

A faint, light-grey background diagram showing a control loop. It includes several circular nodes and arrows indicating the flow of signals. One node at the top left has two incoming arrows. A horizontal line with an arrow points from a node on the left to a node on the right. Below this, a curved arrow points from the left node to a central node. From the central node, an arrow points to another node on the right, which then has an arrow pointing back to the top-right node. Another curved arrow points from the bottom node back to the central node.

ENGINE TUNNING IN FUEGO 5.5

TUNNING THE FUEGO 5.5 STANDALONE ENGINE

Pablo Victory
pvictory@fuego.com

May 9, 2005

Contents

1 Overview	4
2 Configuring a Fuego Engine	5
2.1 Engine Locations	5
2.1.1 Advanced Properties	7
2.2 Database Configuration	8
2.3 Log	11
2.4 Execution	13
2.4.1 Startup	14
2.4.2 Memory	15
2.4.3 Threads	18
2.4.4 Timeouts	23
2.5 Services	26
2.5.1 Disposer	26
2.5.2 IPC	28
2.5.3 Socket Factory	29
2.5.4 SNMP	29
2.6 Networking	30
2.7 Others	31
2.7.1 Runtime	31
2.7.2 Directory	31
2.7.3 Events	32
2.7.4 PAPI	33
3 Other Performance Considerations	34
3.1 User Interaction	34
3.2 External Components	35

A Appendix: Fuego RMI	36
A.1 What is Fuego RMI?	36
A.2 What are the transport layers protocols provided by Fuego?	36
A.2.1 TCP Protocol Transport Layer Implementation	36
A.2.2 MMP Protocol Transport Layer Implementation . . .	36
A.2.3 SSL Protocol Transfer Implementation	36

1 Overview

This document describes the different configuration parameters available for a Fuego engine. It also provides tuning recommendations and tips aimed at improving Fuego engine performance.

These parameters govern how a Fuego engine behaves and performs. Understanding the various parameters and how they interact is critical in understanding how to configure and tune the engine for any given solution.

Most of the tuning takes place in Fuego's Web Console. This Fuego web application provides several panels that can be used to tune different aspects of the engine. The correct configuration of all these values will provide better overall performance for the processes being executed in FuegoBPM. Through out this document we assume that you have already created an engine.

2 Configuring a Fuego Engine

Open your Web Console and select the *Engines* link. A list of all the available engines is displayed on the right panel. Select the engine you want to configure to access the properties page for that particular engine.

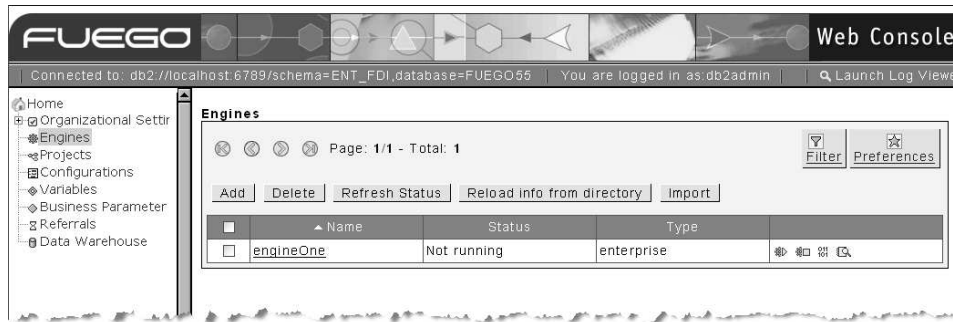


Figure 1: Select the engine to be configured from the list of available engines

2.1 Engine Locations

Once you accessed the desired engine, you will see its properties starting with the *Basic Configuration* tab. From this page you will have to select the *Locations* link which will take you to the list of available locations for this engine.

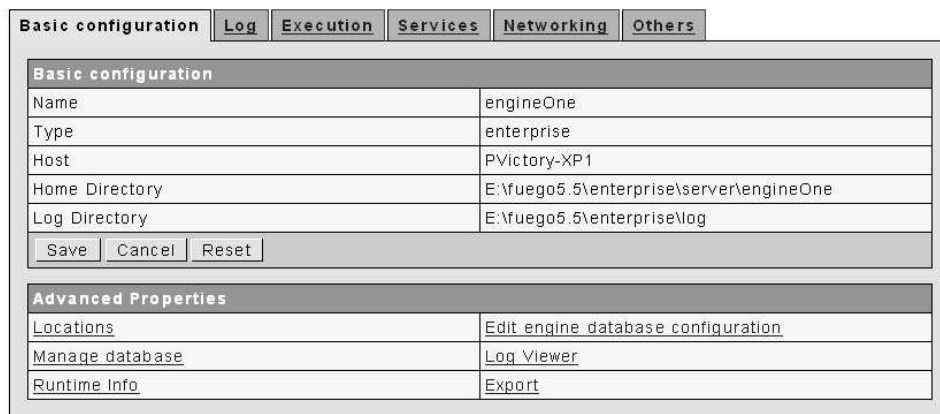


Figure 2: Select the *Locations* link from the *Basic Configuration* tab

Typically an engine will run from a single location, but if you want to use the failover capabilities of Fuego you will need to define another location where the failover engine will run¹.

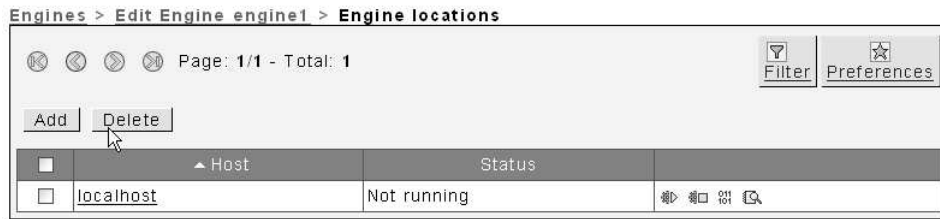


Figure 3: List of available locations

You can now select a location to access its properties.

Location configuration	
Host	PVictory-XP1
Home Directory	E:\fuego5.5\enterprise\server\engineOne
Log Directory	E:\fuego5.5\enterprise\log
Protocol	ssl
Port	10099
Web Console Protocol	http
Web Console Port	8585
<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Reset"/>	

Figure 4: Location properties

- **Host**

Each location will have a host defined as the *Host* property where the Fuego engine can be executed. The rest of the parameters will determine the protocol and properties associated to the connection to be used when a client connects to the Fuego engine.

- **Home Directory**

The home directory for the engine. By defining a path as home directory, the administrator can determine where to place resources that are generated for exclusive use by this Engine. For example, this is

¹For a complete description on the failover capabilities see the *Fuego Failover* guide.

often used when you want to reserve a library for exclusive use by a specific Engine.

- **Log Directory**

The directory where the engine log will be generated.

- **Protocol**

This attribute can be either **mmp** (for multiplex protocol), **tcp** (for TCP/IP protocol), **local** or **ssl** (for secure socket layer). See [Appendix A](#) for a complete description of each protocol.

- **Port**

Fuego Remote Method Invocation (RMI) port. The engine will use this port number to receive incoming connections from the clients trying to perform an operation on him.

- **Web Console Protocol**

Select the protocol to use by the Web Console. You can choose plain *http* or the more secure *https*.

- **Web Console Port**

The port to be used by the Web Console. by default this value is 8585.

2.1.1 Advanced Properties

The following is a brief description on a location's advanced properties. In most cases the values provided are adequate and don't need to be changed.

Properties	
External request queue size	<input type="text" value="300"/>
Maximum external request latency	<input type="text" value="30"/> Seconds
Maximum number of connections per engine	<input type="text" value="50"/>
Maximum number of connections per external agent	<input type="text" value="5"/>
Handshake timeout	<input type="text" value="1"/> Seconds
Protocol additional parameters	<input type="text"/>
<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Reset"/>	

Figure 5: Location advanced properties

- ***External request queue size***
Every request that gets to this location through PAPI will be placed in a queue. This parameter lets you configure the size of this queue.
- ***Maximum external request latency***
Every request placed in the queue will wait a certain amount of time before it is answered back with a *ServerBusyException*. This parameter lets you configure the amount of time before a request is answered back with such exception.
- ***Maximum number of connections per engine***
This parameter sets the maximum number of connections the engine will accept.
- ***Maximum number of connections per external agent***
This is the maximum number of connections the engine will grant to each PAPI client connecting to it. Usually PAPI clients use two connections.
- ***Handshake timeout***
This parameter specifies the maximum amount of time that the engine will wait after it accepted the connection and before the client identifies itself as a Fuego client. If the identification can not be performed successfully in this amount of time then the engine will close the connection.
- ***Protocol additional parameters***
Any additional communication parameters should be specified here.

2.2 Database Configuration

The engine's database configuration has a huge impact on the overall performance of the engine. Each engine needs to persist information to the database each time a FBL is executed or when an instance needs to be routed to the next activity so the state of any process instance can be recovered at any point in time. In order to ensure a maximum level of performance, configure the connection pool of the engine accordingly so that connections are available when needed.

The engine's database configuration properties can be accessed through the *Edit engine database configuration* link.

Basic configuration		Log	Execution	Services	Networking	Others
Basic configuration						
Name	engineOne					
Type	enterprise					
Host	PVictory-XP1					
Home Directory	E:\fuego5.5\enterprise\server\engineOne					
Log Directory	E:\fuego5.5\enterprise\log					
<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Reset"/>						
Advanced Properties						
Locations	Edit engine database configuration					
Manage database	Log viewer					
Runtime Info	Export					

Figure 6: Select your engine's configuration

Once you click on the link, the database configuration parameters are displayed. Of particular interest for tuning the engine performance are the parameters contained in the *Runtime* section.

Type	
Name	Engine 'engineOne' database configuration
Type	SQL Database
Subtype	IBM DB2 JDBC (Type 3) Change subtype
Properties	
Host	<input type="text" value="localhost"/>
Port	<input type="text" value="6789"/>
Schema	<input type="text" value="ENT_ENGINE"/>
Database	<input type="text" value="FUEGO55"/>
User	<input type="text" value="db2admin"/> Change Password
Runtime	
Maximum Pool size	<input type="text" value="10"/>
Connection Idle time (Mins)	<input type="text" value="5"/>
Maximum opened cursors	<input type="text" value="50"/>
<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Reset"/>	

Figure 7: The link displays the parameters to be configured

- **Maximum Pool Size**

This parameter should reflect the number of concurrent interactive

users. This is the determining factor. Make sure the database has that many client connections configured ready to be consumed by Fuego. This number should reflect the combined number of interactive thread plus the number automatic execution threads that can be concurrently active.

Do not define a huge value (i.e.: 400) for this parameter since this will also create contingency in the RDBMS used by the Fuego Engine. It is preferable to wait for a connection to be released after a transaction is finished than to generate a big concurrency of transactions in the target RDBMS. There must be a trade off between how soon the transactions can be finished without generating a bottleneck and the contingency in the database. This will also depend on the hardware where your RDBMS is deployed and the dimensioning of the RDBMS used by the Fuego Engine.

Make sure that the database is configured so that there are enough sessions to handle the number of maximum connections that the Fuego Engine may use.



Some Oracle RDBMS recommendations:

- Make sure you have enough sessions. Check Oracle's SESSIONS parameter. For no contingency, you should have a session for each Engine connection that the Fuego Engine may use for a transaction.
- Make sure there are enough processes on the Oracle side. Check Oracle's PROCESSES parameters. This parameter depends also on the values assigned to SESSIONS and TRANSACTIONS. For no contingency, you should have a process for each Fuego Engine connection that can be executing a transaction at the same time.
- Make sure the Oracle memory is properly configured for the number of concurrent transactions to be executed by a Fuego Engine. The dimensioning of the memory in Oracle is related to the configuration of the SGA.

Fuego only uses from the SGA the following sections:

- * BUFFER CACHE
- * LARGE POOL
- * SHARED POOL

- *Connection Idle Time*

Fuego ensures that a connection does not remain open and idle more than the amount of time specified by this parameter. Usually, if the connections are not closed, the associated resources (such as cursors or statements) are not released by the JDBC Driver. This is why if the concurrency is not too high this value should not be too high either.

- *Maximum Opened Cursors*

This parameter specifies the maximum number of cursors that a session can open at one time. This value should be greater than the number of cursors that may be opened by a Fuego Business Method triggered from an activity task.

2.3 Log

The engine's logging can significantly affect the performance of an engine if not configured properly. Fuego's logging mechanism has the ability to differentiate between engine messages and FBL messages that are dumped into the engine's log through the usage of the `logMessage` FBL statement.

Properties	
Messages logged from Engine	Warning
Messages logged from BP-Methods	Info
Messages sent by mail	None
Log detail level	1 (min=1)
Maximum size of log file	2000 kb
Maximum number of log files	5

Save Cancel Reset

Figure 8: Log configuration parameters

- *Messages logged from engine*

This parameter dictates which messages will be logged and which ones will be ignored and not included in the log. The messages logged will be those with severity equal to that specified by this parameter or greater. In QA or Production environments, set this value to **Warning** since warning level messages (or greater) should not be disregarded.

since they bring important situations to the system administrator's attention. This value may be set to **Debug** in QA or Production environments when trying to identify a problematic situation, but under normal running conditions **Debug** message should be avoided since it will slow down the overall engine performance.

- *Messages logged from BP-Methods*

The same rules as in the engine messages apply here since logging unnecessary data into the log degrades overall engine performance. The proper log level can be set when invoking `logMessage` FBL statement from FBL. The following example shows how to log a message specifying a **Debug** level.

```
1 logMessage "Executing Initial Customer Info"
2     using severity = DEBUG
```

- *Messages sent by mail*

This parameter works in a similar manner as the previous ones. It will define which messages will be sent by email. In order to receive the messages you have to have the *Networking* section properly configured (see below).



Mail Bombing Warning: Have in mind that if you have this value improperly configured you can quickly saturate any mail server with messages. In production or QA environments this value should never be set below **SEVERE**.

- *Log detail level*

This configuration parameter should be set to 10 when working in development environments. Messages dumped by the engine in the log have different detail levels. Important messages usually have a low detail level (i.e., 1). Higher detail levels are usually implemented when **DEBUG** messages are enabled. In a production environment this value should be set to 1 unless you are trying to analyze a particular problem, then it can be set to 10 to get as much information as possible.

- *Maximum size of log file*

This configuration parameter determines how large each one of the engine log files can be. The size of the log can have also a huge impact

on the performance of the engine. Adding information to a big file takes considerably more time than adding information into a small file. By default, the engine log file is set to 2MB (2000 KB), but if needed this value can be decreased and the number of files increased if the same amount of data needs to be logged.

When the log files run out of space, the Engine will delete the oldest logs and re-use the files.

- **Maximum number of log files**

This configuration parameter represents the number of engine log files. The engine stores information in the log files in a cyclic way among all engine log files. So if you have a file size of 1000 (KB) and you set this value to 10 then you will effectively be logging 10MB of messages. This will log the same amount of data than the default values of 2000 and 5.



Performance Tips

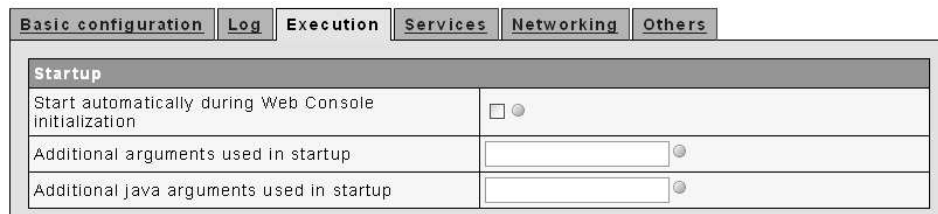
- Try always to use the severity argument when using `logMessage` so that you are conscious of the severity of the message to be logged in the Fuego engine log.
- It is always better to have more log files of smaller size than a reduced number of big log files. The OS will take more time to write on a big file than on a small file.
- Writing to the Fuego Engine log file in each individual transaction can in turn result on a performance degradation problem.
- Do not leave the *Trace Components* option enabled in production environments.

2.4 Execution

There are a number of parameters that directly affect the execution of the Fuego engine. These parameters, from the JVM heap size to the methods timeout, can be configured through the *Execution* tab.

2.4.1 Startup

This section allows the administrator to configure whether Fuego will automatically start the engine and whether additional arguments should be passed during startup to the engine.



The screenshot shows the 'Startup' configuration section of the Fuego Web Console. At the top, there are tabs for 'Basic configuration', 'Log', 'Execution', 'Services', 'Networking', and 'Others'. The 'Startup' section is active and contains three rows of configuration options:

Startup	
Start automatically during Web Console initialization	<input type="checkbox"/> <input checked="" type="radio"/>
Additional arguments used in startup	<input type="text"/> <input type="button" value="⊕"/>
Additional java arguments used in startup	<input type="text"/> <input type="button" value="⊕"/>

Figure 9: Startup properties section

- **Start automatically during WebConsole initialization**
Indicates whether the engine should be started up during the Web Console application initialization. Because the Web Console can be installed as an operating system service, administrators can also configure several engines to be started with the host.
- **Additional arguments used in startup**
This are arguments that will be passed on to the engine during startup.
- **Additional Java arguments used in startup**
This are arguments that will be passed on to the Java JVM during startup.

Parameter	Purpose
-c	Rebuild instance counters. This rebuilds the instance counters that are displayed in the Work Portal in case they have become corrupted. This can happen if the engine has been upnormally shutdown (i.e. the machine was turned off or rebooted with the engine up and running).
-p	Rebuild instance counters for the process specified.
-s	???
-t	Delay (in seconds) before the engine starts shutting down when the command is given.

Table 1: List of available engine arguments

2.4.2 Memory

This section allows the configuration of the memory being allocated to Fuego and its Java JVM.

Memory		
Maximum JVM heap size	256	MBytes
Maximum instance size	16	KBytes
Instances cache	5000	

Figure 10: Memory properties section

- **Maximum JVM heap size**

This field contains the maximum amount of memory that the engine can consume. If for some reason, the engine reaches this limit, it will stop its execution and re-launch itself again so that it starts with a new clean memory. The value associated with this field depends on how many processes are deployed in this Engine and how much memory each process consumes. It's also related to the instance cache and the maximum size of the instances. This value is expressed in megabytes and its default is 256MB.

Always try to set the minimum value for the JVM heap size. You should only increase this value if really needed. Pay attention to the messages related to the memory consumption in the Fuego engine logs. Whenever you are getting to the limit, the engine will start logging **WARNING** messages.

- *Maximum instance size*

This is the maximum amount of memory an instance can consume. This configuration parameter is used to limit the size of all process instance variables defined in a process. Attachments and notes added to an instance in the Work Portal are not included in this size limit. Instance variables defined at design time are the objects that mainly affect this parameter. It is a best practice to only include key elements as process instance variables. In this way, the instance variable serialization executed by the engine at the end of each task execution will take a shorter time. Not only will it take less time but it will also keep consistency with external backend applications being accessed by the business process.

When the size of the instance gets close to this limit, the engine will start warning the administrator about this situation through **WARNING** entries in the engine's log. When any instance exceeds this limit, the engine will not be able to persist the instance data, the task will fail and it will have to be re-executed after resizing this parameter. The error will be logged into the engine's log and if the mail for the administrator is properly configured, an error mail will be sent to him/her. When setting the value of this field, we should make sure that you validate this constraint:

$$\text{Max Instance Size} \times \text{Cache Size} < \frac{\text{Max JVM Heap Size}}{2}$$

This setting is needed to enforce that the instance cache will not consume more than half of the engine's memory. If the max size of the instance does really need to be increased, you should also increase the Max JVM Heap Size value accordingly to validate the previous equation.

It is strongly recommended to check the process instance variables before increasing this value since the larger the objects, the longer it will take to persist instance information and the more memory that it is going to be consumed by the engine due to cache size. Should

your instance size be too large then you may need to redesign your process to reduce the size of the instance. This value is expressed in kilobytes and its default is 16KB.

Always try to define a small value for the instance size. Whenever the size is exceeded, you should analyze if the offending process does really need to be exceeding this value or it should be redesigned. A cause for this limit to be exceeded is due to the usage of arrays in process instance variables. If this is the case, suggest the developer storing only the key in the array and not the whole structure. Always try to minimize its size in the desing and coding side before changing this parameter.



Warning: *The previous parameters are related. The amount of memory used to store instances in memory (Instance max size * cache size) must be less than half the maximum JVM Heap Size. If not, the properties will not be saved and an error message will be displayed on the Web Console.*

- **Instances cache**

This configuration parameter defines the size of the cache for the engine measured in number of instances. This cache is used to store most recently accessed process instances and it is shared by all deployed processes. This means that this cache is engine wide. This value is used in conjunction with the instance max size to check that the engine will not use more than half of its memory in process instance cache.

The default value of 5000 instances in the cache is usually considered above average for medium size installations. If the engine has processes with thousands of instances running and these processes have a high concurrency, it is advisable to increase the size of the cache so that the engine does not need to reload instance information when new instances are accessed.

If you have processes that are using instances concurrently and accessed frequently, you may want to analyze if the instances really need to be loaded back into the cache. Fetching instances from the database very often may incur in performance degradation problems. Again a redesign of your processes may be in order.

2.4.3 Threads

This section allows the administrator to configure the number of threads that will be available for processing every single task in Fuego. Each interactive activity that a user wants to execute and every automatic activity or task that the engine has to execute will use one of the available threads. If there are no threads available, the engine will wait until one is freed so that it can execute the tasks requested. These numbers should be configured in such a way that they reflect the load on your deployed processes. For example, if the engine had only one thread to use then it will only be able to do one thing at a time.

Execution threads		
Maximum number of execution threads used for interactive executions	<input type="text" value="50"/>	
Maximum number of execution threads used for automatic tasks	<input type="text" value="5"/>	
Priority of Automatic Execution Threads	<input type="text" value="5"/>	
Automatic items queue size	<input type="text" value="1000"/>	
Retry times	<input type="text" value="5"/>	
Retry interval	<input type="text" value="1800"/>	Seconds

Figure 11: Threads properties section

- **Maximum number used for interactive executions**

This value indicates how many threads will be attending user requests. These requests are typically represented by interactive activities in your processes. This value depends on your process size, the number of concurrent users, and the amount of interactive tasks that must be processed. In most cases, the default value is sufficient.

In every case and specially if the process has a high end-user concurrency it is highly recommended that the process takes advantage of the *screenflows* feature provided by Fuego. *Screenflows* free the threads being used and allows other threads to take over when waiting for an end user response. When the end user responds, the execution is resumed by another thread. In this way, a limited number of threads can actually handle hundreds of concurrent users.

- **Maximum number used for automatic tasks**

This configuration parameter determines the size of the thread pool

dedicated to automatic tasks (those requiring no end user intervention).

The default size of 5 is configured for highly interactive processes on a medium size installation. If all the processes deployed are mainly automatic and/or you have a large number of instances, it is recommended that this value be increased to at least 10 so that more tasks can be executed concurrently and at the same time. This setting also assumes that the execution time for automated activities is relatively short. If, for some reason, the automatic tasks take more than 5 to 10 seconds and this time cannot be reduced, you will need to increase the number of concurrent automatic execution threads so that more tasks can be executed simultaneously. When increasing this parameter, take into consideration that the thread creation consumes not only time to be launched but also engine resources (especially memory).

It should also be taken into consideration that, if many threads are being used, more administration would need to take place on behalf of the JVM (Java Virtual Machine) executing the engine. This context switching usually can have an Engine performance impact when working with many threads. This is usually the case when working with values over 100. In other cases, it is better to queue rather than increase the number of threads. When there are no more threads to execute a particular task, the task is going to be queued until one thread is freed up to be used.

If for some reason the number of execution threads needs to be increased, we should also check that the engine's maximum pool size (see section 2.2) is also enlarged. This is needed since each execution threads has a transaction with the engine database associated and we need to guarantee that at least each execution thread will be able to open the connection with the database. As such, the following condition should validate:

$$(\text{MaximumDatabaseConnections}) \geq (\#of\text{AutomaticExecThreads} + 30\%\#of\text{AutomaticExecThreads})$$

This formula takes into account that 30% of the Number of Automatic Exec Threads is used by all other connections to the database required for the engine.

- **Priority of Automatic Execution Threads**

When working with Java Threads, you have the ability to determine

the priority on thread groups. The underlying Operating System uses these priorities when a new thread needs to be dispatched. As the Engine is nothing more than a Java application, Fuego takes advantage of this feature so that the execution thread priorities can be set higher than other threads being executed by the Engine (for example component threads). A normal rule of thumb is to set this field to 5 (average) but if the process is highly automated, this value can be increased.

If the priority is set to a higher value, you must be aware that all other jobs (as external connections) might not be attended or the performance might decrease if there are too many automatic tasks.

- ***Automatic items queue size***

This configuration parameter is used to define the automatic task queue size in the engine's memory. All automatic tasks required for all the processes within the engine will be queued. This queue is persisted in the database in case the engine stops running. The engine, loads this queue in memory from the database based on the timestamp of the tasks. If this queue is defined with a very large value, the queue will consume more space but less database access will be required. If the queue is defined with a small value, there will be more database accesses but it will take less memory. A number of 1000 is a good default value since it does not take too much memory and also reduces the number of database accesses. If for some reason, the number of concurrent automated tasks is more than 1000, this configuration parameter can be increased. If the processes are not highly automatic, this value can be decreased to at least 500.

The automatic tasks that are queued are divided into 2 groups:

1. Global or General tasks: these tasks are few and don't really determine the Queue Size
2. Tasks that depend on the number of instances: the number of tasks will grow if the processes within the engine manage a big number of instances and therefore they need to be considered to determine the Queue size

To determine the Queue Size you must analyze the following for all processes that run in the engine:

Global or general tasks: These are tasks that have a low impact in the size of the queue.

- Each Polling Automatic Global Activity
- Each Scheduled Automatic Global Activity
- Each Listening Automatic Global Activity
- One for the daily tasks to be executed
- One for the assignment replacement that might need to take place
- One for the Datawarehouse
- One if the Engine has a scheduled shutdown
- One to refresh the Directory Service
- One for the Engine's Disposer
- One to control the instances ID

Tasks that depend on the number of instances: These are the most important key factors to determine the Queue Size. You must analyze all the processes that run in the Engine and consider all of the following variables considering the *average of number of instances* that might flow simultaneously in *each process*:

- Each Automatic Activity
- If the instances have an expiration
- If the Automatic Activity might regularly exceed the maximum retry times (this will generate a queued task)
- Each IPC Activity, for example the Subflow activity
- Each IPC Interruption, for example if an instance in the Notification Wait expires and is rooted to the Exception.
- If all copies will need to be aborted if the parent instance dies
- Each Wait Notification if the expected instance has not arrived yet to the Wait activity
- If a rollback is required for the instance

To summarize, you must take into account for each process the number of instances and the characteristics of that process based on the above variables.

There is a straight relationship in between the *Automatic Items Queue Size* and the *Number of Automatic Execution Threads* (see section 2.4.3).

If the Queue Size is set to 1000 (and completely full) and the Automatic Threads is set to 10, for example, the Engine will attend the 1000 queued tasks, 10 at a time as maximum.

Have in mind that when the engine is reaching the maximum a **WARNING** is posted to the log (if the usage percentage is low, the log message will appear as **INFO**):

The Engine is running out of Automatic Execution Threads (XX% of the pool is in use). If you are receiving this message with high frequency, it is recommendable to expand the size through the Web Console.

If the engine detects that the database is accessed very often, a **WARNING** is also posted to the log:

The ToDoQueue is being filled from database too often (every XX seconds). This may impact in the performance of the automatic work. If you are receiving this message with high frequency, it is recommendable to expand this queue through the Web Console.

- **Retry times**

This configuration parameter specifies how many times an automatic task that fails should be re-executed. Once this number is exceeded, the engine will raise an exception for this instance and send it to the associated exception handler. This is specifically used for processes containing automatic activities. This value should be set properly and in association with *Retry Interval* (see below). The max retries set by default is 5.

- **Retry interval**

This configuration parameter indicates the time that should pass between automatic task executions, in case an automatic activity task failed.

Configuration of these last 2 parameters depends upon how quickly you want the engine to retry failed tasks.



Missing Instances: *Sometimes an instance seems to vanish from the Portal after being processed by a user. It just does not show up in the expected activity even after waiting for several minutes. Usually the reason*

is that the instance has encountered a problem and its is retrying the specified number of times before showing up as an **exception** in the Portal. For example, if the number of retry times is 6 and the interval is 5 minutes, then half an hour will have to pass before the instance shows up in the exception handling flow.

2.4.4 Timeouts

Timeouts let you control how long the engine can be executing any method and activity in Fuego. Even though timeouts seem like a constraint at first, they actually help the engine protect itself. In case that an external component freezes up or takes too long to process a request, the engine would not be able to release resources and keep that thread running until that component finishes its execution (which may never happen). Timeouts limit the time the engine will wait in this state and help him release resources used by a runaway component.

Timeouts		
Maximum BP-Methods timeout	1800	Seconds
Interactive component timeout	720	Minutes
Maximum Process Web Service Session timeout	300	Seconds

Figure 12: Timeouts properties section

- **Maximun BP-Methods timeout**

This parameter (expressed in minutes) is used to state the Maximum timeout for any CIL execution of any process within the engine. By default the timeout of a method is 30 minutes (expressed in seconds in the WebConsole), but the developer for a particular method can redefine this timeout. The new timeout cannot exceed the maximum BP-Method timeout set in this field.

To determine the maximum you must consider all the methods for all the processes running on the engine; the timeout must always be as short as possible and defined based on the standard situation and not considering the exceptional case.

If for some reason the method does not complete in this period of time, the engine will abort the task's execution and all resources taken

by this task will be freed and rolled back if they provide transactional control. This is used to prevent never-ending task execution usually generated when invoking components that never return the control to the CIL execution.

To overwrite the default method timeout you can add the following statement at the top of the method to set a total execution timeout, for example, of 10 minutes (the set value must be lower than the Maximum BP-Methods timeout set in the Web Console).

```
1 timeout = 10m'
```

When the Engine is reaching this maximum a **WARNING** is posted to the log (if the usage percentage is low a log message will appear as **INFO**):

The engine is running out of Execution Threads (XX% of the pool is in use). If you are receiving this message with high frequency, it is recommendable to expand the size through the Console.

- ***Interactive component timeout***

This value limits the time the engine will wait for an interactive component to be completed. An example of an interactive component is a Fuego Object presentation. The engine allocates resources to each one of these components while waiting for them. Set this parameter to a reasonable value so that those resources can be freed if users don't send a response in the expected time. This value is expressed in minutes and its default value is 720.

- ***Maximum process webservice session timeout***

Fuego allows you to publish as WebServices the creation and notification of process instances. When someone wants to invoke one of these WebServices, he has to get a session from Fuego. The *Maximum process webservice session timeout* lets you specify how long these sessions will be kept alive by the engine while they are not in use. This value is expressed in seconds and its default value is 300.

Take special attention to the timeout values. Although you may have methods that take a long time to execute, you **should not** define a large value for the timeout. If for some reason, there are tasks that cannot complete and

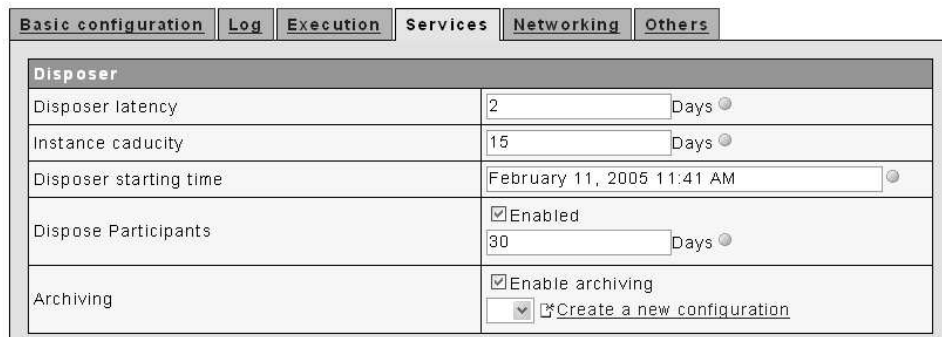
release resources successfully these resources (execution thread for example), will not be released and destroyed until the FBL timeout expires. If for some reason, you need to define a large method timeout value, make sure that normal methods are given a shorter timeout value. To achieve this, you can assign a smaller timeout value in each process activity method.

2.5 Services

The Services tab allows the user to configure how different services in the engine will behave.

2.5.1 Disposer

In this section the administrator can configure several aspects of the engine's *disposer service*. The *disposer service* is in charge of removing instances that have reached the end of every process (either completing it or because they were aborted by the user). These instances are then moved into the archiving database if one is configured or discarded entirely.



Disposer	
Disposer latency	2 Days
Instance caducity	15 Days
Disposer starting time	February 11, 2005 11:41 AM
Dispose Participants	<input checked="" type="checkbox"/> Enabled 30 Days
Archiving	<input checked="" type="checkbox"/> Enable archiving <input type="checkbox"/> Create a new configuration

Figure 13: Disposer properties section

- **Disposer latency**

This configuration parameter determines how often the *disposer service* should be executed (in days). This service checks if deployed processes have instances in the **End** activity for more than a specified amount of time. This amount of time is defined in the *Instance Caducity* configuration parameter (see below). If the instance caducity time has expired, the engine will delete all information about this instance. This is usually useful when there are large volumes of instances being processed by the engine. You need to take into account also the ones that are created in split-join circuits and subflows to determine the total number of instances created.

A large number of instances that have already completed the processes but remain in the **End** activity can have an impact on the over-

all performance of the engine since engine queries to the database will take longer to complete due to the large volume of data stored. If the instance caducity time is not very high, instance information is going to be purged more frequently and the engine queries will be able to complete in a shorter timeframe, which improves the overall engine's performance.

If an archiving database is defined and configured then the instances are not only deleted from the engine's own database, but they are also moved into the archiving database.

- ***Instance caducity***

This configuration parameter determines the caducity time for an instance after the instance has reached the End activity in the process (the instance has completed the process or has been aborted by the user). This value determines when the instance should be purged from the engine's database so there is not a backlog of instances, which improves the overall engine performance. Until removed by the disposer, the instances will be kept in the engine's database.

If you have processes that generate big volumes of instances that are quickly getting to the processes **End** activity, you may want to specify a small value for this parameter (again consider instances created in split-join circuits and subflows).

- ***Disposer starting time***

This property determines when the disposer service begins acting for the first time. After this first run, the disposer service will follow the latency rules. The starting time will be valid, only if the defined value, is greater than the last time the disposer was executed.

- ***Dispose Participants***

These two properties determines whether participants that have been disabled will be removed from the organization by the engine or not. If turned on, the engine will periodically delete all participants that have been disabled.



In-flight Instances: If any participant has instances assigned to him, disabling it will immediately remove those instances from him. Unassigned instances can be seen by every participant in the corresponding role for the activity where the instances are located.

The number of days specify how long the engine will wait before deleting a participant. This parameter only takes effect if the *Dispose Participants* parameter is enabled. For example, if it is set to 30 days (the default value) the engine will delete participants that have been disabled 30 days or more.

- **Enable Archiving**

This boolean value defines whether the archiving feature will be used by the engine or not. If enabled then a configuration must be defined in the next parameter (see below). If this value is turned off then the instances will be deleted from the engine's database once they reach the end of the processes and no record of them will be kept in Fuego.

This configuration will be used for the archiving database. This parameter only takes effect if the archiving feature is enabled.

For example, if Disposer latency = 2 days and instance caducity = 10 days, the Disposer will be executed once every 2 days and will delete instances created at least 10 days before. Therefore, if today is Jan 20 and the last time the Disposer ran was Jan 18, the Disposer will run today, deleting those completed or aborted instances that were created on Jan 10 and before.

2.5.2 IPC

This section allows the configuration of the *Inter-Process Communication* service (IPC). This service allows two processes deployed in two different engines using two different directories (FDI)² to work (create instances, call subflows or send notifications between them) as if the processes were deployed within the same engine. This can be used when implementing a B2B solution or a highly distributed environment.

IPC	
Enable IPC service	<input type="checkbox"/>
IPC service Port	<input type="text" value="54350"/>
Maximum incoming connections	<input type="text" value="5"/>

Figure 14: IPC properties section

²This section does not apply to two different engines that share a common directory (FDI) even if the engines are in two physically different computers.

- **Enable IPC service**
This boolean value defines whether the IPC feature will be used by the engine or not.
- **IPC service port**
The port number to be used by the engine to handle all IPC related communication.
- **Maximum incoming connections**
The maximum number of external connections that a Fuego engine will accept at the same time. Any connection request received after this maximum number is denied. The default value is 5. Note that for each connection you specify here an execution thread will be assigned and used by it (see section 2.4.3).

2.5.3 Socket Factory

Socket Factory	
Enable socket factory service	<input checked="" type="checkbox"/>

Figure 15: Socket Factory Service section

2.5.4 SNMP

This section allows the configuration of the *SNMP service*. Simple Network Management Protocol (SNMP) is the protocol governing network management and the monitoring of network devices and their functions. Through it, Fuego allows administrators to monitor the status and performance of an engine.

SNMP	
Enable SNMP service	<input type="checkbox"/>
SNMP agent port	<input type="text" value="161"/>
SNMP manager host	<input type="text" value="PVictory-XP1"/>
SNMP manager port	<input type="text" value="162"/>
<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Reset"/>	

Figure 16: SNMP properties section

- **Enable SNMP service**

This boolean value defines whether the SNMP service will be available for the engine or not.

- **SNMP agent port**

Defines the port in which the SNMP agent will be started.

- **SNMP manager host**

Host of the remote console that will receive the SNMP information.

- **SNMP manager post**

Port of the remote console that will receive the SNMP information.

2.6 Networking

Through the *Networking* tab administrators can configure the mail server the engine will use and the Work Portal URL.

Engines > Edit Engine engine1

Basic configuration	Log	Execution	Services	Networking	Others
Properties					
Mail server name		smtp			
Administrator mail		ftadmin			
Web Work Portal URL		http://PVictory-XP1:9			
<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Reset"/>					

Figure 17: Networking properties section

- **Mail server name**

The mail server that the engine will use when sending e-mail notifications.

- **Administrator mail**

The Administrator's e-mail address. Replies to the e-mail that are sent by the engine will go to this account. This is also the account that will receive any messages from the log that are selected in the Log tab.

- **Web Work Portal URL**

URL address for logging into the Work Portal in the Web Work Portal URL field. This location is where the Work Portal application is running. The URL for the Work Portal is generally **http://host:port/WorkPortal/**. For example, when the Engine notifies users who have new instances by e-mail, this URL is part of the message. That way, they can open the application directly.

2.7 Others

The *Others* tab groups miscellaneous engine parameters and configurations.

2.7.1 Runtime

The screenshot shows a configuration window with several tabs: Basic configuration, Log, Execution, Services, Networking, and Others. The 'Others' tab is selected, and within it, the 'Runtime' section is active. This section contains a 'Runtime Database Configuration' label, a checkbox for 'Use creation configuration' (which is unchecked), and a checkbox for 'Create a new configuration' (which is checked). A link with a magnifying glass icon is next to the 'Create a new configuration' checkbox.

Figure 18: Runtime properties section

- **Runtime Database Configuration**

The configuration ID of the database that will be used at runtime. By default, this is the same ID that is defined in the basic properties of the engine (that is, the one to be used in the database management tasks). Despite this, at runtime, it is possible to configure another configuration for the engine database. This will be the case when you have deployed the engine within an application server and you wish to use the application server's resource pool to handle all the database connections. Note that the configuration combo-box will only appear if you turn off this property.

2.7.2 Directory

The screenshot shows a configuration window with the 'Directory' section selected. It contains a label 'Directory polling interval' followed by a text input field containing the number '1', and a unit selector dropdown menu currently set to 'Minutes'.

Figure 19: Directory properties section

- **Directory Polling Interval**

This parameter should be taken into consideration when working with an FDI that does not support listeners. In this case, the engine will need to connect the FDI service to check if there are updates on information stored in the FDI repository. You should always pay attention to this value unless you are working with Netscape iPlanet's Directory Service (this is the only support Directory Service by Fuego that supports listeners).

If your FDI data will change frequently, then you should define a small value based on the rate of change. If your FDI data does not change frequently, you should assign a larger value (i.e.: 60 minutes).

2.7.3 Events

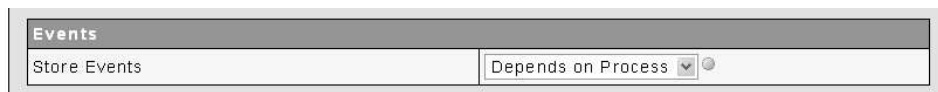


Figure 20: Events properties section

- **Store Events**

If you are interested in getting analytical or auditing information based on instance flow in business processes, you should have this option enabled. However, if you are not going to be using this feature, it is recommended that the value is set to **Never**. This will basically shorten transaction times when executing FBL tasks since the engine does not need to log event information. Note that if you disable this option, you will not be able to access the instance's audit trail. Another alternative is to use the option **Depends on Process**. In this case, only the processes that have been designed to log auditing and event information, will involve storing the event information in any engine transaction applied to a process instance.

By default, select **Depends on Process** as the value for *Store Events*. If you are not interested in auditing or archiving, then select **Never**. If you select **Always**, you should be aware of the performance issues related to storing every single action performed on a process instance.

2.7.4 PAPI

PAPI	
Instance retrieval size	1000
Notify thread priority	1 (min=1)
Latency between notifications	15 Seconds
<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Reset"/>	

Figure 21: Events properties section

- **Instance retrieval size**

This configuration parameter is used to determine the number of instances that are going to be exchanged between a client and the engine. In most situations this flow of instances will take place mainly between the engine and the Web Work Portal. The default value of 1000 can be considered optimal for a small to medium size installation with interactive processes. If the interactive processes have a high concurrency (thousands of concurrent users), then this value can be increased. However, you should try to find a trade off of value if the network connection between the client and the engine is not as optimal as desired.

The amount of data that flows between the engine and the client will be at the most:

$$\text{Instance Retrieval Size} \times \text{Max Instance Size}$$

- **Notify thread priority**

This configuration parameter defines the thread priority for the notification thread. The notification thread notifies the connected clients to update instance counts on different process activities. The JSP+Servlet engine where the Work Portal is deployed is usually a client receiving notifications. These notifications are used to update client caches. If the processes implemented are highly automated, this priority can be set lower than the *Automatic Execution Threads Priority* (see section 2.4.3).

- **Latency between notifications**

This configuration parameter is used to state the frequency of the engine updates to connected clients (i.e. the Work Portal).

3 Other Performance Considerations

The following section contains several considerations, which have a big impact on the final performance of the engine, that have to be taken into account when designing and coding a project.

3.1 User Interaction

When working with interactive processes, some statements or components invoked from interactive activity tasks, need to call or invoke components that run on the client side (typically the WorkPortal), normally because they require end user interaction. When this happens, the Engine passes the execution control to the client, and waits until that client-side piece of code finishes executing before continuing with the rest of the code associated to the Interactive activity task.

Unfortunately, this wait is not without impact. The engine keeps the execution thread for this method locked, waiting to resume execution. What if the client never executes its part, and thus never returns? (the user took a break and left the browser open in a Fuego Object presentation or input statement)

The engine uses a timeout to avoid locking the thread forever (default 5 minutes). Although this should be enough for most cases, processes with a lot of end user interaction require a more efficient way of managing execution threads. This is where *screenflows* come in.

Screenflows are specifically design to handle user interaction **without locking any resources on the engine side**. Whenever Studio detects a situation in which screenflows should be used it will display a warning message when the project is checked saying:

(21, 1) For production projects, consider using Screenflows instead of interactive methods. Screenflows scale better and are the only way to build complex interactions for J2EE Engines.

In this cases you should seriously consider why it is better to incur in a performance penalty instead of using a screenflow to handle that particular user interaction. Note that eventhough the process might be just asking for user confirmation for some action, he can still leave that question not

answered for several hours. It will not be the first time a user had a meeting or went to lunch and left the portal opened with the question pending. Multiply this situation by the number of users your process has and you can quickly see that leaving the engine without resources is not that hard if screenflows are not used.

3.2 External Components

When fine-tuning the engine, you should always check how well the components being invoked by the process scale when processing a larger volume of instances. If the underlying invoked components DO NOT scale appropriately, the engine will have a bottleneck on component invocation that will degrade the engine's overall performance.

In most of the cases, the APIs or components invoked by processes open and close connections with backend applications each time they are invoked. This model obviously will not work well since it will consume too much time opening and closing the connection with the backend application when only one transaction needs to be executed. As such, we should enforce that connection pool techniques are implemented so that the access to backend applications is faster, achieving better response times.

In any event where a performance bottleneck by an external component is suspected, Fuego provides the facility to trace the performance of every external component (in the *Execution* tab). You can turn this feature on and check whether an external component is taking too much time for the number of instances being processed or the timeout defined in the engine.

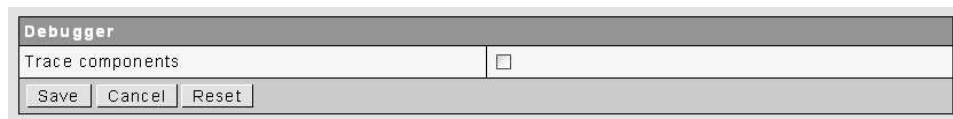


Figure 22: Debugger properties section

A Appendix: Fuego RMI

A.1 What is Fuego RMI?

Fuego RMI stands for Fuego Remote Method Invocation. Fuego RMI is the framework layer implemented by Fuego to be able to use different transport layers to enable communication between a Fuego client application and a Fuego Engine.

A.2 What are the transport layers protocols provided by Fuego?

Fuego currently provides out of the box, four transport protocols:

- **TCP:** Socket Implementation
- **MMP:** Multiplex Protocol Implementation
- **SSL:** Secure Socket Layer Implementation

A.2.1 TCP Protocol Transport Layer Implementation

This transport protocol will open a new TCP/IP socket connection between the connecting client and the engine each time the client establishes a connection. This means that the resource usage will be directly related to the number of users connecting to a Fuego Engine. If 1000 users are connected simultaneously to a Fuego Engine, 1000 TCP/IP sockets are going to be opened, used and managed by the Fuego Engine.

A.2.2 MMP Protocol Transport Layer Implementation

This transport protocol is kept for compatibility reasons, but its implementation is similar to the TCP/IP transport protocol. In every case it is recommended that you use TCP/IP instead of the deprecated MMP protocol.

A.2.3 SSL Protocol Transfer Implementation

This protocol work in a similar way as the TCP/IP protocol transport layer with the added benefit that each connection uses the public-and-private

key encryption system in order to encrypt the information being passed with it.