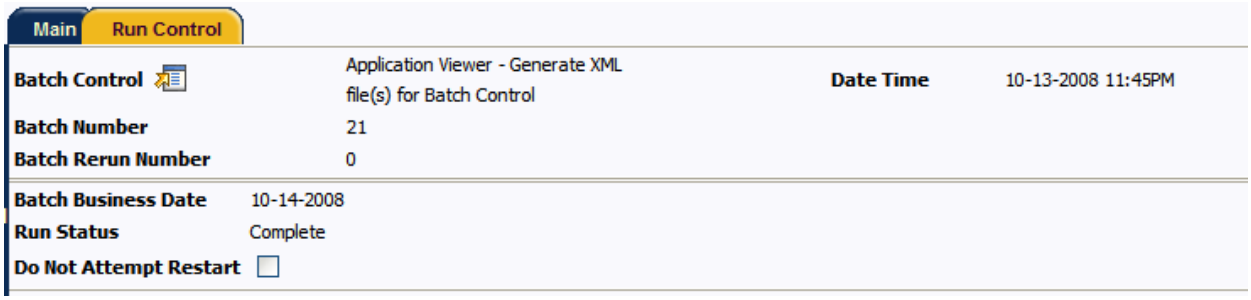
There are a number of common operations that are applicable to the background processing component of the product.

Contents

Forcing a process to not attempt restart

In some cases it is necessary to "force" a background process to be "complete" within the product. This tells the product not to attempt to restart the process but start "afresh". For example, a process may error and it may take a while to fix the error, instead of potentially holding up other processes you can tell the system to assume it has completed so that the next execution can start from the beginning and in fact reprocess the records.

To force the process to not to attempt a restart following the instructions to access the batch status information and select the "**Run Control**" tab and select the "**Do not Attempt Restart**" field. Remember to save the change using the **Save** button. A sample of this screen is illustrated below:



Error Processing

When a background process detects an error, the error may or may not be related to a specific object that is being processed. For example, if the program finds an error during batch parameter validation, this error is not object-specific. However, if the program finds an error while processing a specific bill, this error is object-specific. The system reports errors in one of the following ways:

- Errors that are not object-specific are written to the error message log in the Batch Run Tree.
- Some batch processes create entries in an "exception table" for certain object-specific errors. For example, an error detected in the creation of a bill may be written to the bill exception table. If an error is written to an exception table, it does not appear in the batch run tree. For each exception table, there is an associated ToDo Entry Process that creates a To Do Entry for each error to allow a user to correct the problem on-line.

- For some background processes, errors that do not result in the creation of an exception record may instead generate a To Do entry directly. For these processes, if you wish the system to directly create a To Do entry, you must configure the To Do type appropriately. Refer to To Do entry for object-specific errors for information about configuring the To Do type. If the background process detects an object specific error AND you have configured the system to create a To Do entry, the error is not written to the batch run tree. If you have configured your To Do type to not create To Do entries for certain errors, these errors are written to the batch run tree. Each process that may be configured in this way is indicated in the following sections in the Error May Generate To Do column. Note that not all tables below include this column. If the table does not include the column, then the creation of a ToDo for an object-specific error is not applicable for the types of processes documented in the table.

Some processes create exceptions and To Do entries. It is possible for a background process to create entries in an exception table AND create To Do entries directly, depending on the error. Consider batch billing; any conditions that cause a bill or bill segment to be created in error status result in a record added to the bill exception table or the bill segment exception table. However, any object-specific error that is not related to a specific bill or bill segment or any error that prevents a bill or bill segment from being created may result in a To Do entry for the object-specific error.

Marking a process complete from the command line

One of the situations that may occur in the product is that an executing process may prematurely stop before completion. This situation occurs if:

- The process was manually stopped using the UNIX/Windows kill command at the OS level. Operators may choose to kill a process if it appears to be having a detrimental effect on the system.
- The application server that is running the process has a hardware fault that causes the process to stop prematurely.
- The database server that is running has a software or hardware fault that severs the connection to the database prematurely.

In all the above situations the status within the product does not reflect the current status of the process as the background process was prevented from updating its batch control records in time.

In most cases a simple rerun of the process with the same parameters may be performed, after the situation that caused the fault has been remedied, to start the process from its last consistency point. If there is a desire to ensure that the batch control information reflects the status after a failure then the **UPDERR** process should be executed prior to any restart.

Submission Methods

The product objects that can be run in the background are all submitted through the operating system command shell in one format or another.

This section outlines the technical aspects of submitting batch processes using the following methods:

- **Interactive Submission** - Interactive mode submission.
- **Online Submission** - Submission of jobs online and using the inbuilt job scheduler (where available).
- **External Scheduler Submission** - How to use a third party scheduler (or command line) to run background processes.

Contents

Interactive submission (SPLBATCH)

One of the first methods supported for developers is the "Interactive" method of submission. This method takes its name from the level of inactivity required during the initiation of the background process.

The idea with this method is that the background process driver program is invoked directly (via a supplied utility) and the parameters required for the job will be requested for input interactively. At the end of the interactivity the background process is executed and control is returned when the process is completed. Effectively you input the parameters interactively and run the background process in the foreground.

This submission method is only suggested to be used for development testing only for the following reasons:

- The process is actually run the foreground so interaction during execution is limited.
- To execute the background process a single threaded JVM is executed with the full context. This means the whole framework is loaded into memory before the actual execution is performed. This is not efficient for non-development use.
- The interactivity will not allow re-specification of incorrect values for parameters. While some validation is performed during input of parameters, full validation is performed during execution of the actual background process.

To use this method of submission the following process needs to be performed:

- Logon to the host machine using an appropriate authorized account (for UNIX/Linux it must be a member of the group used for the product. Ensure that
- The environment must be attached to using the **`splenviron[.sh]`** utility. For example:

  ```
  splenviron.sh –e DEV
  ```

- Execute the **`SPLBATCH[.sh]`** utility from the command line to initiate the interactive submission of the background process.
- When prompted, answer the parameter prompts using the following guidelines.

- Output is displayed to screen and batch run tree (see **Contents**
- **Batch Run** Tree

Using SQL Queries to monitor background processes
Monitoring using JMX classes
- Batch Run Tree for further information).

Contents

Anatomy of an interactive submission

During the interactive submission process the following occurs:

- A Java Virtual Machine (JVM) is initiated according to the precepts in the **SPLBATCH[.sh]** utility and the configuration settings specified in the **$SPLEBASE/splapp/standalone/config** (or **%SPLEBASE%\splapp\standalone\config** on Windows) directory. Please refer to the Operations and Configuration Guide for an explanation of the configuration settings.
- A full version of the product (including online classes) is loaded into the JVM.
- Database connections are established to the underlying database according to the **$SPLEBASE/splapp/standalone/config/hibernate.properties** (or **%SPLEBASE%\splapp\standalone\config\hibernate.properties** on Windows) configuration file. Please refer to the Operations and Configuration Guide for an explanation of the configuration settings.
- The parameters for the background process are prompted. For example:

Parameter Prompt	Usage
Batch Code	Batch control for job. This is mandatory as it is used to determine the program to execute.
Batch Thread Number	Number of Thread to execute. This is mandatory and must be less or equal to thread limit. Usually specify **1**.
Batch Thread Count	Thread limit. Usually specify **1**.
Batch Rerun Number	The batch run number to rerun (Background process must support rerun numbers for this to be used). Specify **0** to ignore.
Batch Business Date. If not accepted, will use the system date	The business date in ISO format (i.e. YYYY-MM-DD). Use blank entry to use system date.
Maximum Number of records to commit:	Commit interval. Use blank entry to use program default.

Parameter Prompt	Usage
If not accepted, the program default will be used	
Maximum Time-out minutes: **If not accepted, the program default will be used**	Timeout for ORACLE. Use blank entry to use program default.
User ID	Specify a valid user for security reasons.
User Password	Not applicable. Use blank entry to ignore.
Language Code	Language code used for error messages. This is mandatory. Specify **ENG** for English. The valid language pack must be installed to use this.
Trace Program Start (Y/N)	Enable tracing of program starts. Specify **Y** for Yes or **N** for No. This is mandatory. Usually specify **N**.
Trace Program Exit (Y/N)	Enable tracing of program exits. Specify **Y** for Yes or **N** for No. This is mandatory. Usually specify **N**.
Trace SQL (Y/N)	Enable tracing of SQL Statements. Specify **Y** for Yes or **N** for No. This is mandatory. Usually specify **N**.
Trace Standard Output (Y/N)	Enable tracing of debug output from programs. Specify **Y** for Yes or **N** for No. This is mandatory. Usually specify **N**.
Additional Run Parameters (Blank line to end) : **parmname=parmvalue**	Specify additional parameters as specified on Batch Control. Use a blank line to indicate last parameter.

- The program indicated on the batch control of the Batch Code specified is executed. For each execution the following information is output:

SCHEDULER ID	Internal number allocated to instance of process execution. All messages are associated with this Scheduler Id.
BATCH CD	Batch Code submitted.
BATCH THREAD NBR	Batch Thread number submitted
BATCH THREAD CNT	Thread Limit used for submission
BATCH NBR	Batch Number allocated to this execution. Value of **0** indicates that the program does not support batch

	numbers or current batch number is used.
BATCH BUSINESS DT	Batch Business date used for execution in YYYY-MM-DD format.

- The execution of the batch submission is also written to **$SPLOUTOUT** (or **%SPLOUTPUT%** on Windows) in a log file named <*batch_cd*>.<*datetime*>.THRD<*threadnumber*>.stdout and <*batch_cd*>.<*datetime*>.THRD<*threadnumber*>.stderr where <*batch_cd*> is the batch code submitted, <*datetime*> is the date and time of the execution (in format YYYYMMDDHHMMSS.S format) and <*threadnumber*> is the thread number submitted.

Return Codes

The following return codes apply to the processing using this method:

Return Code	Usage
0 (zero)	Successful
Non-zero	Unsuccessful. See log files for more information.

Limitations of the interactive submission method

The interactive submission method is recommended for use with development only for the following reasons:

- Each thread runs within its own JVM. This is not efficient for multiple simultaneous jobs or multiple threads.
- The prompting is interactive and designed for developers only. While it is possible to pass a parameter file containing the values for each of the prompts into the **SPLBATCH[.sh]** utility it is not recommended.
- Incorrectly specified values for prompts cannot be corrected. You must wait for the job failure to start again.
- There is no monitoring method or cancelling from the foreground execution (apart from killing the terminal session).
- JMX monitoring is not possible with this method.

In summary, this method is designed for development usage only.

Online Submission

One of the most important useful testing/demonstration facilities of the product is the ability to submit batch processes from the online component of the product. An authorized user can submit any batch process using an online batch submission page.

The on-line batch submission page enables you to request a specific background process to be run. When submitting a background process on-line, you may override standard system parameters and you may be required to supply additional parameters for your specific background process. After submitting your background process, you may use this page to review the status of the submission.

Basically the following process is used to submit background processes using the online submission method:

- The process to be executed is registered online as to be submitted (or queued). This marks the process execution as "Pending". When you request a batch job to be submitted from on-line, the execution of the desired background process will result in the creation of a batch run. Just as with background processes executed through your scheduler, you may use the Batch Run Tree page to view the status of the run, the status of each thread, the run-instances of each thread, and any messages that might have occurred during the run.

  Note: Your online submission record is assigned a status value so that you may know whether your job has been submitted and whether or not it has ended, however, it will not contain any information about the results of the background process itself. You must navigate to the Batch Run Tree page to view this detail.

- A background process is scheduled (using cron or using the submission daemon) that will pickup any "Pending" background process executions and execute them. When you save a record on the batch job submission page, the batch job does not get submitted automatically. Rather, it saves a record in the batch job table. A special background process will periodically check this table for pending records and will execute the batch job. This background process will update the status of the batch job submission record so that a user can determine when their job is complete.

Note: At installation time, your system administrator will set up this special background process to periodically check for pending records in the batch job submission table. Your administrator will define how often the system will look for pending records in this table.

It should be noted that this special background process only submits one pending batch job submission record at a time. It submits a job and waits for it to end before submitting the next pending job.

Note: If you request a batch job to be run multi-threaded, the special background process will submit the job as requested. It will wait for all threads to complete before marking the batch job submission record as ended.

During execution the status of the execution in the batch run tree is updated as well as the original submission screen. If you wish the system to inform you when the background process completes, you may supply your email address. The email you receive will contain details related to the batch job's output; similar to the job results you would see from the batch run tree.

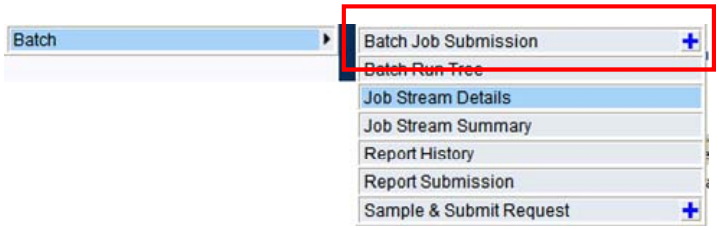
Note: This assumes that during the installation process, your system administrator configured the system to enable email notification. Your administrator may also override the amount of detail included in the email notification.

Contents

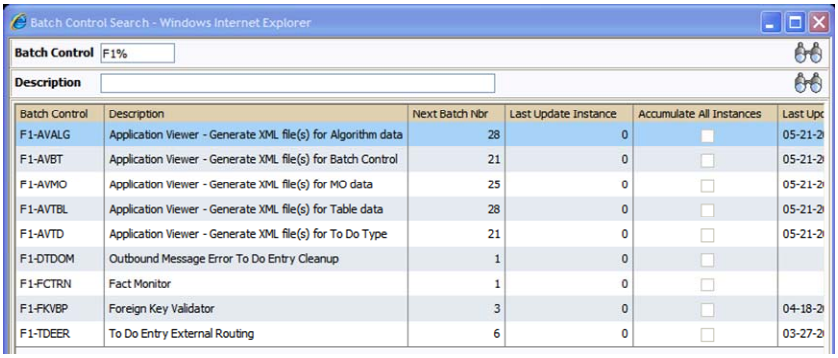
Using Online Submission

The process of submitting using the online method is as follows:

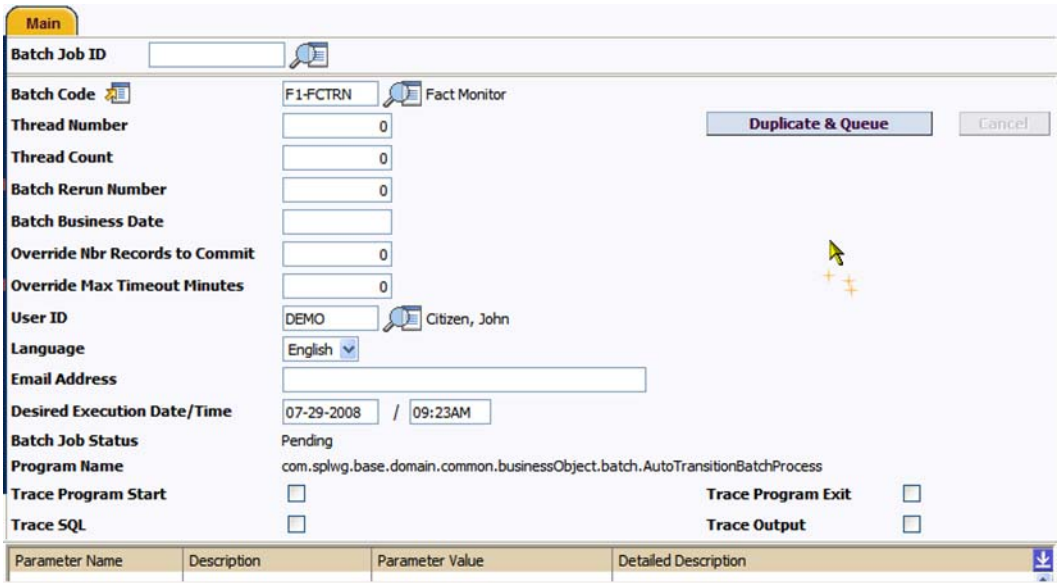
- Logon to the product environment using your browser. Use the appropriate URL.
- Navigate to **Main → Batch → Batch Submission**



- Find the batch control you wish to submit. You can use the Batch Code or the Description of the job to find it. It is possible to submit any valid job in the list.



- Fill in the prompts on the screen with the appropriate values.



Prompt	Comments

Prompt	Comments
Batch Job Id	The Batch Job ID is a system generated random number that identifies a particular submission.
Batch Code	To submit a batch job, choose the Batch Code for the process you wish to submit.
Batch Thread Number	Thread number is used to control whether a background processes is run single threaded or in multiple parallel threads. It contains the relative thread number of the process. For example, if the process X has been set up to run in 20 parallel threads, each of the 20 instances receives its relative thread number (1 through 20). Note: Not all processes may be run multi-threaded. Many of the system background processes may be run multi-threaded. When submitting a background process on-line, you may also run a multi-threaded process or run a single thread of a multi-threaded process. The fields Thread Count and Thread Number on the batch submission page control the multi-threaded process requests: To run a multi-threaded process, indicate the number of threads in Thread Count and enter 0 in the Thread Number. For example, to run the job XXX with 10 threads, enter Thread Count = 10 and Thread Number = 0. This will execute all 10 threads of job XXX. To run a single thread in a multi-threaded process, indicate the number of threads in Thread Count and indicate the Thread Number you would like to run. For example, to run only thread 1 out of 10 threads for job XXX, enter Thread Count = 10 and Thread Number = 1. This will execute thread 1 out of 10 for XXX. To run a process as a single thread, enter Thread Count = 1 and Thread Number = 1. This will execute the background process single-threaded. *Note:* When running a multi-threaded process, the completion of the last of the threads will "mark" the batch job submission record as ended.
Batch Thread Count	Thread count is used to control whether a background processes is run single threaded or in multiple parallel threads. It contains the total number of threads that have been scheduled. For example, if the billing process has been set up to run in 20 parallel threads, each of the 20 instances receives a thread count of 20.
Batch Rerun Number	Rerun number is only used for background processes that download information that belongs to given run number. It should only be supplied if you need to download an historical run (rather than the latest run).
Batch Business Date	Business date is only used for background processes that use a date in their processing. For example, billing using the business date to determine which bill cycles should be downloaded. If this parameter is left blank, the system date is used at the time the background process is executed.

Prompt	Comments
Override Nbr Records To Commit and Override Max Timeout Minutes	These parameters are optional and override each background process's Standard Commit Records and Standard Timeout Minutes (each background process's Standard Commit Records / Standard Timeout Minutes is documented in the list of system background processes).
User ID	Enter the user ID for the background process. This field defaults to the id of the current user.
Language Code	Language code is used to access language-specific control table values. For example, error messages are presented in this language code.
Email	If you wish the system to notify you when the batch job is complete, enter your Email ID. This field defaults to the email address for the current user, if populated on the user record. *Note:* SMTP support must be configured to operate.
Desired Execution Date/Time	The Desired Execution Date/Time defaults to the current date and time. Override this information if you wish the background process to be executed at some future date and time. If you wish to request a batch job to be submitted in the future, you may do so when creating your batch job submission record by entering a future submission date. The special background process, which looks for pending records in the batch job submission table, will only submit batch jobs that do not have a future submission date.
Batch Job Status	This indicates the current status of the batch job.
Program Name	The Program Name associated with the batch control code is displayed. This is used for tracking purposes
Trace Program Start	Toggle this switch on if you wish a message to be written whenever a program is started.
Trace Program Exit	Toggle this switch on if you wish a message to be written whenever a program is exited.
Trace SQL	Turn on this switch if you wish a message to be written whenever an SQL statement is executed.
Trace Output	Turn on this switch if you wish a message to be displayed for special information logged by the background process.

Note: The trace parameters are typically only used during QA and benchmarking.

Note: The information displayed when the trace output switch is turned on depends on each background process. It is possible that a background process displays no special information for this switch.

Note: The location of the output of this trace information is defined by your system administrator at installation time.

- If additional parameters have been defined for this background process on the Batch Control page, the Parameter Name, Description and an indicator of whether or not the parameter is Required are displayed. Enter the desired Parameter Value for each parameter.

  Each of the batch processes has, as part of its run parameters, a preset constant that determines how many errors that batch process may encounter before it is required to abort the run. You can override this constant with an optional additional parameter (**MAX-ERRORS**). The input value must be an integer that is greater than or equal to zero. The maximum valid value for this parameter is 999,999,999,999,999.

- Press the "Save" key. Once you have entered all the desired values, Save the record in order to include it in the queue for background processes.

- If you wish to duplicate an existing batch job submission record, including all its parameter settings, display the submission record you wish to duplicate and use the Duplicate and Queue button. This will create a new Batch Job Submission entry in pending status. The new submission entry will be displayed.

- If you wish to cancel a Pending batch job submission record use the Cancel button. The button is disabled for all other status values.

Note: Saving a record on this page does not submit the batch job immediately. A special background process will run periodically to find pending records and submit them. Depending on how often the special process checks for pending records and depending on how many other pending records are in the 'queue', there may be a slight lag in submission time. If the desired execution date/time is close to midnight, it is possible that your batch job will run on the day after you submit it. If you have left the business date blank in this case, keep in mind that your business date would be set to the day after you submit the job.

After saving the process in the batch submission screen the following process is performed:

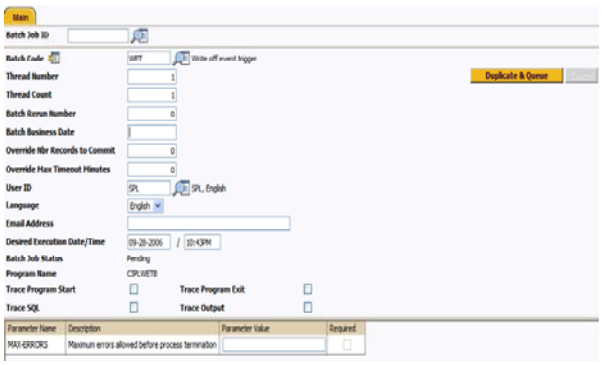
- The execution of the process is registered within a batch run table in "Pending" status. Prior to execution the user may cancel the job by pressing the cancel button. This updates the process status to "Canceled".

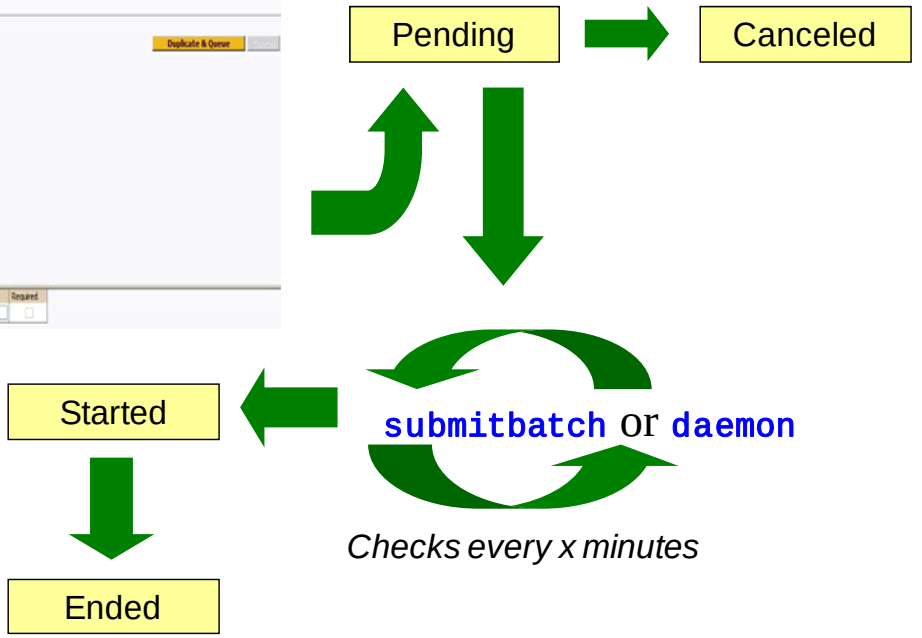
- At installation time, the product administrator sets up an additional process, the online daemon (or **submitbatch[.sh]** cron utility see submitbatch – Command based daemon for more details) which polls the batch run table every x minutes (where x is the parameter used on the command line).

- It processes each "Pending" process in sequence, using FIFO and at process start updates the batch run table with a status of "Started". This indicates the process is executing. The user cannot cancel the process after it has been "Started". At this time the batch run tree is populated with the run information as it is executing, including restart information and threading.

- If the process is successful or errored, the batch run information with an "Ended" status. You must check the Batch Run tree to see if has been successful.

This diagram illustrates the process:

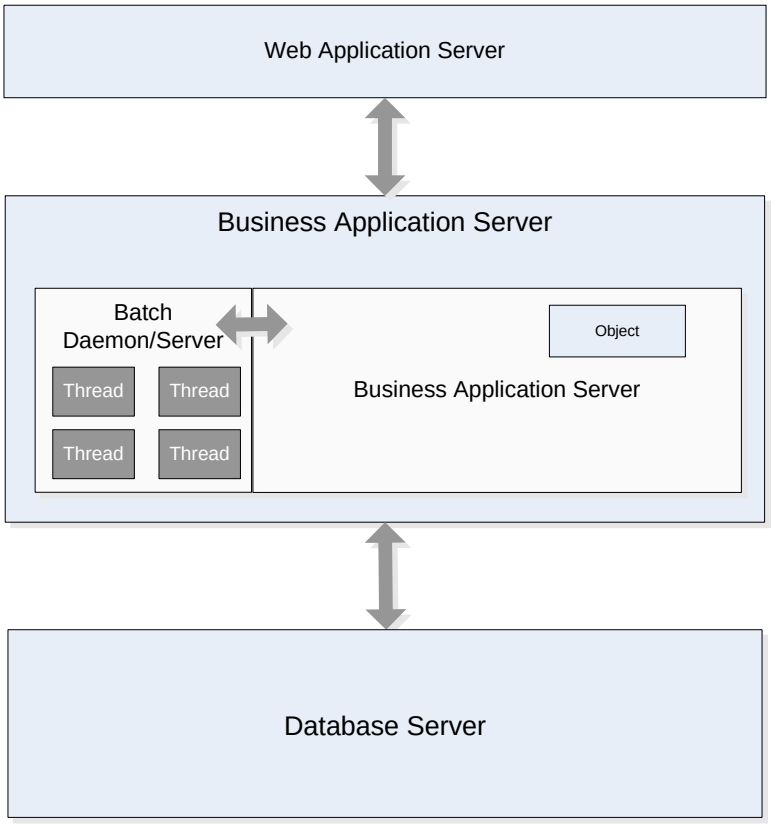




Online Batch Daemon

During the installation of the Business Application Server component of the product, it is possible to configure part of the Business Application Server runtime to become a batch daemon. This means that part of the JVM used by the Business Application Server can be used as a daemon (or "listener") for processes submitted online.

If configured, the Business Application executes an internal process to poll for "Pending" processes registered using the online submission screen. This batch daemon then executes the batch process within the Business Application Server JVM. The daemon can be configured to limit the impact on the online system by limiting the number of concurrent threads that can be executed. The following diagram illustrates this process:



At installation time the installer asks additional questions to disable/enable the batch daemon:

```
5. Environment Configuration
…
      Batch Server Enabled:                                    true
      Batch Threads Number:                                    5
      Batch Scheduler Daemon:                                  true
…
```

The three settings used can be configured using the following guidelines:

- **Batch Server Enabled** – Enable batch to be run within the JVM.
- **Batch Threads Number** – Maximum number of threads to limit background processes to within the JVM if "Batch Server Enabled" is set to Yes. This number of threads represents the number of threads surrendered from the main JVM thread pool and allocated to running batch exclusively. The default is 5.
- **Batch Scheduler Daemon** – Enable the daemon to check for pending jobs in the Batch Submission or inbuilt job scheduler[1]. If a pending job is found it is passed to the "Batch Server" for execution.

Note: After application of Patch 8219387 the default settings for the online daemon and batch server are "false" for backward compatibility NEW.

The valid combinations of these settings are as follows:

[1] The inbuilt job scheduler is NOT covered in this guide. Refer to the online documentation provided with your product for details of this facility (if provided with your product).

Batch Server Enabled	Batch Scheduler Daemon	Comments
Yes	Yes	Batch Daemon runs on this Business Application Server and any batch jobs found by the daemon are executed on this Server.
Yes	No	Batch Daemon does not run on this Business Application Server but this Business Application Server can execute batch programs. It is assumed that another business application server has been allocated as a batch scheduler daemon.
No	Yes	Batch Daemon does run on this Business Application Server but Batch submission does not run on this Business Application Server. It is assumed that another business application server has been allocated as a batch server.
No	No	Batch does not execute on this Business Application Server and no Batch Daemon has been allocated to this

Contents

Guidelines for using the Batch Server/Batch Scheduler Daemon

This facility is not applicable to all environments and all situations at a site, the following guidelines will assist in the appropriate use of the facility:

- If the environment is going to use the online submission (or inbuilt scheduler) then the Batch Server and Batch Scheduler Daemon should be enabled for Business Application Server allocated to the environment. If multiple Business Application Servers are allocated to the same environment, then there should only be one server with the "Batch Server enabled" set to "Yes" and only one server with the "Batch Scheduler Daemon" set to "Yes" (they can be the same server or different servers). This setting is common for non-production environments.
- If the environment is not going to use the online submission, then both "Batch Server Enabled" and "Batch Scheduler Daemon" should set to "No". This is a common setting for Production as online submission is usually disabled in production.

Logging using the Batch Server/Scheduler Daemon

The execution of any batch submission is also written to **$SPLOUTOUT** (or **%SPLOUTPUT%** on Windows) in a log file named ***<batch_cd>.<datetime>*.THRD*<threadnumber>*.stdout** and ***<batch_cd>.<datetime>*.THRD*<threadnumber>*.stderr** where <*batch_cd*> is the batch code submitted, <*datetime*> is the date and time of the execution (in format YYYYMMDDHHMMSS.S format) and <*threadnumber*> is the thread number submitted

Configuring JMX with the Batch Server/Scheduler Daemon

The executing threads Batch Server can be monitored using the JMX adapter by adding the following lines to the **$SPLEBASE/etc/conf/root/WEB-INF/classes/spl.properties** (or **%SPLEBASE%\etc\conf\root\WEB-INF\classes\spl.properties** on Windows) file:

```
spl.runtime.management.rmi.port=<port>
spl.runtime.management.connector.url.default=service:jmx:rmi:///jndi/rmi://<server>:<port>/spl/fw/jmxConnector
java.rmi.server.hostname=<server>
```

Where:

<server> Name of the host (or IP address) where the Business Application Server is located.

<port> A unique port allocated for the JMX agent to broadcast on. This port must be unique to the host it is located upon.

To implement the change the **initialSetup[.sh]** or **genupdatewar[.sh]** commands must be executed. Additionally on platforms when a WAR/EAR file is used, the WAR/EAR file must be redeployed. Refer to the Operations and Configuration Guide for details.

submitbatch – Command based daemon

Note: This facility is documented for completeness only, it is recommended that the online submission daemon be used in preference to this facility.

For backward compatibility purposes, there is a facility that can invoked on the command line (or in cron, or similar, facility) to act as an alternative to the online scheduler daemon. This facility will run a polling script that will detect a pending process and invoke the interactive method (in background) to execute the process.

This can be configured using the following command line

submitbatch[.sh] [-e *<env>*] [-s *<seconds>*] [-h] [-k]

where:

-s *<seconds>* Run as deamon and pause ***<seconds>*** seconds between loops of checking whether there is more work to do. Without the –s it runs one job and stops (recommended if used with cron).

-k Stop the background batch processor

-v Print out verbose messages

-e ***<env>*** Dummy parameter that is only used to make the process more identifiable so 'ps -edf' can be used to determine which **submitbatch** script belongs to which environment (***<env>***).

-h Print command line help.

External Scheduler Submission

In V2.2 of the Oracle Utilities Application Framework, an externally callable interface and batch framework was introduced. A number of utilities **threadpoolworker** and **submitjob** have been include in the product to allow external schedulers (or a command line) to establish a JVM to run background processes and then submit background process to that JVM.

Note: The words "node" and "JVM" are interchangeable in this section.

Contents

threadpoolworker[.sh] Utility

Note: **CLUSTERED** mode outlined in the documentation below is only available as a post SP6 as Fix number 9166248 NEW.

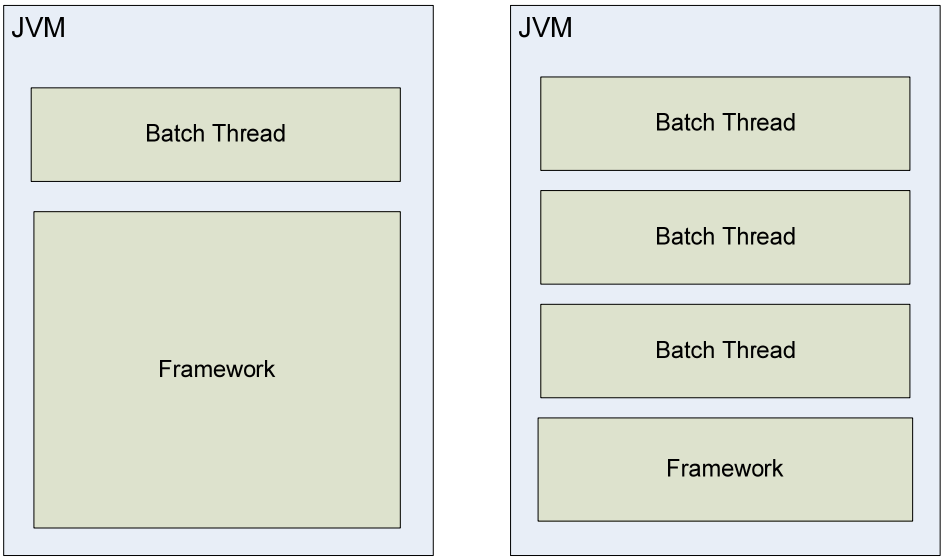
This script starts a "long-lived worker" node (JVM) in a distributed batch grid environment. Once successfully started, this process will accept submissions from lightweight submitter nodes and execute the jobs as requested by the submitters. A worker JVM may also host a scheduler daemon, which, if activated inside a worker, will poll for job submission requests from the web application that were done via the Batch Job Submission transaction.

There are three modes to execute background processes using this facility **CLUSTERED**, **DISTRIBUTED** and **THIN**.

- In **THIN** mode, the application program (Java or COBOL, if used) is executed in a "stand-alone" JVM. This means that a full application context is established before the application program is invoked, and destroyed when the execution ends. If a job is submitted in multiple threads, each thread requires its own context, in its own separate JVM. **THIN** mode is typically used by developers to isolate their tests from other developers and is not recommended for production use.

- In **DISTRIBUTED** mode, at least one "worker" JVM must be started and left running to poll for work requests from "submitter" JVMs. Worker JVMs are also known as "grid nodes", because multiple workers can be started to load-balance batch. **DISTRIBUTED** mode is the existing, classic way that customers have been running their batch processes in past releases of the Oracle Utilities Application Framework. In this mode, one or more batch workers are started and left to run as long-running, background tasks. Each worker can be individually configured to process n number of threads concurrently, and this can further be grouped into “thread pools”. The workers also have the option to host a job scheduler daemon, whose role is to listen for and execute online job submissions (via the Batch Job Submission page for example).
- In **CLUSTERED** mode, as with **DISTRIBUTED** at least one "worker" JVM must be started. These "worker" JVM's may be standalone or clustered with appropriate batch. The difference between **CLUSTERED** and **DISTRIBUTED** is that in a **CLUSTERED** setup, worker and submitter JVMs (members) are more tightly joined in a Coherence based cluster, resulting in better management of various events, such as workers abruptly stopping (because of Java/COBOL crashes for example), jobs getting cancelled, etc. As long as at least one member is active in a cluster, jobs can be appropriately handled in the case of unexpected interruptions.

It is highly recommended that customers use the **CLUSTERED** mode for production.



THIN DISTRIBUTED/CLUSTERED

If only THIN submissions are ever used, such as is currently done through **SPLBATCH[.sh]**, script **threadpoolworker[.sh]** does not have to be executed. If this is the case, jobs can be submitted in THIN mode from the command-line using script **submitjob[.sh]** – i.e. script SPLBATCH is not required for THIN submissions.

This script may be executed more than once if multiple workers are required. This may be for performance or load-balancing purposes, or to simply provide separate thread pool configurations in a distributed grid. Worker JVMs in a grid can be started on different machines (even if the platforms differ), provided they all access the same database and contain the COBOL runtime appropriate for the architecture. If the web application is also batch-enabled, the worker hosted by the web application then becomes one node among the nodes that were started as described above.

If multiple worker nodes, including the web application, are configured to host a scheduler daemon, only one of those will be the active daemon. The others are dormant until the active one becomes unavailable for some reason – e.g. if the JVM is killed – in which case one of the dormant ones will automatically become active.

Note: A single product environment can be either **CLUSTERED** or **DISTRIBUTED**. Mixing JVMs that start up in **CLUSTERED** and **DISTRIBUTED** mode will have unpredictable results. NEW

threadpoolworker and F1_TSPACE_ENTRY

The **DISTRIBUTED** and **CLUSTERED** approaches use a database tuple space table **F1_TSPACE_ENTRY** for operations and management. The role of this table varies differently depending on the execution mode of the worker or submitter.

In **DISTRIBUTED** mode, whenever a **threadpoolworker** or **submitjob** starts an entry is created in the **F1_SPACE_ENTRY** table. This is used by the **threadpoolworker** to advertise that it is ready to accept work (known as a **THREAD_OFFER**) using a lease (to indicate when it is to check back with **F1_TSPACE_ENTRY**). If the **threadpoolworker** finds work in the **F1_TSPACE_ENTRY** it issues a **GRID_WORK** record grabbing the submitters work and executes the indicated job. After the job has ended it issues a **WORK_ENDED** against the **submitter** record in the **F1_TSPACE_ENTRY** table and updates the **threadpoolworker** entry to **THREAD_OFFER** again to accept more work. The submitter process creates an **F1_TSPACE_ENTRY** table of **WORK_OWNER** to indicate it waiting to be executed and waits. When the **threadpoolworker** accepts the submitters work it has marked the submitters records as **GRID_WORK** indicating it is processing the task. At this time, the submitted polls regulars waiting for the **WORK_ENDED** message to indicate the work has been completed.

The issues with this type of processing is when there are issues with the **threadpoolworker** or **submitter**. If these processes fail, then the **F1_TSPACE_ENTRY** does not adequately reflect the state of the processes. This may cause internal synchronization issues in some cases. A common technique used by sites when this happens is to clear the offending **F1_TSPACE_ENTRY** entries manually or truncating the table altogether and reissue the work. The latter is dangerous if there is work still running in the product.

The **CLUSTERED** mode was created to address this issue. It uses **F1_TSPACE_ENTRY** for some persistence but each **threadpoolworker** in the cluster is aware of the other nodes and the work that is allocated to it. Any node can be used in execution of processes and in the case of submitter failure the node will communicate to the appropriate process to keep the relevant parties informed. This also occurs when a **threadpoolworker** failure where the other nodes inform the relevant parties involved of the failure.

threadpoolworker.properties configuration file

To use the **threadpoolworker** utility a configuration file must be created to specify the attributes of the JVM. The **threadpoolworker.properties** file should be placed in the **$SPLEBASE/etc** directory (or **%SPLEBASE%\etc** directory on Windows).

The properties file contains the default properties for **threadpoolworker**. The following sample illustrates the values:

```
com.splwg.grid.distThreadPool.threads.DEFAULT=5
com.splwg.grid.distThreadPool.threads.LOCAL=0
```

```
com.splwg.batch.scheduler.daemon=true
spl.runtime.management.rmi.port={port}
spl.runtime.management.connector.url.default=service:jmx:rmi:///jndi/rmi://{host}:{port}/spl/fw/jmxConnector
```

The following table describes the parameters:

`com.splwg.grid.distThreadPool.threads.<poolname>`	Number of Threads for pool <*poolname*>.
`com.splwg.batch.scheduler.daemon`	Whether the node will act as a scheduler daemon.
`com.splwg.grid.executionMode` NEW	Mode of the threadpoolworker. Valid values are: **`THIN`**, **`DISTRIBUTED`** or **`CLUSTERED`**
`spl.runtime.management.rmi.port`	JMX RMI Port to use. If omitted, JMX is disabled.
`spl.runtime.management.connector.url.default`	Default JMX service. Required if rmi.port is specified. In the example, above the URL format is shown. Substitute {host} for hostname of machine and {port} for unique RMI port.
`com.splwg.batch.submitter.maxExecutionAttempts`	This specifies how many times the worker(s) in the grid should attempt execution of the work submitted by this submitter. If the application program crashes and brings down the worker JVM with it, this parameter is designed to prevent any other worker nodes in the grid from picking up this same bad work request and thereby spreading the "poison work" around the grid, crashing JVMs along the way and ultimately bringing the batch grid down completely. The default is set to 1 and should be left like that unless there is a good reason to change it
`tangosol.coherence.log` NEW	Specifies destinations of log entries. This is set to **`log4j`**. This setting should not be changed. Only used for **`executionMode CLUSTERED`**
`tangosol.coherence.log.level` NEW	Level of logging: 0 - only output without a logging severity level specified will be logged

	1 - all the above plus errors 2 - all the above plus warnings 3 - all the above plus informational messages. Default is **3**. Only used for **executionMode CLUSTERED**
`tangosol.coherence.guard.timeout` NEW	The timeout value used to guard against deadlocked or unresponsive services. It is recommended that service-guardian/timeout-milliseconds be set equal to or greater than the packet-delivery/timeout-milliseconds value. A timeout of zero will disable service guardians. A timeout of zero will disable service guardians. The default value is **0**. Only used for **executionMode CLUSTERED**
`tangosol.coherence.cluster` NEW	The cluster-name element contains the name of the cluster. In order to join the cluster all members must specify the same cluster name. The name an be up to 32 characters to define the name of the cluster. This is required and must be unique for each environment. With classic **DISTRIBUTED** mode, the batch JVMs for an environment are naturally grouped because they register themselves through database table **F1_TSPACE_ENTRY**, but in **CLUSTERED** mode the JVMs are joined through a Coherence cache". The cache may be across all environments, so a unique cluster name, along with address and port (see below), is required to ensure that they are appropriately grouped per environment. Environments are typically separated by database and/or database user, so a possible convention may be to use a combination of database name and owner Id as the cluster name, for example "CCBDEMO.CISADM".

	Only used for **executionMode CLUSTERED**
`tangosol.coherence.clusteraddress` NEW	Specifies the multicast IP address that a Socket will listen or publish on. Valid values are from 224.0.0.0 to 239.255.255.255. For non-multicast implementations use the Well Known Addresses (WKA) functionality.
`tangosol.coherence.clusterport` NEW	Specifies the multicast port that the Socket will listen or publish on. Valid values are from 1 to 65535. For non-multicast implementations use the Well Known Addresses (WKA) functionality.
`tangosol.coherence.localhost` NEW	Specifies the IP address that a Socket will listen or publish on for WKA functionality. *Note:* The **localhost** setting may not work on systems that define **localhost** as the loopback address; in that case, specify the machine name or the specific IP address. Default is **localhost**. Only used for **executionMode CLUSTERED**
`tangosol.coherence.localport` NEW	Specifies the port that the Socket will listen or publish on for WKA functionality. Legal values are from 1 to 65535. Default value is **8088**. Only used for **executionMode CLUSTERED**. *Note:* By default, this port will automatically increment if the specified port cannot be bound to because it is already in use.[2]
`tangosol.coherence.wka` NEW	Specifies a list of "well known" addresses (WKA) that are used by the cluster discovery protocol in place of multicast broadcast. If one or more WKA is specified, for a member to join the cluster

[2] This behavior can be overridden by specifying the property **tangosol.coherence.localport.adjust=false**

	it will either have to be a WKA or there will have to be at least one WKA member running. Additionally, all cluster communication will be performed using unicast. If empty or unspecified multicast communications will be used. Only used for **executionMode CLUSTERED**
`tangosol.coherence.wka.port` NEW	Port numbers associated with WKA addresses in `tangosol.coherence.wka`. Only used for **executionMode CLUSTERED**

This file should be modified for site-specific values. For example, the scheduler daemon may not be required to be activated by default, in which case property **com.splwg.batch.scheduler.daemon** should be changed to false (or removed entirely to use the system default).

Note: If using Coherence Cluster Address and Coherence Cluster port then they form a multicast address unique to the environment/cluster. All worker and submitter JVMs that want to join this cluster must have the same cluster name, cluster address and cluster port.

The first worker JVM that starts for a particular combination of cluster/address/port establishes that cluster. Other JVMs with this same combination will then join this cluster.

The framework guards against the submission of jobs to the wrong cluster in two ways. Firstly, if the address/port matches an existing cluster's address/port, but the cluster name is different, the JVM will exit with this error message:

This member could not join the cluster because of a configuration mismatch between this member and the configuration being used by the rest of the cluster.

Secondly, if a JVM's cluster name references an existing cluster, but the database to which the existing cluster is connected is not the same as the joining JVM's, it will exit with this message:

Error validating cluster membership. Terminating...

In either case it is a configuration issue that needs to be corrected

Contents

Multi-cast or Uni-cast NEW

The **CLUSTERED** mode can use multi-cast or uni-cast to communicate across the **threadpoolworker** nodes in a cluster. By default **CLUSTERED** mode uses a multicast protocol to discover other nodes when forming a cluster. For information about multi-cast and uni-cast see the following sites:

- Discussion of protocols - http://wiki.tangosol.com/display/COH35UG/Network+Protocols
- Advanced Configuration of the multi-cast listener - http://wiki.tangosol.com/display/COH35UG/multicast-listener
- Advanced Configuration of the uni-cast listener - http://wiki.tangosol.com/display/COH32UG/unicast-listener

Well Known Addresses NEW

The default option at installation is use of multicast, if multicast is not an option, the "well-known-addresses" feature may be used. It requires manual edits of the **threadpoolworker.properties** and **submitbatch.properties** properties files.

The WKA properties specify one or more "well-known" nodes (JVMs) that are used to start a cluster and are likely to be available for other nodes to join. These well-known nodes are used by the other nodes to find their way into the cluster without the use of multicast. Note that only one of these nodes is required to be up; they don't all have to be up at the same time.

The following example shows a WKA configuration.

threadpoolworker.properties on server test1

```
tangosol.coherence.log=log4j
tangosol.coherence.log.level=3
tangosol.coherence.guard.timeout=0
tangosol.coherence.cluster=tugbudemo.cisadm
tangosol.coherence.localhost=test1
tangosol.coherence.localport=19000
tangosol.coherence.wka=test1,test2
tangosol.coherence.wka.port=19000,38000
```

threadpoolworker.properties on server test2

```
tangosol.coherence.log=log4j
tangosol.coherence.log.level=3
tangosol.coherence.guard.timeout=0
tangosol.coherence.cluster=tugbudemo.cisadm
tangosol.coherence.localhost=test2
tangosol.coherence.localport=38000
tangosol.coherence.wka=test1,test2
tangosol.coherence.wka.port=19000,38000
```

submitbatch.properties

```
tangosol.coherence.log=log4j
tangosol.coherence.log.level=3
tangosol.coherence.guard.timeout=0
```

```
tangosol.coherence.cluster=tugbudemo.cisadm
tangosol.coherence.wka=test1,test2
tangosol.coherence.wka.port=19000,38000
```

This example defines two threadpoolworker JVMs as WKA nodes:

- test1, port 19000
- test2, port 38000

With at least one of these two threadpoolworkers available, any other node that wants to join the cluster will be able to, provided that node's configuration specifies the same list of WKAs.

This is illustrated with the **submitbatch.properties** example, which is what all submitters will use. The wka properties reference the two WKA worker nodes, which allow it to join the cluster.

For further details, refer to http://wiki.tangosol.com/display/COH35UG/unicast-listener.

Note: WKA and multicast may not be mixed within the same cluster.

When using WKA, these properties in **threadpoolworker.properties** and **submitbatch.properties** must be removed:

```
tangosol.coherence.clusteraddress
tangosol.coherence.clusterport
tangosol.coherence.clusterport
```

tangosol-coherence-override.xml NEW

The **tangosol.coherence.*** properties as discussed above are "property overrides" for the Coherence configuration elements. The **threadpoolworker** and **submitbatch** properties are configured like this for conformity and so that the Oracle Utilities Application Framework configuration utility can be used. The overide feature is documented at http://wiki.tangosol.com/display/COH35UG/Command+Line+Setting+Override+Feature .

For manual configuration the standard Coherence **tangosol-coherence-override.xml** file may be used and placed in the **$SPLEBASE/splapp/standalone/config** (**%SPLEBASE%\splapp\standalone\config** in Windows) directory. In that case, the corresponding Coherence overrides in the **threadpoolworker** and **submitbatch** properties need to be removed. The following example illustrates a well-known-addresses configuration with logging level set to debug (5):

```
<coherence>
    <cluster-config>
        <unicast-listener>
            <well-known-addresses>
                <socket-address id="1">
                    <address system-property="tangosol.coherence.wka1"> sf-ugbu-21</address>
                    <port system-property="tangosol.coherence.wka1.port">19000</port>
                </socket-address>
                <socket-address id="2">
                    <address system-property="tangosol.coherence.wka2"> sf-ugbu-18</address>
```

```
                    <port system-property="tangosol.coherence.wka2.port">38000</port>
                </socket-address>
            </well-known-addresses>
        </unicast-listener>
    </cluster-config>
    <logging-config>
        <destination>log4j</destination>
        <severity-level>5</severity-level>
    </logging-config>
</coherence>
```

threadpoolworker[.sh] command line options

Note: that the appropriate environment has to be attached to before this script can be executed (i.e. **splenviron[.sh] -e <environment>** has to be run), unless the script is directly invoked from the Windows explorer by double-clicking on it. In that case it will automatically attempt to attach to the environment that owns the bin directory in which it is located and then prompt for options.

The following options can be specified when executing script **threadpoolworker**.

threadpoolworker[.sh] [-d] [-e][-h][-i][-J][-p][-Q][-R][-s]

where command line options are:

-d <Y|N>	Whether the node is acting as a scheduler daemon. Specify **N** for No and **Y** for Yes. If you are already using a scheduler daemon in the online system or are not using online submission then set this to **N**. Default is **N**.
-e <DISTRIBUTED|CLUSTERED> NEW	Execution mode for this threadpool. If **CLUSTERED** is the threadpool will join the cluster specified in the **threadpoolworker.properties** file.
-h	Show command line help. List the available options and their descriptions. It is formatted for a 121-column width display. The information is not logged.
-i *<RMI Port>*	Override port number for JMX. If specified with **-R**, this number will be used only to substitute applicable URL **{port}** references. This option will not add any new RMI/JMX properties - it can only be used to override existing ones. This option specifies the port number to: • Use when the framework starts an RMI Registry and • Substitute in all JMX Connector URL

	{port} references.
-J	Do not start JMX monitoring. For each property prefixed by **spl.runtime.management.connector.url** that is defined with the default set of properties (e.g. in the **threadpoolworker.properties** file), the framework will start a JMX Connector for the specified URL. This activates JMX monitoring inside the worker node so that a client JMX console can be used to monitor and manage active threads. If this option is specified, the framework will not start any JMX connectors.
-p *<name=value,name=value,…>*	Thread pool(s) offered by this worker node. Consists of one or more name=value pairs, where "name" is the name of the pool and "value" the number of threads offered in the pool. For example, DEFAULT=5,ONLINE=3
-Q	Preview the properties that would be active for this run. Used for testing. Preview the properties that would be in use for the run without actually running the application. Specify other options along with this option to show how they would merge with, override or substitute the default properties. The information is not logged.
-R	Do not start a local RMI registry. If property **spl.runtime.management.rmi.port** is defined as a default property (e.g. in the **threadpoolworker.properties** file), the batch framework will attempt to start an RMI registry on the given port number. This option can be used to suppress the automatic RMI registry startup. It may be required if an externally started RMI registry is already running. *Note*: If this option is used, the RMI port number supplied through the **-i** option is only used for substitution in the JMX Connector URLs.
-s *<space name>*	Space name for "hard partition" of workers. Default is **MAIN**. Reserved for internal use only.

When **threadpoolworker** is invoked, the command-line options will alter its default configuration. The default configuration options come from either internal system defaults or the **threadpoolworker.properties** file described above.

The properties are overridden in the following order:

1. The **threadpoolworker.properties** supersedes the internal system defaults.
2. The command-line options supersede the defaults in **threadpoolworker.properties** and the internal system defaults.

Example 1

Assuming we have the above set of properties in **threadpoolworker.properties** and script **threadpoolworker** is invoked as follows:

```
threadpoolworker[.sh] –d Y
```

This will replace the default "daemon" property to "N" (i.e. false) so that the properties now look as follows:

```
com.splwg.grid.distThreadPool.threads.DEFAULT=5
com.splwg.grid.distThreadPool.threads.LOCAL=0
com.splwg.batch.scheduler.daemon=false
spl.runtime.management.rmi.port=9999
spl.runtime.management.connector.url.default=service:jmx:rmi:///jndi/rmi://{host}:{port}/spl/fw/jmxConnector
```

workersubmitterlog4j.properties

The threadpoolworker logs information to **$SPLSYSTEMLOGS/threadpoolworker.*<datetime>*.log** (or **%SPLSYSTEMLOGS%\threadpoolworker.*<datetime>*.log**) where <datetime> is the date and time in YYYYMMDDHHMMSS format of the start of the node. This configuration file is provided in the **$SPLEBASE/etc** directory (or **%SPLEBASE%\etc** directory on Windows).

This is a standard log4j properties file, but declares two appenders specifically for **threadpoolworker**:

1. the console and
2. a file in the product's **$SPLSYSTEMLOGS** (or **%SPLSYSTEMLOGS%** on Windows) directory.

Note: If Patch 7642944 is installed then the **threadpoolworker.log** file is written to **$SPLOUTOUT** (or **%SPLOUPUT%** on Windows) NEW

This log4j configuration allows the worker's output to be logged to the command prompt window as well as a log file. This file should not be altered unless desired.

Contents

Automatic Log Rotation NEW

By default the **threadpoolworker.log** file is appended to while the **threadpoolworker** is active. If the **threadpoolworker** is long running and the log needs to be automatically rotated on a daily basis the following changes should be applied to the **workersubmitterlog4j.properties** file:

Replace:

```
...
### F1 is set to be a FileAppender.
log4j.appender.F1=org.apache.log4j.FileAppender
…
```

with

```
### F1 is set to be a RollingFileAppender
log4j.appender.F1=org.apache.log4j.DailyRollingFileAppender
log4j.appender.F1.DatePattern='.'yyyy-MM-dd
```

Refer to http://logging.apache.org/log4j/1.2/index.html for additional options.

threadpoolworker.log Log format NEW

Note: For this facility, Patch 7642944 must be installed NEW.

By default all system parameters are output to the log for the **threadpoolworker** for support purposes. All parameters are output for the internals and JVM in alphabetical order.

For example:

```
(api.batch.ThreadPoolWorker) Properties for this run:
(api.batch.ThreadPoolWorker)   awt.toolkit=sun.awt.windows.WToolkit
(api.batch.ThreadPoolWorker)   com.splwg.batch.scheduler.daemon=false
(api.batch.ThreadPoolWorker)   com.splwg.batch.submitter=false
(api.batch.ThreadPoolWorker)
com.splwg.grid.distThreadPool.threads.DEFAULT=3
(api.batch.ThreadPoolWorker)   com.splwg.grid.spaceName=XXXX
(api.batch.ThreadPoolWorker)
com.splwg.threadPoolWorker.defaultPropertiesFile=…
(api.batch.ThreadPoolWorker)
com.splwg.threadPoolWorker.showPropertiesForRun=true
(api.batch.ThreadPoolWorker)   file.encoding.pkg=sun.io
(api.batch.ThreadPoolWorker)   file.encoding=ISO8859_1
(api.batch.ThreadPoolWorker)   file.separator=\
(api.batch.ThreadPoolWorker)   hibernate.c3p0.max_size=6
(api.batch.ThreadPoolWorker)
hibernate.connection.url=jdbc:oracle:thin:@xxxxx:1531
(api.batch.ThreadPoolWorker)
java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment
(api.batch.ThreadPoolWorker)
java.awt.printerjob=sun.awt.windows.WPrinterJob
```

```
(api.batch.ThreadPoolWorker)   java.class.path=…
(api.batch.ThreadPoolWorker)   java.class.version=49.0
(api.batch.ThreadPoolWorker)   java.endorsed.dirs=…
(api.batch.ThreadPoolWorker)   java.ext.dirs=…
(api.batch.ThreadPoolWorker)   java.home=…
(api.batch.ThreadPoolWorker)   java.io.tmpdir=…
(api.batch.ThreadPoolWorker)   java.library.path=…
…
```

submitjob[.sh]

The **submitjob[.sh]** utility provides a means for the scheduler to submit a job. It can be invoked from a command prompt, Windows explorer (on Windows platforms) or a 3rd party scheduler. This script can be used to submit the job to an active **threadpoolworke**r process, or to run it in a full-context standalone JVM.

submitbatch.properties configuration file

To use the **submitjob[.sh]** utility a gconfiguration file must be created to specify the global attributes of all the jobs. The **submitbatch.properties** file should be placed in the **$SPLEBASE/etc** directory (or **%SPLEBASE%\etc** directory on Windows).

The properties file contains the default properties for **threadpoolworker**. The following sample illustrates the values:

```
com.splwg.grid.executionMode=DISTRIBUTED
com.splwg.batch.submitter.distThreadPool=DEFAULT
com.splwg.batch.submitter.promptForValues=false
com.splwg.batch.submitter.rerunNumber=0
com.splwg.batch.submitter.threadNumber=0
com.splwg.batch.submitter.threadCount=1
com.splwg.batch.submitter.maximumCommitRecords=200
com.splwg.batch.submitter.userId=AUSER
com.splwg.batch.submitter.languageCd=ENG
```

The following table describes the parameters:

com.splwg.grid.executionMode	Mode of execution. Valid values: **THIN**, **DISTRIBUTED** or **CLUSTERED**.
com.splwg.batch.submitter.distThreadPool	Name of pool to be used for job. If **threadpoolworker** not used then LOCAL must be specified.

com.splwg.batch.submitter.promptForValues	Whether interactive mode is to be used. Specify **true** for Yes (development use only) and **false** for No
com.splwg.batch.submitter.rerunNumber	Default run number. Default 0
com.splwg.batch.submitter.threadNumber	Default thread number. Default: 0
com.splwg.batch.submitter.threadCount	Default thread limit. Default: 1
com.splwg.batch.submitter.maximumCommitRecords	Default commit interval.
com.splwg.batch.submitter.userId	Default userid used for all jobs
com.splwg.batch.submitter.languageCd	Default Language code used for messages. Relevant language pack must be installed.
tangosol.coherence.log NEW	Specifies destinations of log entries. This is set to **log4j**. This setting should not be changed. Only used for **executionMode CLUSTERED**
tangosol.coherence.log.level NEW	Level of logging: 0 - only output without a logging severity level specified will be logged 1 - all the above plus errors 2 - all the above plus warnings 3 - all the above plus informational messages. Default is **3**. Only used for **executionMode CLUSTERED**
tangosol.coherence.guard.timeout NEW	The timeout value used to guard against deadlocked or unresponsive services. It is recommended that service-guardian/timeout-milliseconds be set equal to or greater than the packet-delivery/timeout-milliseconds value. A timeout of zero will disable service guardians. A timeout of zero will disable service guardians. The default value is

	0. Only used for **executionMode CLUSTERED**
tangosol.coherence.cluster NEW	The cluster-name element contains the name of the cluster. In order to join the cluster all members must specify the same cluster name. The name an be up to 32 characters to define the name of the cluster. This is required and must be unique for each environment. With classic **DISTRIBUTED** mode, the batch JVMs for an environment are naturally grouped because they register themselves through database table **F1_TSPACE_ENTRY**, but in **CLUSTERED** mode the JVMs are joined through a Coherence cache". The cache may be across all environments, so a unique cluster name, along with address and port (see below), is required to ensure that they are appropriately grouped per environment. Environments are typically separated by database and/or database user, so a possible convention may be to use a combination of database name and owner Id as the cluster name, for example "CCBDEMO.CISADM". Only used for **executionMode CLUSTERED**
tangosol.coherence.clusteraddress NEW	Specifies the multicast IP address that a Socket will listen or publish on. Valid values are from 224.0.0.0 to 239.255.255.255. For non-multicast implementations use the Well Known Addresses (WKA) functionality.
tangosol.coherence.clusterport NEW	Specifies the multicast port that the Socket will listen or publish on. Valid values are from 1 to 65535. For non-multicast implementations use the Well Known Addresses (WKA) functionality.
tangosol.coherence.localhost NEW	Specifies the IP address that a Socket will listen or publish on for WKA functionality.

	Note: The **localhost** setting may not work on systems that define **localhost** as the loopback address; in that case, specify the machine name or the specific IP address. Default is **localhost**. Only used for **executionMode CLUSTERED**
tangosol.coherence.localport NEW	Specifies the port that the Socket will listen or publish on for WKA functionality. Legal values are from 1 to 65535. Default value is **8088**. Only used for **executionMode CLUSTERED** *Note:* By default, this port will automatically increment if the specified port cannot be bound to because it is already in use.[3]
tangosol.coherence.wka NEW	Specifies a list of "well known" addresses (WKA) that are used by the cluster discovery protocol in place of multicast broadcast. If one or more WKA is specified, for a member to join the cluster it will either have to be a WKA or there will have to be at least one WKA member running. Additionally, all cluster communication will be performed using unicast. If empty or unspecified multicast communications will be used. Only used for **executionMode CLUSTERED**
tangosol.coherence.wka.port NEW	Port numbers associated with WKA addresses in tangosol.coherence.wka. Only used for **executionMode CLUSTERED**

[3] This behavior can be overridden by specifying the property **tangosol.coherence.localport.adjust=false**

This file can be modified for site-specific values. For example, the user Id will need to be changed from "AUSER", so property **com.splwg.batch.submitter.userId** should be modified to specify the appropriate user id allocated to batch. The user id property could also be removed so that there is no default, forcing every job submission to specify its own user id.

Job Specific parameters files

Note: Not ALL jobs require a job specific parameter file. It is recommended that ONLY jobs that require any of the additional parameters listed below should have a job specific parameter file.

For individual jobs it is possible to create a specific file to handle the job specific parameters. These parameters can override existing parameters or add additional parameters.

To configure individual job specific parameters configuration file named **<batchcode>.properties** or **<batchcode>.properties.xml** must existing in the **$SPLEBASE/scripts/cm** directory (or **%SPLEBASE%\scripts\cm** directory on Windows). In the vast majority of cases the standard text based properties file will be adequate, but if UTF-8 formatted soft parameter values have to be specified (e.g. Cyrillic characters), it is recommended the xml file format be used.

The format of the job specific parameter file is similar to the **submitbatch.properties** file with the following additional parameters:

com.splwg.grid.executionMode	Mode of execution. Valid values: **THIN**, **DISTRIBUTED** or **CLUSTERED**.
com.splwg.batch.submitter.distThreadPool	Name of pool to be used for job. If **threadpoolworker** not used then LOCAL must be specified.
com.splwg.batch.submitter.batchCd	Batch Code this job is associated with.
com.splwg.batch.submitter.processDate	Business Date. This may be omitted. Format: **YYYY-MM-DD**
com.splwg.batch.submitter.maximumTimeoutMinutes	Specifies the number of minutes the thread(s) can run between database commits. Only COBOL programs, if used, use this value to avoid "snapshot too old" errors on Oracle databases. Java batch classes completely ignore this parameter.
com.splwg.batch.submitter.traceProgramStart	Set this to **true** to see program start messages in the log.
com.splwg.batch.submitter.traceProgramEnd	Set this to **true** to see program end messages in the log.
com.splwg.batch.submitter.traceSQL	Set this to **true** to see all SQL

	statements in the log.
com.splwg.batch.submitter.traceStandardOut	Set this to **true** to see program debug messages in the log.
com.splwg.batch.submitter.softParameter.*<parmname>*	For any program-specific parameters, use this form of property specification. The *<parmname>* denotes the name of the parameter. For example, to specify a "number of rows to skip" when submitting a validation program: **com.splwg.batch.submitter.softParameter.SKIP-ROWS=1000** Multiple soft parameters may be specified.

A number of samples in **$SPLEBASE/etc** directory (or **%SPLEBASE%\etc** directory on Windows) named **SAMPLE.properties** and **SAMPLE.properties.xml** are provided. These should be copied and edited to provide site specific values.

Note: If Patch 7642944 is installed NEW the environment variables may be substituted in the command line and the properties file. Environment variable references must be surrounded by the ${…} construct. For example **${SPLOUTPUT}** refers to the **$SPLOUTPUT** (or **%SPLOUTPUT%** on Windows) directory.

submitjob[.sh] Command-Line Options

Note: that the appropriate environment has to be attached to before this script can be executed (i.e. **splenviron[.sh] -e <environment>** has to be run), unless the script is directly invoked from the Windows explorer by double-clicking on it. In that case it will automatically attempt to attach to the environment that owns the bin directory in which it is located and then prompt for options.

The following options can be specified when executing utility **submitjob[.sh]**:

submitjob[.sh] [-B][-b][-c][-d][-e][-f][-g][-h][-i][-J][-l][-L][-m] [-n] [-p][-P][-Q][-R][-r][-s][-t][-u][-x][-X]

where command line options are:

-B *<COBOL program name>*	Batch "helper" COBOL program to perform scheduling activity.
-b *<batch code>*	Batch code of the job to submit. When submitting a job, a batch code is always required. Either this option or **–P** may be specified, not both. If this option is specified, **submitjob** will use the supplied batch

	code to look for a default properties file for that batch code (e.g. **VAL-SA.properties** or **VAL-SA.properties.xml** as discussed above) and use those properties if found.
`-c <thread count>`	Concurrent number of threads in which to run the process.
`-d <date>`	Process / business date. Format is **YYYY-MM-DD**
`-e <DISTRIBUTED|THIN|CLUSTERED>`	Execution mode for this submission. If execution mode THIN is selected, the JVM will create a full application context and run the job inside the JVM – i.e. it will not be submitted to a thread pool for a worker JVM to pick up and run. This is analogous to running the job using the existing **SPLBATCH[.sh]** utility. If **DISTRIBUTED** or **CLUSTERED** is selected, the job will be submitted to run in the specified distributed thread pool (option **-p**). It is also possible to have the submitter JVM be a worker JVM and run the job (similar to THIN mode, but in parallel threads). See option **-L**.
`-f <record count>`	Record commit frequency count.
`-g <four Y/N switches>`	Positional tracing switches: 1) Program entry 2) Program exit 3) SQL statements 4) General program debugging info For example, NNYN will trace all SQL statements
`-h`	Show help information. Display the available options and their descriptions. The information is not logged.
`-i <RMI port number>`	Port number of RMI Registry to start and/or reference. If specified with **-R**, this number will be used only to substitute applicable URL **{port}** references. This option will not add any new RMI/JMX properties - it can only be used to override existing ones. See note below
`-J`	Do not start JMX connector. This option disables JMX monitoring for this JVM. As far as submitjob is concerned, options **-i**, **-R** and **-J** are only applicable to jobs submitted in **THIN** mode, or **DISTRIBUTED** or **CLUSTERED** mode to the LOCAL thread pool. For each property prefixed by **spl.runtime.management.connector.url** that is defined

	with the default set of properties (e.g. in the **submitbatch.properties** file), the framework will start a JMX Connector for the specified URL. This activates JMX monitoring inside the worker node so that a client JMX console can be used to monitor and manage active threads. If this option is specified, the framework will not start any JMX connectors.
-l <ENG|FRA|etc.>	Language code. Relevant language pack must be installed.
-L	Submit this batch to the LOCAL thread pool (i.e. this JVM). Only applicable for **DISTRIBUTED** or **CLUSTERED** mode. If specified, any default thread pool property is ignored. This option and **-p** are mutually exclusive. By specifying option **–L**, the job is submitted to the LOCAL thread pool that every submitter JVM offers by default. This option is only applicable in a **DISTRIBUTED** or **CLUSTERED** mode execution (**-e**). This is similar to submitting the job in THIN mode (i.e. a worker JVM is not needed to run the job), except thread pool LOCAL can run multiple batch threads concurrently. For example, the following command will run job VAL-SA inside this submitter JVM (LOCAL thread pool) in 8 threads concurrently: **submitjob[.sh] –b VAL-SA –c 8 –L –e DISTRIBUTED**
-m *<number of minutes>*	Minutes between database commits to avoid "snapshot too old" errors.
-n *<email address>*	Send a notification email when a batch job has ended to *<email address>*. See "Sending emails at the conclusion of batch jobs" for more information.
-p *<threadpool name>*	Distributed thread pool in which to run the batch process. This option and **-L** are mutually exclusive.
-P	Issue console prompts for the standard job parameters. When submitting a job, a batch code is always required. Either this option or **–b** may be specified, not both. If **–P** is specified, the submitter JVM will prompt for the batch code and other run parameters. If a batch-specific properties file exists for the batch code entered at the prompt, it will NOT be used; the only defaults in effect would be the ones specified in **submitbatch.properties.**
-Q	Preview the properties that would be in use for the run without actually running the application. Specify other options along with this option to show how they would override or substitute the default properties. The information is not logged.
-R	Do not start a local RMI registry. As far as **submitjob** is concerned, options **–i**, **-R** and **–J** are only applicable to jobs

	submitted in **THIN** mode, or **DISTRIBUTED** or **CLUSTERED** mode to the LOCAL thread pool. If property **spl.runtime.management.rmi.port** is defined as a default property (e.g. in the **submitbatch.properties** file), the batch framework will attempt to start an RMI registry on the given port number. This option can be used to suppress the automatic RMI registry startup. It may be required if an externally started RMI registry is already running. Note that if this option is used, the RMI port number supplied through the **-i** option is only used for substitution in the JMX Connector URLs.
-r *<run number>*	Run number of job to rerun.
-s *<space name>*	Space name for "hard partition" of workers. Default is **MAIN**. Used for development only.
-t *<thread number>*	Number of individual thread for this submission. Specify 0 to automatically submit all threads.
-u *<user id>*	Application user id used for job
-x *<name=value,name=value,...>*	Name=value pairs of INDIVIDUAL soft parameters expected by the batch program. Value portion may be enclosed in quotes. These parameters will be merged with any existing (defaulted) soft parameters. This option and -X are mutually exclusive.
-X *<name=value,name=value,...>*	Name=value pairs of ALL soft parameters expected by the batch program. Value portion may be enclosed in quotes. These parameters will replace all existing (defaulted) soft parameters. This option and **-x** are mutually exclusive.

Contents

Property Override Order

When **submitjob.[sh]** is invoked, it can accept a number of command-line options to alter its default configuration. The default configuration options come from internal system defaults, the **submitbatch.properties** file or the batch specific properties file as described above.

The properties are overridden in the following order:

1. The **submitbatch.properties** supersedes the internal system defaults.
2. The batch-specific properties (e.g. **VAL-LL.properties**) supersede the **submitbatch.properties** and the internal system defaults.

3. The command-line options supersede the defaults in **submitbatch.properties**, the job-specific properties and the internal system defaults.

Port number of RMI Registry (-i)

As far as **submitjob[.sh]** is concerned, options **-i**, **-R** and **-J** are only applicable to jobs submitted in THIN mode, or DISTRIBUTED mode to the LOCAL thread pool.

This option specifies the port number to:

- Use when the framework starts an RMI Registry and
- Substitute in all JMX Connector URL {port} references.

For example, given the following properties in **threadpoolworker.properties**

```
spl.runtime.management.rmi.port=9999
spl.runtime.management.connector.url.default=service:jmx:rmi:///jndi/rmi://{host}:{port}/spl/fw/jmxConnect
```

or and this command-line

```
submitjob[.sh] -i 1099
```

will cause the value for property **spl.runtime.management.rmi.port** AND the "**{port}**" string in the **spl.runtime.management.connector.url.default** property to be substituted.

Note: that in this case, the special substitution string "**{host}**" will also be replaced by the name of the host machine. Therefore, assuming the host name is "localhost", the properties will therefore be modified to look as follows:

```
spl.runtime.management.rmi.port=1099
spl.runtime.management.connector.url.default=service:jmx:rmi:///jndi/rmi://localhost:1099/spl/fw/jmxConnector
```

Soft Parameters (-x) vs (-X)

For jobs that have multiple soft parameters, this option controls how the soft parameter properties are managed. For example, assume the following soft parameter properties are defined for batch job XXXX in its XXXX.properties file:

```
com.splwg.batch.submitter.softParameter.FILE-PATH=C:\data
com.splwg.batch.submitter.softParameter.FILE-NAME=default.dat
```

These soft parameters are therefore the defaults for this job if it is submitted with no command-line options. The following command will submit the job with a new **FILE-NAME**, but leave the **FILE-PATH** as the default:

```
submitjob[.sh] -b XXXX -x FILE-NAME=newfile.dat
```

In other words, only the specified soft parameter (**FILE-NAME**) has been overridden.

It may be necessary in some cases to replace ALL the soft parameters with the ones specified, particularly where there are many soft parameters and only one or two are required. This command will submit the above job with a new **FILE-NAME**, but remove the **FILE-PATH** property for this execution:

```
submitjob[.sh] -b XXXX -X FILE-NAME=newfile.dat
```

Environment Variable substitution at runtime

Note: To use this facility Patch 8206129 should be installed NEW.

At runtime it is possible to substitute local variables by the individual thread parameters at runtime. Three new local variables will be added that will be replaced at the thread level:

- **{threadNumber}** – will be replaced by the number of the executing thread
- **{processDate}** – will be replaced by process date
- **{processDateTime}** – will be replaced by process date time

These variables can be used in the soft paramaters. For example, if the process date is 01-31-2009 1:30 PM and the current thread is thread number 1, specifying the parameter **FILE-NAME** as:

-x FILE-NAME=MYOUTPUTFILE-{processDateTime}-{threadNumber}

would result in the **FILE-NAME** parameter being resolved to:

MYOUTPUTFILE-2009-01-31-13.30.00-1

Return Codes

The following return codes apply to the processing using this method:

Return Code	Usage
0 (zero)	Successful
Non-zero	Unsuccessful. See log files for more information.

jmxbatchclient[.sh] – JMX batch command line

Note: To use this facility Patch number 8222333 must be installed.

While the JMX client interface provided allows real time information to be displayed in a JMX "browser", if a JMX "browser" is not used then the JMX interface may be interfaced using a command line utility. This utility is useful to allow 3rd party products (such as batch schedulers) or other systems to control and monitor the state of the system.

This JMX batch command line allows the following to be performed:

- Identify what thread pools are defined in a threadpoolworker
- See what active jobs or threads are currently running
- Be able to cancel a particular thread or a batch job
- Gracefully shutdown a threadpoolworker
- The command line utility is in the following format:

To execute the command line, the administrator must:

- Logon to the machine running the product (any tier where the product software exists).

- Attach to the environment using the **splenviron[.sh]** command. This sets the appropriate environment settings for the script.
- Execute the JMX Batch command line utility:

jmxbatchclient[.sh] –j [URL] [options]

where options are:

-c Cancel Active Threads

-d Display Details

-f Filter results (uses Regular expressions)

-h Show help

-j JMX URL to perform the action against *(Required)*

-k Shutdown threadpoolWorker

-l Suppress ALL logging

-s Display Summary Information

Contents

Help -h:

Display the available options and their descriptions.

JMX URL-j:

Specify the JMX URL to connect to. This should match the **spl.runtime.management.connector.url.default** property specified in the **threadpoolworker.properties**.

For example:

```
jmxbatchclient.sh –j
service:jmx:rmi:///jndi/rmi://myserver:9999/spl/fw/jmxConnector
```

Cancel Active threads -c:

Specifies that active threads should be cancelled. Can be used with –f option to cancel only jobs matching the regular expression provided. For example:

```
jmxbatchclient.sh -j
service:jmx:rmi:///jndi/rmi://myserver:9999/spl/fw/jmxConnector -f .*BAT1.* -
c
```

Note: Cancelled threads are "marked" with the date, time, userid and IP address of the user who initiated the cancel command.

Display details -d:

Display the details of the currently active threads.

```
jmxbatchclient.sh -j
service:jmx:rmi:///jndi/rmi://myserver:9999/spl/fw/jmxConnector -d
```

```
Connecting to service:jmx:rmi:///jndi/rmi://myserver:9999/spl/fw/jmxConnector
ActiveGridNode
threadPools=[MYSERVER:5, SCHEDULER_DAEMON_THREAD_POOL:1,
LOCAL_THREAD_POOL:b9835d11f15fd71b:1df6824f:120011dc94e:-8000:0]
BatchThread_ZZQABAT2_1_of_1.36
ElapsedTime = 0 yrs. 0 days 00:00:20
BatchCd = ZZQABAT2
ThreadCount = 1
ThreadNumber = 1
RerunNumber = 0
ProcessDate = 2009-03-13
MaximumCommitRecords = 200
MaximumTimeoutMinutes = 0
UserId = SYSUSER
LanguageCd = ENG
SoftParameters = []
MaxExecutionAttempts = 1
DateTimeStarted = 2009-03-13-11.54.02
DistThreadPool = MYSERVER
BatchNumber = 4
Status = Running
ProgramType = Java
ProgramName = com.splwg.cm.domain.qa.batch.QaBatch2
RunType = New Run
```

Regular expression filter -f:

If a large number of threads are currently active, a filter can be supplied to only display or cancel threads that match the pattern.

For example, if two active threads were displayed as follows:

```
jmxbatchclient.sh -j
service:jmx:rmi:///jndi/rmi://myserver:9999/spl/fw/jmxConnector -d

Options: -j service:jmx:rmi:///jndi/rmi://myserver:9999/spl/fw/jmxConnector

Connecting to service:jmx:rmi:///jndi/rmi://myserver:9999/spl/fw/jmxConnector

ActiveGridNode

threadPools=[MYSERVER:5,
LOCAL_THREAD_POOL:b9835d11f15fd71b:681ba91d:1200151a3c8:-8000:0,
SCHEDULER_DAEMON_THREAD_POOL:1]

BatchThread_ZZQABAT1_1_of_1.31

BatchThread_ZZQABAT2_1_of_1.32
```

This can be filtered to show only the BAT1 with the option: -f .*BAT1.* as follows:

```
jmxbatchclient.sh -j
service:jmx:rmi:///jndi/rmi://myserver:9999/spl/fw/jmxConnector -f .*BAT1.*

Options: -j service:jmx:rmi:///jndi/rmi://myserver:9999/spl/fw/jmxConnector -
f .*BAT1.*

Connecting to service:jmx:rmi:///jndi/rmi://myserver:9999/spl/fw/jmxConnector

ActiveGridNode

threadPools=[MYSERVER:5,
LOCAL_THREAD_POOL:b9835d11f15fd71b:681ba91d:1200151a3c8:-8000:0,
SCHEDULER_DAEMON_THREAD_POOL:1]

BatchThread_ZZQABAT1_1_of_1.31
```

Kill threadpoolworker -k:

Specifying this option will result in the cancellation of all currently running threads and the stoppage of the **`threadpoolworker`** process.

For example:

```
jmxbatchclient.sh -j
service:jmx:rmi:///jndi/rmi://myserver:9999/spl/fw/jmxConnector –k
```

Note: Active threads within a cancelled **`threadpoolworker`** are "marked" with the date, time, userid and IP address of the user who initiated the kill command.

Suppress logging -l:

By default, all logging information is displayed and logged using log4j. Supplying this option will result in only select information being displayed to the system output.

For example:

```
jmxbatchclient.sh -j
service:jmx:rmi:///jndi/rmi://myserver:9999/spl/fw/jmxConnector -l
```

Display summary -s:

Display the summary of the currently active threads is a listing format.

For example:

```
jmxbatchclient.sh -j
service:jmx:rmi:///jndi/rmi://myserver:9999/spl/fw/jmxConnector -s

Options: -j service:jmx:rmi:///jndi/rmi://myserver:9999/spl/fw/jmxConnector -
s

Connecting to service:jmx:rmi:///jndi/rmi://myserver-
us:9999/spl/fw/jmxConnector

ActiveGridNode

threadPools=[MYSERVER:5, SCHEDULER_DAEMON_THREAD_POOL:1,
LOCAL_THREAD_POOL:b9835d11f15fd71b:-60bf2fc1:120115996cc:-8000:0]


JMX Id DistThreadPool BatchNumber DatTimeStarted ElapsedTime

BatchThread_ZZQABAT1_1_of_1.29 MYSERVER 32 2009-03-16-17.07.29 0 yrs. 0 days
00:03:49
```

Sending emails at the conclusion of batch jobs

It is possible to send a notification email when a batch job has ended. This notification happens after the job has ended and all application-related commits/rollbacks have taken place. It does not impact the batch job itself in the event of errors happening during the notification process. The default email is a simple text email that contains the batch control, date and time of the submission, run number, submission parameters, job summary indicating records processed and in-error, as well as the thread details, including logged messages (up to 100).

The email address can be configured a number of ways:

- **Online submission** – The email address can be specified on the batch submission screen.
- **Lightweight Framework** - The email address is specified on the **–n** option of the **submitjob[.sh]** command.

The email address can be an individual person or a valid mail group (*the latter requires additional configuration in your email system*).

To use email notification the email server must be configure using one of the following options:

1. The mail server can be defined through the default XAI Sender (see XAI Options in the XAI documentation)with the appropriate SMTP settings on the Context tab.
2. Alternatively the properties can be supplied in the form of JVM properties as follows:

```
# Host whose mail services will be used

# (Default value : localhost)

mail.host=<your mail server>

# Return address to appear on emails

# (Default value : username@host)
```

```
mail.from=<name@host>

# Other possible items include:
# mail.user=
# mail.store.protocol=
# mail.transport.protocol=
# mail.smtp.host=
# mail.smtp.user=
# mail.debug=
```

Name	***Type***	***Description***
mail.debug	boolean	The initial debug mode. Default is false.
mail.from	String	The return email address of the current user, used by the `InternetAddress` method `getLocalAddress`.
mail.mime.address.strict	boolean	The MimeMessage class uses the `InternetAddress` method `parseHeader` to parse headers in messages. This property controls the strict flag passed to the `parseHeader` method. The default is true.
mail.host	String	The default host name of the mail server for both Stores and Transports. Used if the `mail.`*`protocol`*`.host` property isn't set.
mail.store.protocol	String	Specifies the default message access protocol. The `Session` method `getStore()` returns a Store object that implements this protocol. By default the first Store provider in the configuration files is returned.
mail.transport.protocol	String	Specifies the default message access protocol. The `Session` method `getTransport()` returns a Transport object that implements this protocol. By default the first Transport provider in the configuration files is returned.
mail.user	String	The default user name to use when connecting to the mail server. Used if the `mail.`*`protocol`*`.user` property isn't set.
mail.smtp.class	String	Specifies the fully qualified class name of the provider for the specified protocol. Used in cases where more than one provider for a given protocol exists; this property can be used to specify which provider to use by default. The provider must still be listed in a configuration file.
mail.smtp.host	String	The host name of the mail server for the specified protocol. Overrides the `mail.host` property.
mail.smtp.port	int	The port number of the mail server for the specified protocol. If not specified the protocol's default port number is used.
mail.smtp.user	String	The user name to use when connecting to mail servers using the specified protocol. Overrides the `mail.user` property.

These properties can be added to the **threadpoolworker.properties** file for the standalone batch **threadpoolworker**, or the **spl.properties** file for an online application server that hosts a batch worker.

Template Overrides NEW

Note: For this facility to be used you must install patch 8813246 on your environment.

By default, some of the configuration files outlined in this document are generated from product templates. The scripts provided with the product for use during installation, patching and configuration regularly rebuild the configuration files from templates. This can cause any manual changes to configuration files to be reset to the base templates, which may mean loss of customizations if backups of the configuration files are not taken.

The Oracle Utilities Application Framework now features a facility where a site may substitute their own templates, based upon the product templates. This allows sites to customize their copies of the custom templates to suit their site standards and retain their settings across upgrades and patches.

The process to use this facility is as follows:

- Make a copy of the template used for the relevant configuration file and prefix the copy with cm.. Use the table below to identify the template used for the relevant configuration file (templates are stored in the **etc** directory of the product environment):

Configuration file	*Template*	*Custom template name*
spl.properties	spl.properties.standalone.template	cm.spl.properties.standalone.template
hibernate. properties	hibernate. properties.template	cm.hibernate.properties.template[4]
log4j.properties	log4j.properties.standalone.template	cm.log4j.properties.standalone.template
e0batch.properties	e0batch.properties.template	cm.e0batch.properties.template

- Make the site specific changes necessary for your site. Remember to use the structure and the environment variables in the template as a guide for the format.
- Save the template.

Once this is done, if the configuration files are ever generated manually or as part of a patch then they will use the custom template instead of the base template.

Note: If this facility is used, then it is the site's responsibility to maintain the custom template in line with the product template. If future fixes add additional facilities to base templates then those changes must be manually applied to any custom templates. Check any custom templates against the base templates on a regular basis.

[4] The **hibernate.properties** template is shared across online, XAI and batch so any template changes will be reflected global changes.