# Overview

This document describes the operation of ActiveNet’s “multi-tenant servlet” system. This allows a single servlet to transparently serve requests from multiple orgs.

This functionality was built into Activenet in 2012, but was never quite finished, and was never deployed in multi-tenant mode. As a result, it is necessary to redocument its operation, and make some changes to the request processing to make it fit better with the new Jetty implementation.

Moreover, although the servlet was made fully multi-tenant aware at that time, because servlets have never received requests from more than one org, bugs have be introduced, in which the needed coding standards for multi-tenant operation were not followed.

# Single tenant model



1. The F5 routes requests for a given site (e.g., “orgsite1”) to one of the assigned servers, and a specific port.

2. On that server, there is one Jetty instance per orgsite, each listening on a specific port (3000+orgsites.servlet\_number). Each instance is started by a start\_service.sh which is customized for that orgsite by the deployer.

3. The context.xml picks up a substitution parameter from the Java command line, to only accept requests for that /orgsite.

4. Our code is configured by two files, sdi.ini and service.properties, which are customized per orgsite by the deployer.

## Multi-tenant benefits

Currently, each server has ~40 orgsites, so 40 servlets, each running in its own JVM/Jetty instance. In a multi-tenant implementation, all 40 orgsites would run in a single JVM/Jetty, giving these benefits

* **Simplified deployment and maintenance:** Only one Jetty instance to configure and start/stop.
* **Reduced memory overhead:** Each of the ~40 servlets on a server has its own JVM and its own set of classes (metaspace). This duplicated overhead is ~1/3 of the server memory. On a 24GB server, we’ll save ~8GB of memory by having one JVM.
* **Greatly increased heapspace headroom for “bad” requests:** Currently each servlet must have a separate configured heapspace. Let’s say there is a 500MB heap and the cached startup data uses 200MB. Then 300MB is available for transient objects created by a request. With 20GB of heap available for the 40 sites, and 8GB of cached data, there is 12GB of headroom.

## Single-tenant benefits

* **Process isolation:** Each servlet is a separate process, so if it badly misbehaves, it can be restarted without effecting other sites.
* **Version independence:** Each servlet can run a different version, making it easy to deploy UAT or hotfix to any group of sites. With MT, all sites on a server will be running the same version, so app support will need to dedicate

# MT Reference integration model



## 1. Requests use existing URL structure

The original multi-tenant servlet implementation required different URL structures for multi-tenant servlets. I believe this was a result of limitations to the ServletExec mapping. This would require changes to web service clients, such as CUI, as well as breaking external links and bookmarks. This has now been changed to use the same URL structure as existing single-tenant servlets:

* /<orgname>/servlet/\*.sdi for standard servlet methods; e.g., /ymcala/servlet/adminlogin.sdi
* /<orgname>/<wsname> for web service requests’ e.g., /ymcala/ActiveNetWS

## 2. The F5 forwards all requests on a server to a single port.

The F5 receives a requests and dispatches it to a server in the appropriate web server pool (data group). This operation is driven by an org mapping file, generated during deployment from the org configuration data in the ActiveNetSites DB. This file will require a port number for multi-tenant:

* For old-style, ServletExec implementation, the file only needed to map an org to an F5 datagroup.
* For Jetty, the file needs to map an org to a data group and a port
* For multi-tenant Jetty, all orgs in the pool have the same port number, so all requests go to the single servlet.

## 3. Only one servlet is deployed on each server

For the single-tenant Jetty installation, there is a servlet per orgsite, with a context path of the orgsite name. Each servlet is listening on a unique port.

For multi-tenant, there is a single servlet, listening on a single port. The context path must be empty (“/”).

## 4. Jetty processes the request based on web.xml

Jetty receives the request, and based on entries in the Web.xml, first sends the request to one or more Filters, then to a servlet.

## 5. OrgContextFilter creates OrgContext for request

Based on the web.xml, all requests for org-specific URLs go to the OrgContextFilter. This identifies the org from the URL (e.g., "ymcala"), and finds or creates an OrgContext for that org.

## 6. All org-specific data is referenced via the OrgContext

The OrgContext is available to the rest of the servlet code via a ThreadLocal. All org-specific "static" data is referenced via the OrgContext. For example, the cached activities HashTable, rather than being a static of the Activity class, is referenced via the getActivityList() method of the OrgContext.

## 7. .ini file is parameterized

During initialization of an OrgContext, the servlet reads its configuration file (.ini) to get key information such as the JDBC URL. The .ini file, this information is parameterized, and when the servlet reads it, it makes substitutions into it based on the OrgContext. For example, in the JDBC URL, it substitutes %org\_site\_name% with the actual orgsite name (which is the database name).

## 8. Threads are started for each OrgContext

During OrgContext init, an instance of each background thread is created for the org, which remembers its OrgContext.

# Areas of environment modification

In order to implement the above model, the following changes need to be made to the deployment / environment:

* **Servlets:** If a pool is flagged multi-tenant, create a single Jetty instance on that server.
* **Org mapping file:** In the org mapping file generation, route requests for all orgs in a MT pool to the same port (presumably a standardized port used by all multi-tenant servlets.
* **.ini file:** When the .ini file is generated by deployment, some entries which should be written with parameters instead of exact values, such as using %org\_site\_name% instead of the org name for the database name and URLs.
* **Context.xml and web.xml:** These are generic, with no substitution required.

# Required coding standards

* All org-level data, such as cached data and global configuration, must be accessed via the OrgContext class, and not implemented as static fields.
* All threads should be implemented using the com.activenet.thread classes to maintain the OrgContext.

# Coding in MT branch

Summary of changes made in the ActivenetServlet project in a special MT branch:

* OrgContextFilter and OrgContext classes, and web.xml reworked to work with “standard” single-tenant URL patterns for both normal servlet methods and web service methods.
* An annotation @TrulyStatic was written to allow (force) developers to explain any static fields.
* “Multi-tenant test” function was added to test the application for improperly written statics and thread detect statics.
* Modifications made to the NetbeansUtility used to automatically convert statics to OrgContext getters and setters.

# Project tasks

Sketch of key Dev, Test and IT tasks to move MT into production, somewhat in order:

**Bring new MT code into trunk and confirm ST operation**

* Merge MT branch code into trunk.
* Smoke test new code in ST operation in Int.

**Make known changes required to complete proper MT operation**

* Static rework:
  + Use “Multi-tenant test” to identify statics which need to be reviewed, then convert them to finals or annotate with @TrulyStatic as appropriate.
  + When only org-specific statics are left in the Multi-tenant test results, use NetbeansUtility to convert remaining statics or OrgContext members.
  + Review resulting code changes and retest with Multi-tenant test.
  + Add unit test which fails if any unannotated statics are introduced.
* Threads rework:
  + Convert all identified threads which don’t use BackgroundThread to use it.
  + Eliminate the default OrgContext, so threads which don’t properly get an OrgContext will fail.
* Smoke test new code in Int with two orgs.

**Deployment changes and testing in QA/Pref**

* Modifications to deployment for MT pools:
  + Single Jetty instance
  + F5 table generation to single port
  + New web.xml format (same for both ST and MT)
  + New activenet.ini format (same for both ST and MT)
* Smoke test MT pool of test sites in QA
* Preliminary load testing in Perf:
  + Deploy MT wide pool of multiple identical orgs
  + ST load testing comparison against final results from previous release.
  + ST load testing comparison against testing with a mix of requests to N orgs.
  + Comparative load testing of requests to N orgs, comparing results with N ST servlets on the same pool to 1 MT servlet.
* Functional testing in QA:
  + Test all threads for proper operation with 2 orgs, with different data and expected outcomes.
  + Regression to include two organization concurrent testing of the same features on at least selected areas of functionality.

**Rollout**

* Phased migrations of groups of organizations to MT pools.
* Monitoring of error rates, performance.

# Areas still requiring development / testing:

## Statics

* All statics need to be reviewed to see if there are any which need to be “hoisted” to the OrgContext. Based on using the automation we previously used to move statics to the OrgContext, there are ~100 statics which need to be reviewed. About half

## Threads

* It appears some of the threads don't currently work without a "default context", which is currently set as the first context created. The default context should be reviewed; I believe OrgContext should not have one, because this prevents correct multi-tenant operation.

## Other

* WS calls are properly handled based on SoapUI testing, but Flex resource scheduler is generating an invalid incoming WS URL (/ActiveNetWS instead of /org/ActiveNetWS). The assumption is that this is Flex code formatting the URL in a way which does work with single-tenant, but doesn’t provide the CUI and Flex UI need to be tested.
* SSO servlet: I see in the web.xml that \*.sso requests are sent to a different servlet; this must be tested for multi-tenant operation.