

Oracle In-Focus

Advanced WebLogic Server Automation

Administration and Monitoring with WLST and JMX

Martin Heinzl



RAMPANT

Donald K. Burleson
Series Editor

Advanced WebLogic Server Automation
Administration and Monitoring with WLST and JMX

Oracle In-Focus Series

Martin Heinzl



RAMPANT
TECHPRESS

Creating Domains with WLST

In the first part of the book, I introduced the WebLogic domain as the main administration concept of WebLogic; that is, everything in WebLogic is structured within a domain. Therefore, this first part of the WLST discussion is focused on the domain. We will start with the minimal domain and extend it with more components until we will finally discuss, in the last section of this chapter, a comprehensive but flexible example of how to create a domain with a number of important features.

Introduction to Domains

A domain is the basic administration unit for WebLogic. Each domain has at least one WebLogic instance but can coordinate with more. All resources like JDBC (database connectivity), JMS (messaging), WTC (Tuxedo connectivity), virtual hosts and more are managed together as a single unit.

Besides a dedicated server instance for administrative purposes, a WebLogic domain consists of additional WebLogic Server instances called Managed-Servers. These server instances can be grouped together to form clusters. A domain may have as many clusters as needed. A WebLogic cluster is going to provide scalability and high availability for applications. Clusters can improve performance and provide failover if any of the server instances crash or become unavailable. The servers within a cluster can run on the same machine, known as a vertical cluster, or they can reside on different machines, known as a horizontal cluster. To the end user/client, a cluster appears as a single Oracle WebLogic Server instance.

One of the main responsibilities of the AdminServer is to provide the configuration to all Managed-Server. All configurations will be done on the AdminServer and from the AdminServer distributed to the different Managed-Server(s). Distribution either happens in real-time or at boot time of the Managed-Server, depending on the nature of the configuration change. Besides the static responsibilities, the AdminServer also has dynamic (runtime) responsibilities. One of the runtime responsibilities is to monitor certain parameters like the health of the Managed-Servers or certain central services like JTA. If a domain contains Managed-Servers, then the Managed-Servers

are responsible to perform the business logic. The business logic is implemented in application units deployed to these Managed-Servers. Managed-Servers depend on the AdminServer to get the configuration, but Managed-Servers can operate independently of the Administration Server. Managed-Servers will send notifications to the AdminServer in case of state changes so that the AdminServer always has the state of the Managed-Servers.

In the offline mode of WLST, we can configure a simple/customized WebLogic domain. There are two different options for development environment configurations. One is the configuration with a standalone AdminServer in the domain, and the other option is a clustered domain with an AdminServer for administrative purposes and Managed-Servers for application hosting. The second option is the preferred option for all environments, and is also helpful for migration processes. On less powerful machines like personal development machines, the first option becomes handy for development.

Domain Templates

Domain creation is based on templates. Oracle provides a set of templates that are ready to use for constructing domains. Templates are essentially JAR archives with a number of XML files, property files, and other artifacts (e.g. libs you want to automatically distribute). The provided templates are located in the folder `WLS_HOME\common\templates\domains` for domains and `WLS_HOME\common\templates\applications` for extensions. The file `wls.jar`, which is the basic template provided by Oracle, is located under `common\templates` and will be read while creating a domain using WLST. The new domain will be written using the *writeDomain* command.

Oracle provides a list of tools for creating domains and templates:

- The Configuration Wizard
- The Domain Template Builder
- WLST Offline
- Pack and unpack commands

The *pack* and *unpack* commands provide a simple method for creating domains from the shell; however, they do not enable you to change or extend the contents of your domain. WLST also offers the command *configToScript*, which creates a script based on an existing domain. I recommend being careful with this command as it might not include everything. This command helps to script a new domain based on an existing one, but should only be considered as a starting point for further script development.

Note that starting with WebLogic 12.1.2, the location of the templates has changed.

Template Categories

Templates can be divided into 2 major categories: domain templates and extension templates.

Domain template: This template creates a full domain with potentially all infrastructure components and services. The most used template - `wls.jar`, which will construct a basic domain - belongs to this category.

Extension template: This template does not define a full domain. Instead, you can define additional applications and/or services you want to add to this domain. This template can only be applied to an already loaded domain template (or configuration). `wls_webservice.jar` is such an example that adds functionality for advanced Web services, including WSRM, Buffering, and JMS Transport.

You can extend the provided templates with the template builder. Based on real project experiences, I really recommend avoiding this. Whenever possible, try to move all modifications/extensions you want for your domain into an appropriate WLST script. The reason for this involves updates and patches. If you update or patch your WLS installation folder, then you might update your templates if, for example, a template has changed due to a bug correction. In this case you need to re-do all your modifications if changed the template and left it in the default location OR your domain will be based on a potentially buggy template if you saved the modified copy elsewhere. Also, moving to another server may lead to a new installation where you forgot that you changed the default templates. If your modifications are in a WLST script, then they have a much better visibility.

Figure 5.1 shows the content of the basic domain template (`wls.jar`):

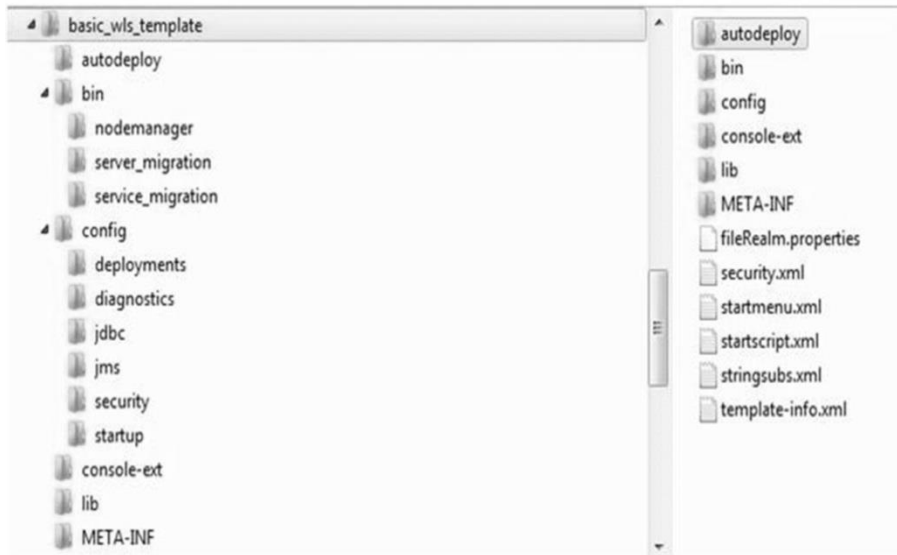


Figure 5.1: *A typical domain template structure*

Templates have to be distinguished between domain templates and extension templates. Domain templates describe the creation of a full domain whereas extension templates add features to a domain. Every template is packed into a JAR file.

Domain templates can be used on their own. In order to use an extension template, you have to either load an existing domain or a domain template first.

The above template consists of a few main XML files that describe the main domain layout and subfolders for resources and extension definitions. The main reason for this is that you can load a template, change and extend it (including libs and deployments), and save it as your own custom template.

Custom templates have great potential to make your scripts and setups easier but - as stated above - there are risks when you update your WebLogic version.

It is possible to create your own templates. Please refer to the links in the reference section at the end of the book for further readings as this topic is out of scope for this book.

Problems and Alternatives to Custom Templates

Using extension templates is basically painless for most environments. But if you want to create your domain with specific settings or additional files, you have different options.

The first option would be to copy the default template (in most cases called `wls.jar` in the WebLogic template folder) and change the content of the XML description files or add files. This is an easy task and also pretty effective. But this also has some hidden disadvantages. The main disadvantage I have faced while using this approach was that whenever you want to upgrade to another WebLogic version, you **MUST NOT** forget that you are using a custom domain template based on a specific WebLogic version, even if you have copied and modified it. You also need to re-create another custom template with the same changes, but based on the default WebLogic template from that new version.

This gets even worse if you are dealing with different WebLogic versions in your environment at the same time, which is actually very often the case.

WLST can of course work with templates (see next sections). It is possible to read templates in WLST using the *readtemplate* method. Therefore, a very elegant alternative to the problem mentioned above would be to always use the default template provided by Oracle but change it on the fly after it was opened by WLST. Unfortunately, I have not found a way either documented or undocumented to modify the template files "on-the-fly" without modifying them on disk. As all are XML files, I had expected to get a document object back after reading the template and then get a chance to modify the content using something such as XPath technologies.

Another alternative, which I actually prefer as it eliminates the above problem, is to use the default template, create the basic domain, and after the files have been created use file operation technologies to change the files written to the domain. As WLST domain (=offline) operations have to happen on the box where the domain is created anyway, network is not an issue. This approach is very useful as it eliminates the need to worry about version-dependent custom templates. It has also another benefit as all changes you make are visible in a script and therefore can be documented. If you only modify template files, it is very hard without additional documentation to find out what exactly you have changed and why.

WLST also offers the Jython file system - as it is based on Jython - and OS access functionalities that can be used to modify files. The following little WLST script will

show one example. I expect that all UNIX admins might be a bit upset but this is only an example, but I have actually used this in production quite well.

The following WLST script example could be used after domain creation but still as part of the WLST script in order to exchange Xms, Xmx, PermSize and MaxPermSize settings for the AdminServer. For the Managed-Server, this can be configured in the server start section of the domain if a NodeManager is used (recommended). Note that this script will demonstrate how to do system calls. In this case, this script will use a series of *sed* calls in order to modify the setDomainEnv.sh file that was created in the bin directory.

```
< ... readtemplate, do settings, ... writeDomain, ... >
domainLocationBinDir = '/wls_domains/testdomain/bin/'

os.system('sed \\'s/\\-Xms256m\\ \\-Xmx512m/\\-Xms1024m\\ \\-Xmx1024m/\\' '+
domainLocationBinDir+'setDomainEnv.sh > '+domainLocationBinDir+'tmp1.sh')
os.system('sed \\'s/\\-Xms512m\\ \\-Xmx512m/\\-Xms1024m\\ \\-Xmx1024m/\\' '+
domainLocationBinDir+'tmp1.sh > '+ domainLocationBinDir+'tmp2.sh')
os.system('sed \\'s/\\-XX:PermSize=128m/\\-XX:PermSize=196m/\\' '+domainLocationBinDir+'tmp2.sh > '
+domainLocationBinDir+'tmp3.sh')
os.system('sed \\'s/\\-XX:PermSize=48m/\\-XX:PermSize=196m/\\' '+domainLocationBinDir+'tmp3.sh > '
+domainLocationBinDir+'tmp4.sh')
os.system('sed \\'s/\\-XX:MaxPermSize=128m/\\-XX:PermSize=256m/\\' '+domainLocationBinDir+'tmp4.sh > '
+domainLocationBinDir+'tmp5.sh')
os.system('cp '+domainLocationBinDir+'tmp5.sh '+domainLocationBinDir+'setDomainEnv.sh')
os.system('chmod +x '+domainLocationBinDir+'setDomainEnv.sh')
os.system('rm '+domainLocationBinDir+'tmp*.sh')
```

(Note that the line breaks must be removed in order to run this script)

The same idea of using system calls after the domain creation can be used in order to extend the domain with custom files (like additional MBeans, libs or configuration files). Instead of extending the default WLS template, it is also possible to just create the base domain and then use an `os.system('tar ...')` tar call in order to extract a prepared tar file on top of the newly created domain.

This approach has several benefits:

- a) It is independent from the WLS template, and
- b) The same command can be integrated in the enroll script on the remote machines so that these required files can also be added to the remote Managed-Servers after the domain was enrolled.

Creation of a Simple Domain

The following example creates a simple test domain with just one server (the Administration Server).

create_simple_domain

```
#####  
# Create a very simple domain with only one admin server !!  
#####  
  
print 'Creating the domain...'  
domainsDirectory='/mydomains';  
domainName = 'TestDomain';  
readTemplate('/opt/wls/wlserver_10.3/common/templates/domains/wls.jar');  
  
# Setting listen address/port  
cd('/Server/AdminServer')  
set('ListenAddress', 'localhost');  
set('ListenPort', 7001);  
  
# SSL Settings  
create('AdminServer', 'SSL')  
cd('SSL/AdminServer')  
set('Enabled', 'true');  
set('ListenPort', 47002);  
  
# Setting the username/password  
cd('/Security/base_domain/User/weblogic');  
cmo.setName('weblogic');  
cmo.setPassword('test1234');  
  
# Set some important domain options  
setOption('CreateStartMenu', 'false');  
setOption('ServerStartMode', 'prod');  
setOption('JavaHome', '/usr/lib/jvm/jre-1.6.0-openjdk.x86_64');  
  
setOption('OverwriteDomain', 'false');  
  
print 'Writing Domain: '+ domainsDirectory+'/'+domainName;  
writeDomain( domainsDirectory+'/'+domainName);  
closeTemplate();  
print 'Domain Created';  
  
connUri = 't3://localhost:7001';  
print 'Starting the Admin Server to test system ...';  
startServer('AdminServer', domainName , connUri, 'weblogic', 'test1234',  
domainsDirectory+'/'+domainName, 'true', 60000, 'false');  
print 'Started the Admin Server';  
  
# Connecting to the Admin Server  
connect('weblogic', 'test1234', connUri);  
print 'Connected';  
  
print 'Shutting down the Admin Server...';  
shutdown();  
print 'Exiting...';  
exit();
```

Now let us explore the different sections of this script in more detail. Note that the complete script is based on offline WLST, which means that we are basically creating and working on the config.xml file (and others as needed).

First we will read the basic template using *readTemplate(...)* for a domain with the name wls.jar, which is provided with the standard WebLogic installation. This will read in the structure and the necessary MBean definitions into memory and thereby make the MBean structure available for the next steps.

This command will open an existing domain template file, which will then be the basis for creating a new domain. When you navigate in the domain template, you are placed into the configuration bean hierarchy for that domain template, and the prompt is updated to reflect the current location in the configuration hierarchy. WLST traverses

the hierarchical structure of the configuration beans using commands such as the *cd* command in a similar way that you would navigate a file system in a UNIX or Windows command shell.

Security settings are always special configuration items. Certain security settings are only available for WLST during domain creation and not while updating a domain. During the design of WLST scripts, which also configures security aspects, it is important to take this into account. Also, certain security aspects are only configurable using the offline mode.

Therefore, in the script section, we can now access the AdminServer MBean and change the values as we require (like port, listener, and SSL settings).

It is very important to define the user and password for the administrative access. Note that this does not create the credentials for the server to log in; it creates the user record in the internal LDAP so that the Administration Server and also Managed-Server can authenticate themselves.

Finally, we set some domain definitions. *ServerStartMode* is an important setting that should always be set to "prod" if the domain is not used for development. Personally I also set my development domains to prod so that I cannot, just by chance, overwrite values.

The final script part starts the Administration Server from the WLST script in order to test if the domain was written correctly and that the server can start.

If you now run this script it will create your domain, start the AdminServer, test the connection, and finally stop the AdminServer again.

start/test_adminserver

```
Initializing WebLogic Scripting Tool (WLST) ...

Welcome to WebLogic Server Administration Scripting Shell

Type help() for help on available commands

Creating the domain...
Writing Domain: /domains/TestDomain
Domain Created
Starting the Admin Server to test system ...
Starting weblogic server ...
WLST-WLS-1352361843103: <Nov 8, 2012 9:04:04 AM CET> <Info> <Security> <BEA-090905> <Disabling
CryptoJ JCE Provider self-integrity check for better startup performance. To enable this check,
specify -Dweblogic.security.allowCryptoJDefaultJCEVerification=true>
WLST-WLS-1352361843103: <Nov 8, 2012 9:04:04 AM CET> <Info> <Security> <BEA-090906> <Changing the
default Random Number Generator in RSA CryptoJ from ECDRNG to FIPS186PRNG. To disable this change,
specify -Dweblogic.security.allowCryptoJDefaultPRNG=true>
WLST-WLS-1352361843103: <Nov 8, 2012 9:04:04 AM CET> <Info> <WebLogicServer> <BEA-000377> <Starting
WebLogic Server with OpenJDK 64-Bit Server VM Version 20.0-b11 from Sun Microsystems Inc.>
[...]
WLST-WLS-1352361843103: <Nov 8, 2012 9:04:22 AM CET> <Notice> <Server> <BEA-002613> <Channel
"Default" is now listening on 127.0.0.1:47001 for protocols iiop, t3, ldap, snmp, http.>
```

```

WLST-WLS-1352361843103: <Nov 8, 2012 9:04:22 AM CET> <Notice> <Server> <BEA-002613> <Channel
"DefaultSecure" is now listening on 127.0.0.1:47002 for protocols iiopts, t3s, ldaps, https.>
WLST-WLS-1352361843103: <Nov 8, 2012 9:04:22 AM CET> <Notice> <WebLogicServer> <BEA-000329> <Started
WebLogic Admin Server "AdminServer" for domain "TestDomain" running in Production Mode>
WLST-WLS-1352361843103: <Nov 8, 2012 9:04:22 AM CET> <Notice> <WebLogicServer> <BEA-000365> <Server
state changed to RUNNING>
WLST-WLS-1352361843103: <Nov 8, 2012 9:04:22 AM CET> <Notice> <WebLogicServer> <BEA-000360> <Server
started in RUNNING mode>
Server started successfully.
Started the Admin Server
Connecting to t3://localhost:47001 with userid weblogic ...
Successfully connected to Admin Server 'AdminServer' that belongs to domain 'TestDomain'.

Warning: An insecure protocol was used to connect to the
server. To ensure on-the-wire security, the SSL port or
Admin port should be used instead.

Connected
Shutting down the Admin Server...
Shutting down the server AdminServer with force=false while connected to AdminServer ...
WLST-WLS-1352361843103: <Nov 8, 2012 9:04:40 AM CET> <Alert> <WebLogicServer> <BEA-000396> <Server
shutdown has been requested by weblogic>
WLST-WLS-1352361843103: <Nov 8, 2012 9:04:40 AM CET> <Notice> <WebLogicServer> <BEA-000365> <Server
state changed to SUSPENDING>
WLST-WLS-1352361843103: <Nov 8, 2012 9:04:40 AM CET> <Notice> <HTTP> <BEA-101278> <There are no
active sessions. The Web service is ready to suspend.>
WLST-WLS-1352361843103: <Nov 8, 2012 9:04:40 AM CET> <Notice> <WebLogicServer> <BEA-000365> <Server
state changed to ADMIN>
WLST-WLS-1352361843103: <Nov 8, 2012 9:04:40 AM CET> <Notice> <WebLogicServer> <BEA-000365> <Server
state changed to SHUTTING DOWN>
WLST-WLS-1352361843103: <Nov 8, 2012 9:04:40 AM CET> <Notice> <Server> <BEA-002607> <Channel
"DefaultSecure" listening on 127.0.0.1:47002 was shutdown.>
WLST-WLS-1352361843103: <Nov 8, 2012 9:04:40 AM CET> <Notice> <Server> <BEA-002607> <Channel
"Default" listening on 127.0.0.1:47001 was shutdown.>
WLST-WLS-1352361843103: Stopped draining WLST-WLS-1352361843103
WLST-WLS-1352361843103: Stopped draining WLST-WLS-1352361843103
WLST lost connection to the WebLogic Server that you were
connected to, this may happen if the server was shutdown or
partitioned. You will have to re-connect to the server once the
server is available.
Disconnected from weblogic server: AdminServer
Disconnected from weblogic server:
Exiting...

```

Note: Starting the AdminServer in a WLST script is nice for testing but is NOT a real production scenario. Under the cover, WLST does a little bit more and especially does not (!) use the start scripts generated in your domain folder.

Especially examine the following line in the script:

```

startServer('AdminServer', domainName , connUri, 'weblogic', 'test1234',
domainsDirectory+'/' +domainName, 'true', 60000, 'false');

```

The script starts the AdminServer and also provides a username and password.

The normal way to start the AdminServer (unless you are using the NodeManager, which is rather uncommon) is to use the generated startWeblogic.sh script, which is generated in your root folder of your new domain.

The addTemplate() Command

The *addTemplate* command is used to load extension templates. This function is similar to “Extend existing domain” in the GUI mode of the domain configuration.

Server Identity

The next step is to start your domain using the `startWeblogic.sh` script, which is generated in your root folder of your new domain. This will reveal a weakness of the domain generation script of the previous section:

```
[...]
<Nov 8, 2012 8:54:00 AM CET> <Info> <WebLogicServer> <BEA-000377> <Starting WebLogic Server with
OpenJDK 64-Bit Server VM Version 20.0-b11 from Sun Microsystems Inc.>
<Nov 8, 2012 8:54:01 AM CET> <Info> <Management> <BEA-141107> <Version: WebLogic Server 10.3.5.0 Fri
Apr 1 20:20:06 PDT 2011 1398638 >
<Nov 8, 2012 8:54:02 AM CET> <Info> <Security> <BEA-090065> <Getting boot identity from user.>
Enter username to boot WebLogic server:weblogic
Enter password to boot WebLogic server:
<Nov 8, 2012 8:54:14 AM CET> <Notice> <WebLogicServer> <BEA-000365> <Server state changed to
STARTING>
<Nov 8, 2012 8:54:14 AM CET> <Info> <WorkManager> <BEA-002900> <Initializing self-tuning thread pool>
<Nov 8, 2012 8:54:14 AM CET> <Notice> <Log Management> <BEA-170019> <The server log file
/domains/TestDomain/servers/AdminServer/logs/AdminServer.log is opened. All server side log events
will be written to this file.>
<Nov 8, 2012 8:54:18 AM CET> <Notice> <Security> <BEA-090082> <Security initializing using security
realm myrealm.>
<Nov 8, 2012 8:54:22 AM CET> <Notice> <WebLogicServer> <BEA-000365> <Server state changed to STANDBY>
<Nov 8, 2012 8:54:22 AM CET> <Notice> <WebLogicServer> <BEA-000365> <Server state changed to
STARTING>
[...]
] The system is vulnerable to security attacks, since it trusts certificates signed by the demo
trusted CA.>
<Nov 8, 2012 8:54:28 AM CET> <Notice> <Server> <BEA-002613> <Channel "Default" is now listening on
127.0.0.1:47001 for protocols iiop, t3, ldap, snmp, http.>
<Nov 8, 2012 8:54:28 AM CET> <Notice> <Server> <BEA-002613> <Channel "DefaultSecure" is now listening
on 127.0.0.1:47002 for protocols iiops, t3s, ldaps, https.>
<Nov 8, 2012 8:54:28 AM CET> <Notice> <WebLogicServer> <BEA-000329> <Started WebLogic Admin Server
"AdminServer" for domain "TestDomain" running in Production Mode>
<Nov 8, 2012 8:54:28 AM CET> <Notice> <WebLogicServer> <BEA-000365> <Server state changed to RUNNING>
<Nov 8, 2012 8:54:28 AM CET> <Notice> <WebLogicServer> <BEA-000360> <Server started in RUNNING mode>
```

Examine the lines in bold. You need to provide username and password, which might be ok for your private development instance, but as soon as the WebLogic instances are managed by an administrator group or must be started/restarted automatically in production environments, this is not acceptable. The reason is that somehow the AdminServer (same issue exists for the Managed-Server) needs to know what its own identity is as the server process must also authenticate itself.

The WebLogic solution is a file called `boot.properties`, which contains the user and password for the server itself. During the first start of the server (admin or managed) the password and username in this file will be encrypted. This property file is located in the `<domain>/servers/<servername>/security` directory.

Let us extend our script so that this file will be generated during domain setup:

create_simple_domain_(extended)

```
#####
# Create a very simple domain with only one admin server !!
#####

from java.io import FileInputStream
from java.io import File
```

```

print 'Creating the domain...'
domainsDirectory='/user_data/weblogic_domains';
domainName = 'TestDomain';
readTemplate('/opt/wls/wlserver_10.3/common/templates/domains/wls.jar');

# Setting listen address/port
cd('/Server/AdminServer')
set('ListenAddress', 'localhost');
set('ListenPort', 47001);

# SSL Settings
create('AdminServer', 'SSL')
cd('/SSL/AdminServer')
set('Enabled', 'true');
set('ListenPort', 47002);

# Setting the username/password
cd('/Security/'+ domainName +'/User/weblogic');
cmo.setName('weblogic');
cmo.setPassword('test1234');

setOption('CreateStartMenu', 'false');
setOption('ServerStartMode', 'prod');
setOption('JavaHome', '/usr/lib/jvm/jre-1.6.0-openjdk.x86_64');
setOption('OverwriteDomain', 'false');

print 'Writing Domain: '+ domainsDirectory+'/'+domainName;
writeDomain( domainsDirectory+'/'+domainName);
closeTemplate();
print 'Domain Created';

# Create boot.properties file !
os.makedirs( domainsDirectory+"/"+domainName+ "/servers/AdminServer/security"
f=open( domainsDirectory+"/"+domainName +
"/servers/AdminServer/security/boot.properties" , 'w')
f.write('username=weblogic')
f.write('password=test1234')
f.flush()
f.close()

connUri = 't3://localhost:47001';
print 'Starting the Admin Server to test system ...';
startServer('AdminServer', domainName , connUri, 'weblogic', 'test1234',
domainsDirectory+'/'+domainName, 'true', 60000, 'false');
print 'Started the Admin Server';

# Connecting to the Admin Server
connect('weblogic', 'test1234', connUri);
print 'Connected';

print 'Shutting down the Admin Server...';
shutdown();
print 'Exiting...';
exit();

```

Examine the marked section. This script creates the security directory and borrows the "File" API from Java in order to create the boot.properties file.

After the script has finished (remember the AdminServer was started for the connection test) the file contains the encrypted values:

```

password={AES}s5cM1qVtFbJ/nKIgk68K0fV9K6JGpAGExf4rP0Mxsvs\=
username={AES}6F/9mt88/+yxz2bgn0FvunWEJE+CfEl0Xe2xGJMM/eg\=

```

Now we have created a minimal domain with nothing else but the AdminServer. However, this domain is ready for use and fully functional. In the next sections we

will extend this domain step-by-step in order to accommodate more features and functionalities.

This script will create the same domain as the script in the previous section, but due to the create identity boot file, there won't be any standard input requests for user or password.

Later in the book we will discuss a better and more secure way to work with user credentials at the client side.

Configure Domain and Server Logging Settings

The domain creates log files and every server - including the AdminServer - also creates log files. Especially for production systems, log files must be saved for future reviews. Log files must also be rotated in order to limit size or separate timeframes. For example, this may result in having one log file for each day.

On production systems, it is a good practice to rotate log files once per day. This makes it easy for scheduled processes to move old log files to a backup or log server in order to save disk space and to avoid filling the local disk with logs. For example, rotating logs every day at 2am means that a scheduled process can run at 2:30am and move the old log file away.

The Domain Log File

The following is an example for changing the settings for the domain log.

Offline example for rotating based on time during domain creation:

```
create('TestDomain', 'Log')
cd('/Log/TestDomain')
cmo.setRotationType('byTime')
cmo.setRotationTime('02:00')
cmo.setFileTimeSpan(24)
cmo.setNumberOfFilesLimited(true)
cmo.setFileCount(7)
cmo.setFileMinSize(10000)
```

During domain creation, we need to create the "Log" entry underneath the domain first, otherwise the `cd` command will fail. Then you can set the desired log settings. You can set the rotation time based on size or time, and you can set log file sizes and amount of files, as well as a number of other settings. The settings that are available and which make sense depend on the rotation time.

Offline example for rotating based on size during domain creation:

```

create('TestDomain', 'Log')
cd('/Log/TestDomain')
cmo.setRotationType('bySize')
cmo.setRotateLogOnStartup(false)
cmo.setFileCount(18)
cmo.setNumberOfFilesLimited(false)
cmo.setFileMinSize(10000)

```

For the domain itself, a default (even if empty) LOG branch will be created if not done by you (see above). This means that in this case for an online script, we do not need to create this MBean. Therefore, in the online example below, the create statement is missing. Log file settings must be done on the Edit-MBeanServer, therefore we need to switch to the edit mode first and then we can change these settings.

Online example:

```

connect(...)
edit()
startEdit()
cd('/Log/TestDomain')
cmo.setRotationType('bySize')
cmo.setRotateLogOnStartup(false)
cmo.setFileCount(18)
cmo.setNumberOfFilesLimited(false)
cmo.setFileMinSize(10000)
activate()
disconnect()

```

The Administration Server Logs

Logs for the Administration Server are a little bit different than for the domain. With the AdminServer we are touching a real server, which consists of multiple log files. Not all of them can be configured using WLST-MBeans. The files that can be configured are the server log file and the server access file. These two are located in two different MBeans

Offline example while creating the domain for AdminServer log-settings:

```

# change settings for the server log
cd('/Server/AdminServer')
create('AdminServer', 'Log')
cd('/Servers/AdminServer/Log/AdminServer')

# set rotation type , time and other information
cmo.setRotationType('byTime')
cmo.setRotationTime('02:00')
cmo.setFileTimeSpan(24)
cmo.setNumberOfFilesLimited(true)
cmo.setFileCount(7)
cmo.setRotateLogOnStartup(true)

cd('/Server/'+servername)
create(servername, 'WebServer')
cd('/Server/'+servername+'/WebServer/'+servername)

# now do the same for the access log but you need another mbean
create(servername, 'WebServerLog')
cd('/Server/'+servername+'/WebServer/'+servername+'/WebServerLog/'+servername)
cmo.setRotationTime('02:00')
cmo.setRotationType('byTime')

```

```
cmo.setFileTimeSpan(24)
cmo.setFileCount(14)
cmo.setFileName(serverLogPath+'/access.log')
```

The above script only shows some of the possible settings available for logging. In addition to the above, settings like LogFileRotationDir, filters, and more can be configured.

Changing the Logs of all Managed-Servers

The following is actually an example for a common server farm requirement, especially for production systems. For production systems, it is common to dedicate the AdminServer to administration tasks only and delegate the application work to all the Managed-Servers. This means that usually all Managed-Servers have different log file requirements than the AdminServer.

This script shows two nice WLS/T features:

- Iterating over all servers of a domain
- Settings options depending on a server name

Example script for changing log settings for all Managed-Servers but NOT for the AdminServer:

```
# connect to the adminserver and start an edit session
connect(...)
edit()
startEdit()

print 'Iterate over the managed servers and set the log settings for all managed servers ';
domainConfig()

# get the list of servers from the config MBean tree
svrs = cmo.getServers()

# switch to the domain runtime tree
domainRuntime()
for server in svrs:
    # Do not set the log settings for the adminserver
    myServerName = server.getName()
    if myServerName != 'AdminServer':
        # set the log settings
        cd('/Servers/'+myServerName+'/Log/'+myServerName)
        cmo.setRotationType('byTime')
        cmo.setRotationTime('02:00')
        cmo.setFileTimeSpan(24)
        cmo.setNumberOfFilesLimited(true)
        cmo.setFileCount(7)
        cmo.setRotateLogOnStartup(true)

# activate the changes
activate()
disconnect()
```

The normal log file MBean for a server is located at:

```
/Servers/<serverName>/Log/<serverName>.
```

For example, for the AdminServer this is /Servers/AdminServer/Log/AdminServer. The log file MBean for HTTP access is located at:

```
/Servers/<serverName>/WebServer/<serverName>/WebServerLog/<serverName>
```

For production and production-like systems, it is necessary to separate the logs from the domain installation. Logs will grow and will likely need other file system access rights. Logs should also be saved on their own file system to protect the domains. For all production systems I have been involved with, logs have been saved outside of the domain file system tree.

In order to change the location of all logs, there are a couple of steps necessary as we are talking about different files. The files involved include

- The domain log
- The AdminServer log
- The AdminServer stdout, stderr
- The AdminServer access log
- The logs of each Managed-Server
- The stdout, stderr of each Managed-Server
- The access log of each Managed-Server

The following table provides an overview what needs to be done in order to really move all logs to a separate location.

domain log	Can be changed via WLST. Should be changed during domain creation: Use <pre>cd('/Log/<domainName>')</pre> <pre>cmo.setFileName(<new log location>)</pre> to change the location of this log file
AdminServer log	Can be changed via WLST. Should be changed during domain creation: Use <pre>[.. mbean creation if needed ..]</pre> <pre>cd('/Server/AdminServer/Log/AdminServer')</pre> <pre>cmo.setFileName(<new log location and name>)</pre> to change the location of this log file
AdminServer stdout, stderr	This cannot be done with WLST as the AdminServer is normally started using the startWeblogic script from the domain Root. What you can do is to overwrite this file during domain creation and add system redirections.

	e.g. for Linux replace the last line this script with <pre>{DOMAIN_HOME}/bin/startWebLogic.sh \$* >> <outfile location> 2>> <errorfile location></pre>
AdminServer access log	Can be changed via WLST. Should be changed during domain creation: Use <pre>[.. mbean creation if needed ...] cd('/Server/AdminServer/WebServer/AdminServer/WebServerLog/AdminServer') cmo.setFileName(<new access log location and name>)</pre> to change the location of this log file
logs of each Managed-Server	Can be changed via WLST. Should be changed during domain creation: Use <pre>[.. mbean creation if needed ...] cd('/Server/<servername>/Log/<servername>') cmo.setFileName(<new log location and name>)</pre> to change the location of this log file
stdout, stderr of each Managed-Server	This really depends how you start the Managed-Server. If you start the Managed-Server with a script, then you can do the same as described above for the AdminServer out/err If you start the Managed-Server using the NodeManager (which is the preferred way) you can do this via WLST. In this case you need to add two arguments to the ServerStart MBean. (See adding Managed-Server section for more details). You need to add the options -Dweblogic.Stdout=<outfile> and -Dweblogic.Stderr=<errorfile>
access log of each Managed-Server	Can be changed via WLST. Should be changed during domain creation: Use <pre>[.. mbean creation if needed ...] cd('/Server/'+servername+'/WebServer/'+servername+'/WebServerLog/'+servername) cmo.setFileName(<new log path>+'/access.log')</pre> to change the location of this log file

Table 5.1: Changing Logs

In production systems, it has worked out well if you combine all logs from one machine in a single place. For example, you can use "/logs" as root folder for all logs and then create a directory for each domain under this root folder. In the domain folder you can place the domain log(s), and in the domain folder you can create a subfolder for each server (including the AdminServer). The creation of these directories can be done with WLST while you configure your domain.

For example:

```

domainLogPath = <logsDirectory>+"/"+<domainName>;
try:
    os.makedirs(domainLogPath);
except:
    print 'Unable to create domain root log path - please check !';
print('Setting Domain Log...');
create(<domainName>, 'Log')
cd('/Log/'+<domainName>)
cmo.setFileName(domainLogPath+'/'+'<name of domain log>')

```

Note that the complete domain creation example will contain this.

Configure Domain Log Filters

In every WebLogic domain exists a domain-wide log file. The domain log filter specifies which messages are sent to the domain log. The message must pass all criteria in the filter to be logged. By default, messages are not filtered so that all messages are reported to the log. For example, it is possible to filter by user-id or set a minimum level of severity or filter by subsystem.

The following two scripts show examples of creating log filters for the domain log.

The first example sets a filter in order to filter all events that are ERROR from the local machine or not (!) coming from 'martin':

```

cd('/')
cmo.createLogFilter('LogFilter-F1')

cd('/LogFilters/LogFilter-F1')
cmo.setFilterExpression('(USERID != \'martin\') OR (MACHINE = \'localhost\') AND (SEVERITY = \'ERROR\')')

```

The second example creates a filter in order to log all events from the subsystem 'test' or from the user 'abcdefg':

```

cd('/')
cmo.createLogFilter('LogFilter-F2')

cd('/LogFilters/LogFilter-F2')
cmo.setFilterExpression('(SUBSYSTEM = \'test\') OR (USERID = \'abcdefg\')')

```

Activate log4j

WebLogic is using an implementation based on the Java Logging APIs by default. Most applications are using log4j. Many administrators have experience in building log file monitors and parsers based on the log4j syntax. The WebLogic log files can be changed to also use log4j.

The following script defines a function that can be used to change the logging implementation to log4j.

The WLST script to enable Log4j Logger on a WebLogic Server is as follows. WebLogic logging services use an implementation based on the Java:

```
def changeLoggingToLog4j(serverName) :
    edit()
    startEdit()
    cd('/Servers/'+serverName+'/Log/'+serverName)
    cmo.setLog4jLoggingEnabled(true)
    save()
    activate()
```

Log4j in WebLogic is available only for server-side and not for client-side logging.

Cluster

Besides the Administration Server, Managed-Servers can be grouped into clusters. Every WebLogic domain can host multiple clusters, but each Managed-Server can only be a member of one cluster (if at all). Every cluster can (but does not need to) include a Managed-Server hosted on different machines. As described in the introduction, there are two different communication models for clusters: Unicast and Multicast.

First of all, let us create one cluster for each communication model with only the necessary configuration parameters.

Creating a unicast cluster:

```
cd('/')
cmo.createCluster('Cluster-UnicastTest')

cd('/Clusters/Cluster-UnicastTest')
cmo.setClusterMessagingMode('unicast')
```

In WLST, switch to root and use this MBean (cmo) to create a new cluster object. Afterwards we are only setting the cluster mode to *unicast*.

Creating a multicast cluster:

```
cd('/')
cmo.createCluster('Cluster-MulticastTest')

cd('/Clusters/Cluster-MulticastTest')
cmo.setClusterMessagingMode('multicast')
cmo.setMulticastAddress('239.192.0.0')
cmo.setMulticastPort(12345)
```

In WLST, switch to root and use this MBean (cmo) to create a new cluster object. Afterwards we are setting the cluster mode to *multicast*. In addition, we need to set the multicast broadcast address and port.

Multicast address: enter the multicast address for the cluster. Cluster members that communicate with each other use this multicast address. The default value is 239.192.0.0. You can choose valid multicast address range of 224.0.0.1 through 239.255.255.255.

Multicast port: enter the multicast port for the cluster. This multicast port is used for cluster members to communicate with each other. The default value is 7001. Here you can use any valid values for multicast ports ranging from 1 through 65534. The ports are not enabled to serve multicast for most machines in real-time. Multicast uses UDP protocol.

Now let us as add another unicast cluster with some more configuration options:

```
cd('/Clusters/Cluster-UnicastTest')
cmo.setNumberOfServersInClusterAddress(3)
cmo.setDefaultLoadAlgorithm('weight-based')
cmo.setClusterType('none')
cmo.setPersistSessionsOnShutdown(false)
cmo.setReplicationChannel('ReplicationChannel')
cmo.setSecureReplicationEnabled(false)
```

NumberOfServersInClusterAddress: This is the amount of servers the cluster will include in the cluster address. This option will not be used if the address is explicitly set. Usually WebLogic maintains this setting automatically.

DefaultLoadAlgorithm: This option specifies the default way load-balancing is used between the appropriate servers of the cluster. This setting only takes effect if it is not overwritten. Options include round-robin (one after the other), weight-based (round-robin with defined weight for each server), or random (random access).

Note the attribute *ClusterType*. This specifies a possibility to cluster between clusters hosted in DIFFERENT WebLogic domains. WebLogic supports three different settings: *none* (own cluster only), *man* (synchronous replication), and *wan* (asynchronous replication).

Here is an example for setting up the cluster in order to replicate to a remote cluster in a different domain:

```
cd('/Clusters/MartinTest_Cluster')
cmo.setClusterType('man')
cmo.setPersistSessionsOnShutdown(false)
cmo.setReplicationChannel('ReplicationChannel')
cmo.setRemoteClusterAddress('192.178.34.123')
cmo.setSecureReplicationEnabled(false)
```

If *wan* is used, which normally means replication across datacenters, the sessions must be stored in a database. Therefore a datasource must be provided for this type of replication:

```
cmo.setClusterType('wan')
cmo.setRemoteClusterAddress('11.22.33.44')
cmo.setDataSourceForSessionPersistence(getMBean('/SystemResources/Prototype_Datasource'))
```

Singleton services are special services you can implement which must be available only once in the whole cluster. Therefore these services will be migrated automatically if the server they are hosted on fails.

Define a singleton service:

```
cd('/')
cmo.createSingletonService('SingletonService-Test')

cd('/SingletonServices/SingletonService-Test')
cmo.setClassName('de.book.test.singleton1')
cmo.setCluster(getMBean('/Clusters/MartinTest_Cluster'))
cmo.setUserPreferredServer(getMBean('/Servers/MartinTest_Domain_MS1'))
set('ConstrainedCandidateServers', jarray.array([ObjectName('com.bea.Name=MartinTest_Domain_MS2, Type=Server'), ObjectName('com.bea.Name=MartinTest_Domain_MS1, Type=Server')], ObjectName))
```

New Dynamic Cluster Feature

12.1.2 ++

WebLogic 12.1.2 has added a complete new way of working with clusters. Up to WebLogic 12.1.1, a cluster was defined out of a list of already defined servers. The new feature is called dynamic clusters. The main concept is that the cluster member (servers) will not be defined individually in the WebLogic configuration but will be constructed based on a server template. A server template is basically a blueprint that is used to create Managed-Servers.

With dynamic clusters it is easy to scale up. During setup of the dynamic cluster, the administrator will set the maximum number of server instances you will need to cope with peak loads. WebLogic will then create the different server instances based on the server template and will replace the so-called calculated attributes during the creation.

Calculated attributes include the name of the server and a number of optional values like different ports, destination machines, and network channels (listen ports). Server names, for example, will be constructed out of a given prefix string and an index. The *CalculatedListenPorts* attributes defines how server ports are calculated. Ports that can be calculated include the plain and SSL port for the default channel, replication ports, and other network channels. Port calculations can also be disabled. The cluster configurations *CalculatedMachineNames* and *MachineNameMatchExpression* define the way WebLogic will assign servers to machines.

Here is an example for creating a dynamic cluster:

```
cd('/')
cmo.createCluster('Cluster-12436')

cd('/Clusters/Cluster-12436')
```

```

cmo.setClusterMessagingMode('unicast')
cd('/ServerTemplates/Cluster-12436-Template')
cmo.setCluster(getMBean('/Clusters/Cluster-12436'))

cd('/Clusters/Cluster-12436/DynamicServers/Cluster-12436')
cmo.setServerTemplate(getMBean('/ServerTemplates/Cluster-12436-Template'))
cmo.setMaximumDynamicServerCount(5)
cmo.setCalculatedListenPorts(true)
cmo.setCalculatedMachineNames(false)
cmo.setCalculatedListenPorts(true)
cmo.setServerNamePrefix('Cluster-12436-')

```

Server Templates

Server templates are new configuration items in WLS 12.1.2 which allow an administrator to define a template for a dynamic server. This template can get quite complex as WebLogic offers a huge number of settings for a Managed-Server. This includes general settings like machine, ports, protocols, and network channel. It also includes special dynamic settings like settings for replication group, cluster weight, replication ports, and preferred secondary group. It also can include settings for server services like JMS configurations. In addition to the well-known categories, templates can be used to define configurations in many other aspects of WebLogic, such as security settings.

Here is an example of creating a server template:

```

cd('/')
cmo.createServerTemplate('ServerTemplate-1234')

cd('/ServerTemplates/ServerTemplate-1234')
cmo.setJMSThreadPoolSize(0)
cmo.setXMLRegistry(None)
cmo.setXMLEntityCache(None)

cd('/ServerTemplates/ServerTemplate-1234/TransactionLogJDBCStore/ServerTemplate-1234')
cmo.setEnabled(false)

```

A more complex template will be demonstrated in the following WLST script. Always keep in mind that, more or less, all settings that are available for Managed-Servers can be part of such a template.

```

cd('/')
cmo.createServerTemplate('Cluster-12436-Template')

cd('/ServerTemplates/Cluster-12436-Template')
cmo.setListenPort(7100)

cd('/ServerTemplates/Cluster-12436-Template/SSL/Cluster-12436-Template')
cmo.setListenPort(8100)

cd('/ServerTemplates/Cluster-12436-Template/WebServer/Cluster-12436-Template/WebServerLog/Cluster-12436-Template')
cmo.setNumberOfFilesLimited(false)

cd('/ServerTemplates/Cluster-12436-Template')
cmo.setListenPort(7100)

cd('/ServerTemplates/Cluster-12436-Template/SSL/Cluster-12436-Template')
cmo.setListenPort(8100)

cd('/ServerTemplates/Cluster-12436-Template')

```

```
cmo.setMachine(getMBean('/Machines/localhost'))
```

Starting Dynamic Clusters

Starting a dynamic cluster will be done in exactly the way as normal clusters.

Complete Example

The following is a complete example of creating a server template and a cluster.



create_server_template/cluster

```
# change to the root folder
cd('/')

# create a new server template
cmo.createServerTemplate('MyDynServer_Template')

# change to the new server template
cd('/ServerTemplates/MyDynServer_Template')
cmo.setListenPort(6000)

# change the SSL port
cd('/ServerTemplates/MyDynServer_Template/SSL/MyDynServer_Template')
cmo.setListenPort(6100)

# set a machine - None means that the adminserver can target the new instances dynamically
cd('/ServerTemplates/MyDynServer_Template')
cmo.setMachine(None)

# create a new dynamic cluster
cd('/')
cmo.createCluster('Cluster-1234')

# configure the cluster
cd('/Clusters/Cluster-1234')
cmo.setClusterMessagingMode('unicast')

# set the template which should be used to create server instances
cd('/ServerTemplates/MyDynServer_Template')
cmo.setCluster(getMBean('/Clusters/Cluster-1234'))

# configure the cluster
cd('/Clusters/Cluster-1234/DynamicServers/Cluster-1234')
cmo.setServerTemplate(getMBean('/ServerTemplates/MyDynServer_Template'))
cmo.setMaximumDynamicServerCount(3)
cmo.setCalculatedListenPorts(true)
cmo.setCalculatedMachineNames(true)
cmo.setCalculatedListenPorts(true)
cmo.setServerNamePrefix('MyDynServer_')

# activate the changes
activate()
```

Limitations

Dynamic clusters also have a number of limitations. Due to its dynamic nature, it is not possible to target a deployment to an individual dynamic server. Also, whole server and service migration are not supported.