**Threads**

**Threading Concepts**

1. Request and response pipelines always execute in separate threads. While the request thread originates from the proxy service transport, the response thread originates from the business service transport.
2. When external services are invoked, threads can be blocking or non-blocking, depending on the pipeline action, the Quality of Service (QoS) configuration, and the transport being used.
3. When using blocking calls, a Work Manager with a minimum thread constraint must be associated with the response in order to prevent server deadlocks.

OSB Pipeline actions that specifically affect threading: route actions, publish actions, and service callout actions:

1. **Route Action**

By default, the HTTP transport uses asynchronous features of WebLogic Server to prevent thread blocking while waiting for a business service response. For the execution of a route action, once the thread finishes sending the request, it returns to the pool where it is then used to process other work. When a response is returned from the external service, a second thread is scheduled to process it. This behavior can be modified by using a route options action and setting the QoS to Exactly Once

1. **Publish Action**

A publish action is a one-way send. It provides the means of invoking an external service but without receiving a response. This is often used to provide notification of an event, such as for logging or auditing. By default, no feedback of whether the call was successful or not is returned to the pipeline thread. For both of the above actions, setting the QoS to Exactly Once forces the request thread to block until a response is received. This allows the request thread to notify the caller of an error immediately, without callback. This behavior is also useful when attempting to throttle the number of threads simultaneously processing a proxy service

1. **Service Callout Action**

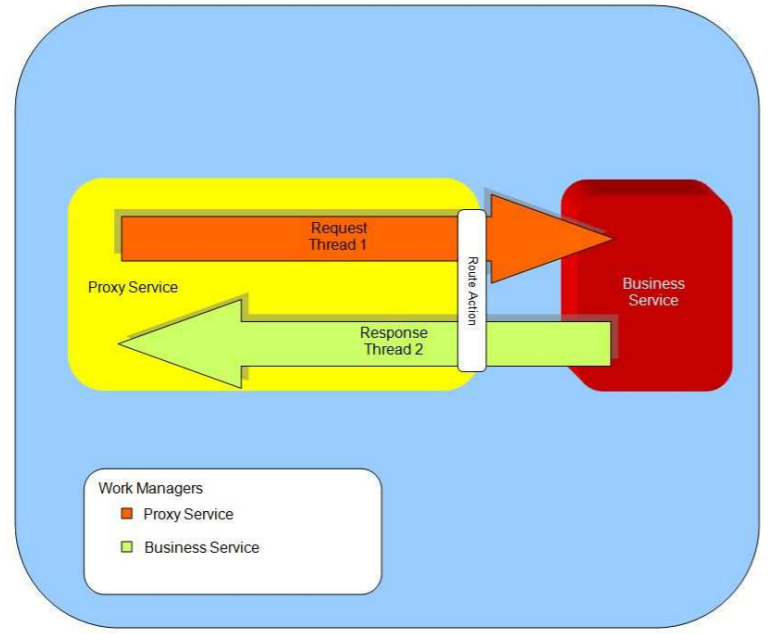
A service callout is implemented as a synchronous blocking call. Its design intention is to provide the ability to invoke an external service to enrich a request message prior to routing the request to the target service. While the callout is awaiting a response, the request pipeline thread blocks until a response thread notifies it that the response is ready and processing can continue.

**OSB Pipeline Threading Model:**

* At least two threads are required to process a proxy request

1. A minimum of 1 for the request pipeline
2. A minimum of 1 for the response pipeline

* The number of threads will vary based on the actions in the pipeline

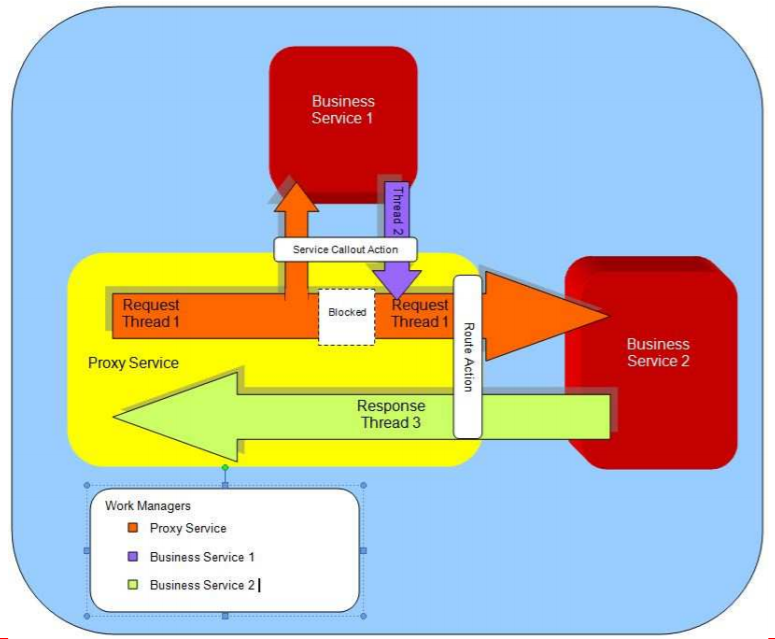


**Threading Behavior Modifiers:**

* Service invocation method
* Quality of Service setting
* Pipeline transactional settings
* Transport implementation
* Business Service retries

1. **Service Invocation Methods**

* Route
* Most commonly used
* Optimized to be the most efficient
* Can be request only, or request-response
* Non-blocking in most cases, depends on transport
* Publish
* One way messaging
* Often used to implement custom auditing/logging
* Generally non-blocking, but can block in certain conditions
* The service is one-way
* There are retries configured for the service
* WSCallout (Service Callout)
* Intended for message enrichment
* ALWAYS a blocking call
* Outbound calls still use asynchronous request response pattern
* Requires an extra thread to receive the response and notify the blocked thread
* Potential server hangs (e.g. if all active threads are used for service callout and blocked waiting for response)



1. **Business Service Retries**

* The retry interval causes a delay once the set of endpoints has been exhausted (not between each retry)
* The retry interval is implemented as a Thread.sleep(nn);
* The thread is not released back to the pool
* Long retry intervals can result in STUCK threads

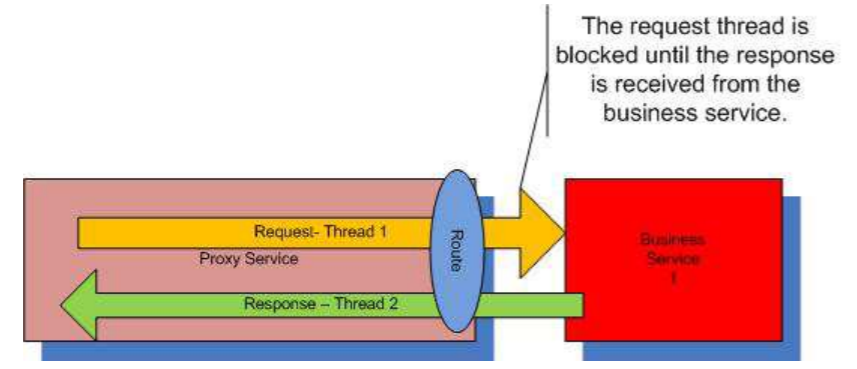
1. **Transport Specifics**

Each transport can vary in its utilization of threads

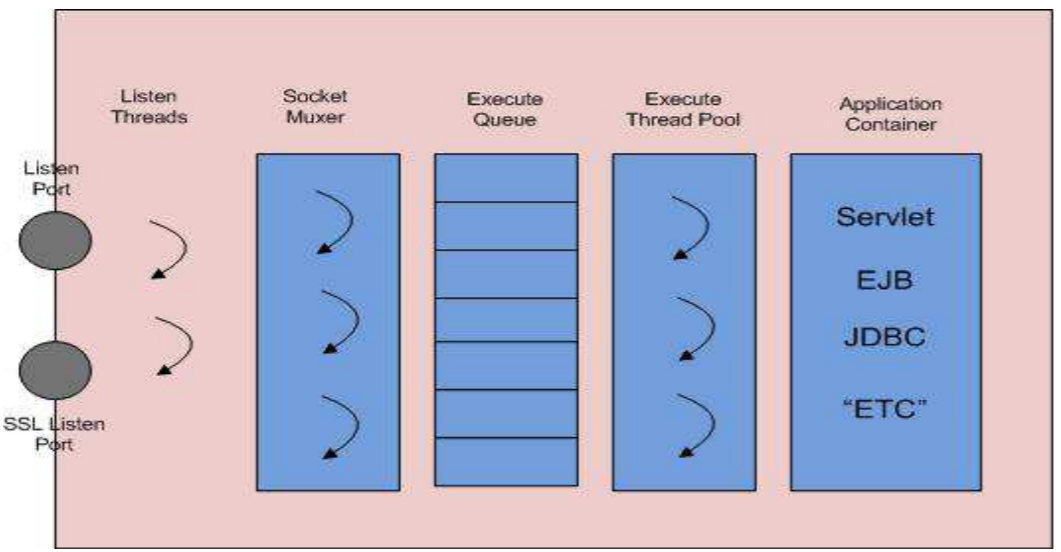
* HTTP – Exactly-once forces request thread to block
* JMS – Number of destination consumers defaults to 16
* Local – Executed in same thread as calling proxy
* MQ – Polling and Worker threads
* JEJB – Synchronous, forces request thread to block until response received

1. **Effect of Quality of Service**

* The Quality of Service specified for a service invocation can modify the threading behaviour.
* For HTTP, Exactly-Once forces the request thread to wait for a response
  + Allows errors to be handled by the request thread



**Threads and Work Managers Server Architecture:**



1. **Listen Ports and Listen Threads**

* Waits for connection requests, accepts the request, hands the socket off to the muxer, and goes back to wait for the next request
* By default, WLS listens on two ports
* Listen Port – All non-SSL based protocols (T3, HTTP, IIOP)
* SSL Listen Port – All SSL-based protocols (T3S, HTTPS, IIOPS)
* May also configure additional network channels to allow additional ports to be defined and limit the protocols accepted

1. **Socket Muxer Overview**

* Processes incoming requests on established connections
* Handles requests for any supported protocol
* Inspects first few bytes of input stream to:
  + Determine the protocol
  + Determine the request target
  + Determines the work manager
* Packages the request up and puts it on the execute queue

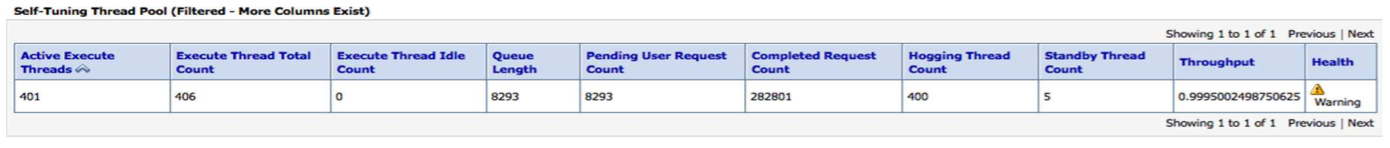
1. **Self Tuning Thread Pool**

* Single user thread pool that automatically grows and shrinks
* 400 max for “active” state
* Can be tuned in some cases
* specify the min thread pool size in –D…
* Don’t change max

1. **Thread State**

* Active (ready to accept or executing requests)
* Idle
* Hogging
* Stuck
* Standby (not ready to accept requests but can be promoted if needed but also able to handle request in some situations)

**Console-> Server: Monitoring -> Threads**



**OSB – Work Manager**

**Work Manager Fundamentals:**

* A WorkManager’s purpose is to define a set of execution characteristics
* These characteristics provide hints to the WorkScheduler as to how requests should be serviced by the Self Tuning thread pool
* A work manager definition consists of a request class and optional constraints

**Constraints**

Constraints allow you to set limits on how a work manager’s requests are processed.

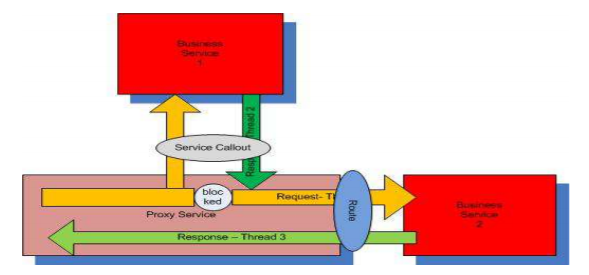
* Maximum Threads Constraint – limit concurrent requests
* Minimum Threads Constraint – ensure threads are always available
* Request Class Constraint
  + Fair Share – relative value of thread usage time (default 50)
  + Response Time – response time goal (ratio, not absolute value)
  + Context – users or groups context of the request
* Capacity Constraint – limiting requests in queue

**Work Managers in Service Bus**

* Configured on service transport tab as “Dispatch Policy”
* Can be configured on both proxy and business services
* Control how many MDB (JMS Proxy) instances (max threads constraint)
* Constraints can be shared by multiple work managers
* The constraint is applied to all work managers, not per work manager
* For example
  + A Max Thread Constraint of 10 shared by 2 work managers
  + The maximum total number of threads processing for both workmanagers at any given time is 10, not 20.

**Work Manager Best Practices**

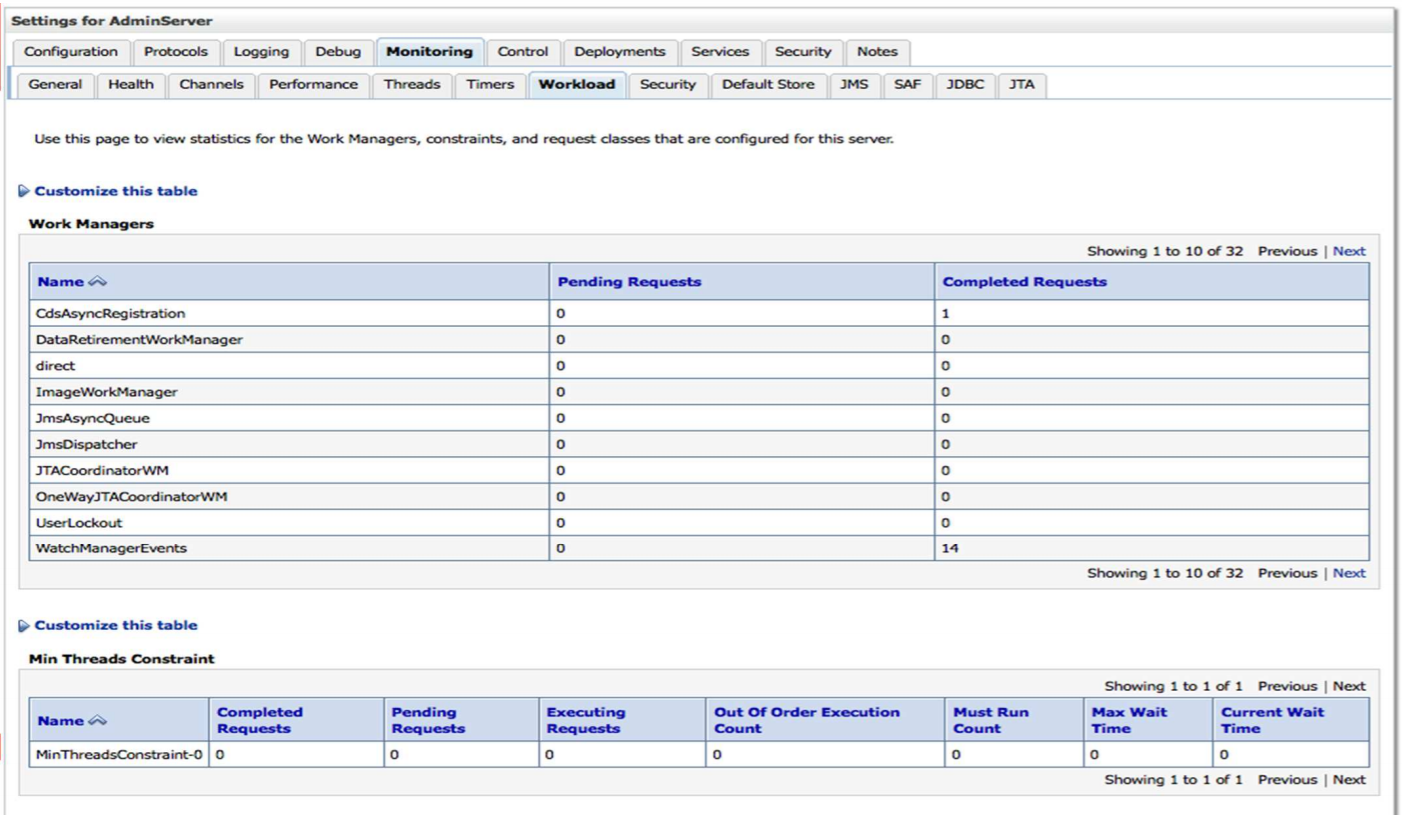
* Assign a minimum constraint work manager to services invoked by a service callout (a value of 1 or 2 is typically sufficient)
* Starting in 11.1.1.7
  + Default Response Work Manage created by default
  + Automatically assigned to all Business Services
  + Doesn’t avoid issues with WSCallout in Response pipeline



* Advisable not to use the same work manager for both the proxy service and business service
* A work manager assigned to a business service comes into play on the response pipeline
* Work managers are per managed server
* Start simple with an objective in mind, get more complicated as required

**Work Manager Monitoring**

**Console->-Server->Monitoring->Workload**



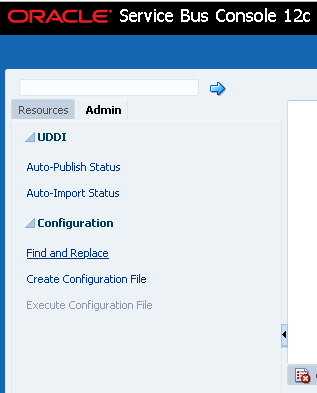
Oracle WebLogic Server prioritizes work and allocates threads based on an execution model for deployed applications, components, Web Services, as well as service-level agreement with a feature named Work Managers. Following are Steps for Creating Work Manager for OSB Service.

**Steps for Creating Work Manager for OSB Service:**

**Step 1:**

At first, you have to disable the proxy service that you’ll be creating Work Manager for, on Service Bus. For this

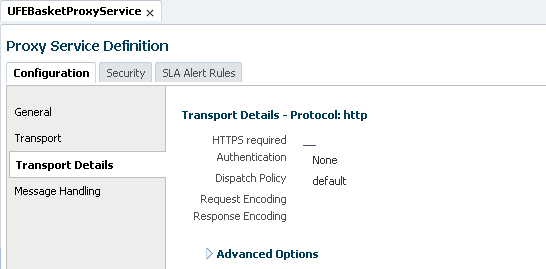
1. Go to Service Bus Console,
2. Open “System Administration” drop menu,
3. Go to “Find&Replace” on the “Customization” part,
4. Write the queue name on “Find Value” area,
5. Click “Find” button.



**Step 2:**

On the list, you’ll see the proxy service of the related queue.

1. Go to the proxy service name.
2. Open “Operational Settings” tab,

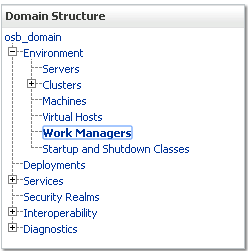


1. Click “Create” button to open a new session,
2. Uncheck the check box of “Enabled” on “State” header,
3. Click “Update” button at the bottom of the page,
4. Click “Activate” button to activate the changes,
5. Finally after clicking on activate button, on the next screen, you define the process done and click “Submit” button.

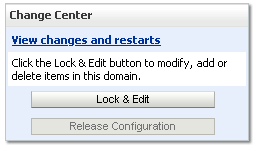
**Step 3:**

Next step is to create Maximum Threads Constraint and Work Manager for OSB Service on WebLogic Server. To do these,

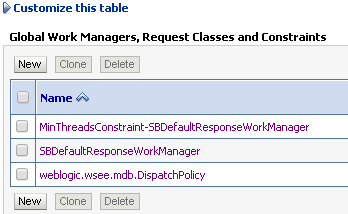
1. Select the “Work Managers” section from under the “Environments” in the WebLogic Console,



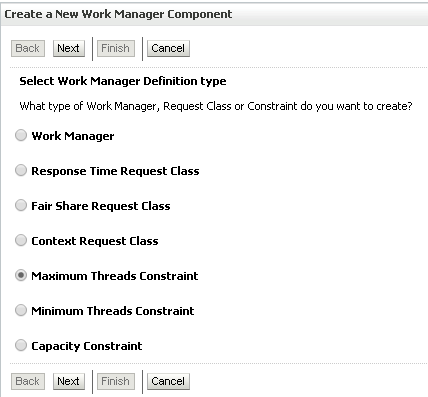
1. Click “Lock&Edit” button



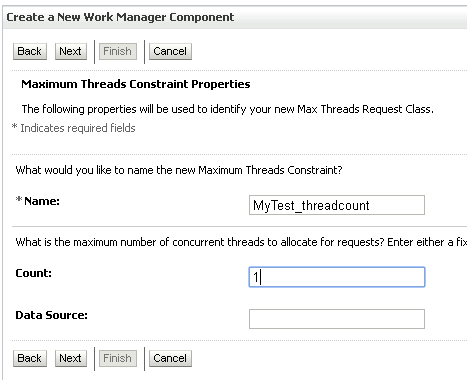
1. Click “New” button to create work manager



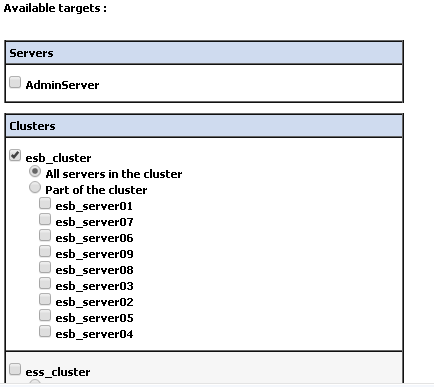
1. Select “Maximum Threads Constraint”



1. Click “Next” button
2. Enter the name ‘My\_MaxThreadsConstraint’ or something to that effect, (This is important as it needs to match that of the dispatch policy in the OSB proxy.)
3. The thread count should be 1 (lowest acceptable) to start with and can be increase during load testing until we find the optimal setting. (value of -1 is unbounded),



1. This Work Manager needs to be targeted to the OSB cluster, so check the checkbox of “osb\_cluster”,

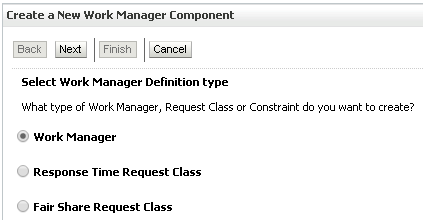


1. Click “Finish” button,
2. And click “Release Configuration” button. This should not require a restart but may do if modify a work manager that is in use.

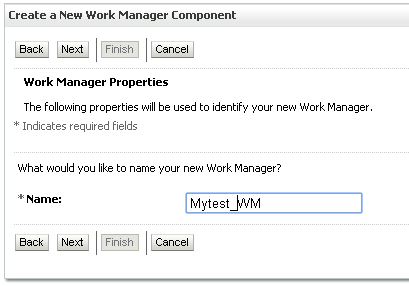
**Step 4:**

After seeing the Maximum Threads Constraint created at the beginning list, now it is Work Manager’s turn to be created:

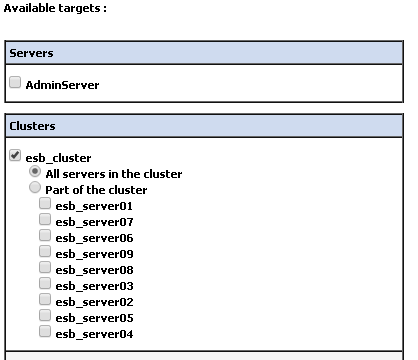
1. Again, select Work Managers part, and Click “New” button
2. This time select “Work Manager” radio button to create work manager



1. Click “Next” button
2. Write the WM Name down on the name field,



1. Click “Next” button,
2. This Work Manager needs to be targeted to the OSB cluster, so check the checkbox of “osb\_cluster”,

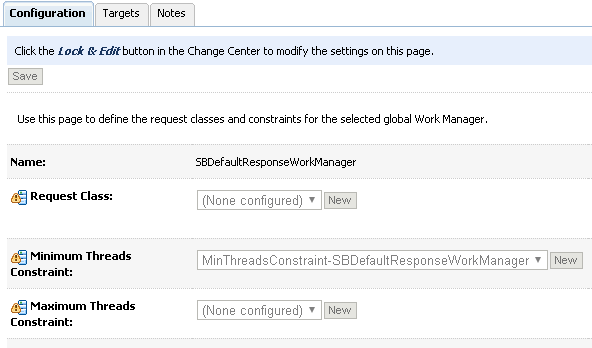


1. Click “Finish” button,

**Step 5:**

Now it is important that you have to go and click to the Work Manager you created in the list, so that you will link your maximum threads constraint to your work manager. To do that:

1. Again, select the “Work Managers” section from under the “Environments” in the WebLogic Console
2. Select the Work Manager you created from the list
3. On the “Configuration” tab, you will see a combo box called “Maximum Threads Constraint”. From that combo box, select your maximum threads constraint name created,



1. Click “Save” button,
2. And click “Release Configuration” button to activate your changes

**Step 6:**

Now, you should return to Service Bus Console, and find the proxy service you disabled before (Follow the step 1 for it).

1. Go to proxy service
2. Click “Create” button to create new session
3. Click “Configuration Details” tab
4. See “JMS Transport Configuration” header, and click on “the icon” (right side of the header)
5. Select the work manager you created from combo box of “Dispatch Policy”
6. Click “Last” button, and click “Save” button.
7. Now, time to enable the proxy service, so go to “Operational Settings” tab,
8. Check “Enabled” check box from “State” part,
9. Click “Update” button at the bottom of the page,
10. Click “Activate” button to activate changes, and write down your changes and click “Submit” button.

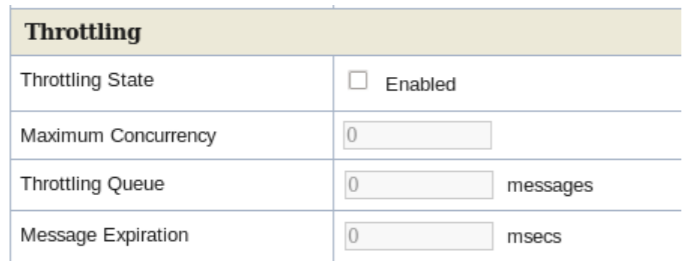
**Throttling:**

**Overview**

* Provides a means of protecting back end systems from overload
* Allows the user to define the maximum concurrent calls
* Once Maximum Concurrency has been reached, additional requests must wait
* Applies only to Business Services
* Throttling Group added in 12c

**Throttling - Configuration**

* Can be enabled/disabled per business service
* Maximum concurrency is per cluster, not per managed server
* Throttling Queue – How many messages can wait
* Additional messages are rejected immediately
* Message Expiration – How long will a given message wait in queue



**Throttling – Blocking vs Non-Blocking**

* When a request is throttled it may or may not block a thread while waiting
* Throttling will inherit the blocking characteristics from the transport and invocation method
* If the call would have blocked, the throttling will block
* One exception, JEJB proxy services.