integral

delivering the complex

# ConfigNOW

## User Guide

integral
delivering the complex

# TABLE OF CONTENTS

integral
delivering the complex

# 1 Welcome to ConfigNOW

ConfigNOW is all about providing a way to automate important tasks that pop up as part of managing your Oracle Fusion Middleware Environment.

Got a new developer coming on to your team? You need a way for them to get their own WebLogic Server environment up to deploy applications to.

Adding a new JMS queue to your WebLogic Server domain? You'll need some way of adding it to the eight environments your code is migrated through before it hits production.

Something gone horribly wrong with your UAT domain and you need it fixed *now* to get testers back on track? It would be great to be able to blow away the domain, start from scratch and have something working in minutes.

These are the sorts of jobs that ConfigNOW does for you. If there's a whole crop of repeatable tasks that need to be done that are not interesting or challenging, but can cost days of developer time if something goes wrong, then ConfigNOW can assist.

ConfigNOW takes the basic building blocks that the Fusion Middleware stack provides and gives you a set of flexible, configurable tools to take care of the configuration management, so you can focus on developing and deploying your applications.

ConfigNOW uses WLST (WebLogic Scripting Tool) in order to create and configure domains. WLST is Oracle's recommended tool for configuration management and is a series of APIs written in Jython. To follow the stack all the way down to the bottom, Jython is a Java implementation of the Python scripting language, which is used for everything from computer games to audio processing to making your life easier with Fusion Middleware.

The Fusion Middleware stack, WebLogic Server particularly, represents most of the moving parts of your runtime environment as MBeans (or managed beans) using JMX – Java Management Extensions. This means that no matter what underlying issue you're dealing with – a JMS queue, a JDBC connection pool or an OSB proxy service, you interact with that object using the same fundamental principles – a set of properties and methods.

WLST can deal with the underlying MBeans, which makes the scripted configuration of objects in your environment straightforward. On top of this, WLST also provides Jython methods to make the process of connecting to existing domains online, or reading the contents of the entire domain while it's down, relatively painless.

Because ConfigNOW is developed as a library of scripts, this makes it easy to extend if there's particular functionality not currently available.

There's more information on extending ConfigNOW available in the "Extending " section of this document.

# 2 ConfigNOW Notes

Please note that this release of ConfigNOW has the following caveats:

- Clustering and high availability, while supported, have had limited testing
- The only platforms tested on are Windows and Linux
- WLST commands are not supported by plug-ins or validators

The following new features have been included as part of the final 4.1.8 release:

- Support for Oracle Service Bus 10g and 11g installation
- Support for SOA Suite 11g and 12.1.3 installation
- Support for 64bit WebLogic installation
- Support for encrypting passwords in the property files
- Support for global properties, which are properties that are available to all commands.
- Support for oPatch commands

If you were previously a RECT user, or are just starting out with ConfigNOW at the moment, we'd love to hear from you about your experiences. If there's something that's been particularly helpful, we're always up for praise!

And if a particular feature been a problem for you, or there's something missing, please let us know.

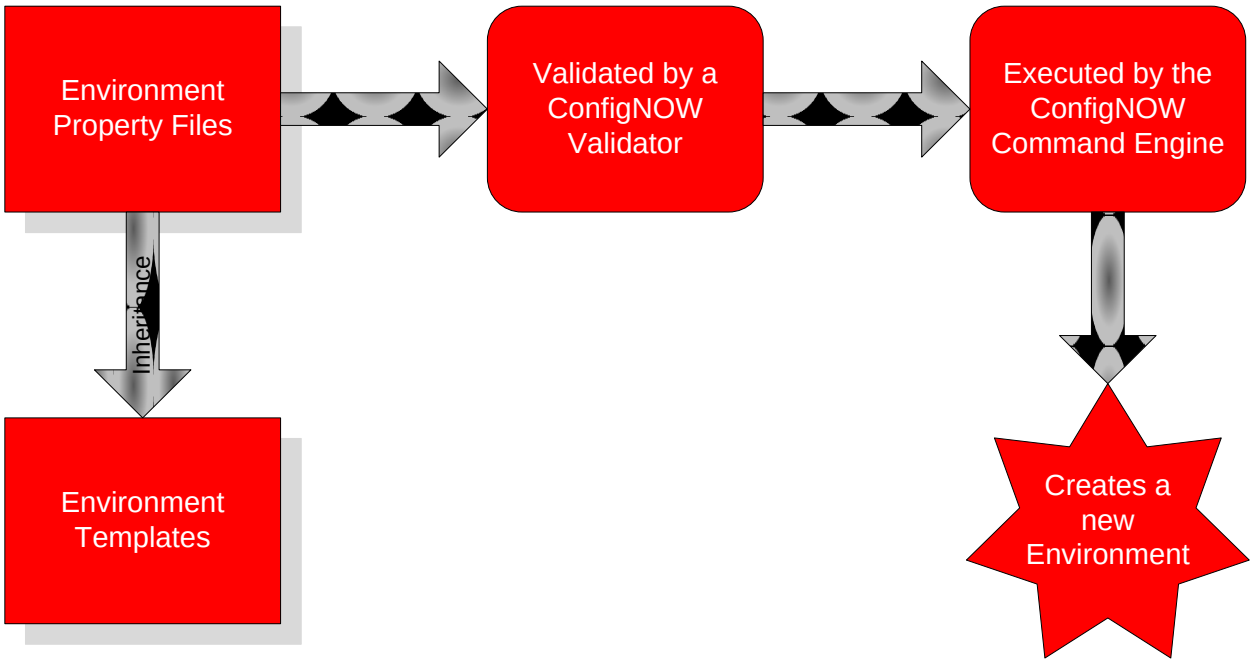
The perfect forum for your feedback is http://www.integraltech.com.au/forum

Alternatively, you can provide feedback and support requests regarding ConfigNOW to supportnow@integraltech.com.au

# 3 Key ConfigNOW Concepts

ConfigNOW is built up of three main concepts:

- Property files - which define entire WebLogic based environments
- Validator - which validates the property files as they are used to ensure that they are correct
- Command - there are inbuilt commands, plus you can dynamically and easily add your own commands to ConfigNOW to build specific components of your environment



integral
delivering the complex

## 3.1 Property Files Defining Configurations

ConfigNOW uses the concept of defining an entire environment as a property file, or potentially a set of property files though inheritance. Each environment is defined as a separate property file, and you can then execute commands against that defined configuration using the ConfigNOW tool. We will look at the concept of commands later - firstly let's consider the properties files.

The basic concept of the properties file is actually very simple. It consists of a file with named properties and values in it. Here is a segment of the:

`[CFGNOW_HOME]/config/templates/soa11g_template.properties`

file as an example.

```
######################################################################
### The properties below are interpreted by the ConfigNOW engine
######################################################################

#######################################
#           WebLogic
#######################################
wls.bea.home=${wls.oracle.home}
wls.name=wlserver_10.3

soa.name=soa
```

Note how some properties such as `soa.name` are set absolutely; others such as `wls.bea.home` use the `${xyz.abc}` format to set their values using other properties.

While you can define your own properties, ConfigNOW uses many out of the box. The full list of properties can be found in the *ConfigNOW Properties Reference*.

### 3.1.1 Built in Properties and Reserved Properties

ConfigNOW automatically creates a number of built in properties that start with the name ConfigNOW. The ConfigNOW name is considered a reserved name and as such you should not create any of your own properties that start with the word ConfigNOW.

The properties created automatically by ConfigNOW are:

| Property name | What it's used for |
|---|---|
| ConfigNOW.config_file_location | Provides the full file name of the configuration file that is being run, i.e.<br>`config\environments\local\simple_inherit` |
| ConfigNOW.configuration | The configuration file as passed in on the command line as an argument, i.e.<br>`Simple_inherit` |
| ConfigNOW.environment | The environment as passed in on the command line as an argument, i.e.<br>`local` |
| ConfigNOW.home | The location of the ConfigNOW instance being run, i.e.<br>`D:\Work\ConfigNOW\trunk\ConfigNOW` |

### 3.1.2 Property Inheritance: Saving Your Time

The key to getting consistent and repeatable installations of your WebLogic based environments is to be able to accurately account for the differences between each environment.

For example, the key differences between your development and production environments may only be the matter of a couple of different configuration options such as ports, domain names, database connections, etc.

ConfigNOW solves this problem through the process of property inheritance, where one property file can inherit and then override the properties from another file. The advantage of this approach is simply that a new environment can be established in only a few lines of configuration.

For example, consider the simple domain template located in:

`[CFGNOW_HOME]/config/environments/local/simple.properties`

```
base=config/templates/wl_as_template.properties

wls.oracle.home=c:/oracle/middleware
wls.name=wlserver_10.3
wls.domain.javahome=${wls.oracle.home}/jrockit_160_05

wls.domain.dir=${wls.oracle.home}/domains
wls.domain.name=my_domain
```

The `simple.properties` template defines the property `base` (relative to [CFGNOW_HOME]) meaning that all the properties defined in the `wl_as_template.properties` file will also be included. It then overrides that file with a number of local settings such as `wls.oracle.home` while at the same time referencing many of the properties through the use of the curly brackets, for example `${wls.oracle.home}`.

There are two key advantages in using this approach:

1. New environments can be created very quickly. For example as you can see above, creating a new WebLogic environment can be reduced to six lines of configuration.
2. Inheritance allows you to quickly and easily see the differences between environments. So for example when you get an issue in production that is not occurring in acceptance, you can examine the files to see what the differences are between the two environments, simply and quickly.

If you are developing a properties file that is going to be overridden at a later stage (for example, a template file) then you can use the question mark to indicate that a property needs to be defined before the configuration will be valid. For example, setting a property of wls.oracle.home=? indicates that any properties file based off the template must explicitly define the wls.oracle.home property.

| Template File: | Properties file based off Template: |
|---|---|
| `config/templates/wl_as_template.properties`<br>`wls.oracle.home=?` | `config/environments/local/simple.properties`<br>`base=config/templates/wls as template.properties`<br>`wls.oracle.home=c:/oracle/middleware` |

ConfigNOW 4.1 has a second form of inheritance that allows you to not only inherit an entire property file as described above, but also inherit groups of properties

Consider the scenario of servers in a WebLogic environment. Most environments define multiple machines and in most environments the core base properties of those machines are probably very similar. ConfigNOW allows you to define these common properties as a set of base properties, for example in

`[CFGNOW_HOME]/config/templates/wl_as_template_stripped.properties`

```
base.wls.domain.machine.machine.name=machine1
base.wls.domain.machine.machine.type=UnixMachine
base.wls.domain.machine.machine.postBindGID=nobody
base.wls.domain.machine.machine.postBindGIDEnabled=false
base.wls.domain.machine.machine.postBindUID=nobody
base.wls.domain.machine.machine.postBindUIDEnabled=false
base.wls.domain.machine.machine.nodemanager.type=Plain
base.wls.domain.machine.machine.nodemanager.address=${wl.host}
base.wls.domain.machine.machine.nodemanager.port=5002
```

Notice how the properties start with the word 'base'. This indicates that they are not a normal property, only one that can be inherited from. As the last stage of the process of building the property files for a configuration, ConfigNOW will remove any properties that start with the word base, so these properties can only be inherited from; they should not form the basis of any properties that you intend to use in commands directly.

As an example, to create a new machine from this base configuration, simply use the base attribute at the end of your property to instruct ConfigNOW to inherit from it

`[CFGNOW_HOME]/config/environments/dev/simple_inherit.properties`

```
wls.domain.machine1.base=base.wls.domain.machine.machine
```

Notice how this file has first inherited all the properties from

`[CFGNOW_HOME]/config/templates/wl_as_template_stripped.properties`

ConfigNOW uses a matching process to produce the new properties for you. It matches everything that starts with `base.wls.domain.machine.machine` and adds the properties that have been defined to `wls.domain.machine1 .`

Property level inheritance can be used in conjunction with Iterators (defined below) to further simplify your configuration.

### 3.1.3 Iterators

Iterators allow you to simply create multiple copies of a set of properties. For example, rather than defining one machine for your environment you may wish to define multiple machines based on a common base set of properties; to do this you would use the iterate attribute. The iterate attribute takes

an iterator set (defined using Jython syntax) and then iterates a set of attributes, replacing the iteration variable % with the value of the iterator. For example:

`[CFGNOW_HOME]/config/environments/dev/simple_inherit.properties`

```
# Inherit and iterate
wls.domain.machine.machine%.base=base.wls.domain.machine
wls.domain.machine.machine%.iterate=range(1,8)
wls.domain.machine.machine%.name=machine%
```

In this scenario the iterator will iterate from 1 to 7 creating machine1, machine2, machine3, etc.

There are a number of variations of Iterators you can use, as any Jython list or command that produces a Jtyhon list can be used in this way. Some examples are defined in the simple_inherit properties file:

`[CFGNOW_HOME]/config/environments/dev/simple_inherit.properties`

```
# Iteration examples

# Iterates from 0 to 2
# wls.server.server%.iterate=range(3)
# wls.server.server%.listener.port=5432%

# Iterates from 1 to 2
# wls.server.server%.iterate=range(1,3)
# wls.server.server%.listener.port=5432%

# Iterates the set 01, 02, 03, 04
# wls.server.server%.iterate=['01','02','03','04']
# wls.server.server%.listener.port=654%
```

## 3.2 Validation: Getting It Right First Time

Because the properties files can contain a large number of values, ConfigNOW supports the concept of property validation. Before any property file is able to be used it must first pass through a series of validators which ensure that the properties are valid and defined as required.

If property files fail to pass the validation test, then errors are reported and the process stops.

## 3.3 Managing Passwords

The properties files you use to provide complete configuration information for environments often contain passwords for sensitive databases, or for the WebLogic domain itself. ConfigNOW provides two main options around managing passwords.

### 3.3.1 Password Prompting

Passwords can be left with an empty value and provided that the `password.prompt` property is set, ConfigNOW will prompt for the values of these passwords. eg: `wls.admin.password=`

ConfigNOW will then prompt for the value of each password configured in this way at runtime, rather than storing the value in the properties file.

### 3.3.2 Password Encryption

ConfigNOW can also encrypt all passwords stored in the properties files for particular environments using the same powerful AES encryption used by WebLogic Server in order to protect sensitive passwords. Once the `password_encryptor` command has been used to encrypt passwords for a given properties file, they will appear similar to the following:

```
wls.admin.password={AES}pDLu56MzfKbo9uyp15A5DQ==OY2M0r8HZIQDU6n1YS7r2w==
```

## 3.4 Where To Get More Information

Integral provides support for the ConfigNOW product through its SupportNOW service. To request support or information from Integral, please send an email to:

supportnow@integraltech.com.au

Include in the email the nature of the issue that you are having and any information that you feel may be useful in supporting your call. A SupportNOW engineer will respond promptly to help you deal with your issue.

If you would prefer to speak to someone about your issue or question, you can contact us via phone on one of the following numbers:

**On call support engineer: +61 (0)425 412 474**
**Brisbane office: +61 (0) 7 3839 1477**

Please note that unless you have a 24x7 support arrangement with Integral, support is only available 9am to 5pm Monday to Friday, Australian Eastern Standard Time (+10 GMT).

# 4 Getting Started

The following Quick Start Guide should help you get started quickly and easily on a Windows platform. This guide provides the steps for creating a simple WebLogic domain consisting of a single administration server.

## 4.1 ConfigNOW Quick Start Guide – Windows

### 4.1.1 Prerequisites

You need to have the following software installed on your system before using the rest of this guide:

- A Java runtime - see www.java.com for details on how to install Java. To remain consistent with the Fusion Middleware products supported by ConfigNOW, it currently supports any Java JDK in the range of Java SE 5 and Java SE 6. If you're using ConfigNOW with Fusion Middleware, the bundled Java JDK will do fine.
- WebLogic Server - see www.oracle.com/technetwork/middleware/fusion-middleware/downloads/index.html for details on how to install WebLogic Server.

### 4.1.2 Downloading ConfigNOW

*This step is only required if you have not been provided a copy of ConfigNOW on CD or some other media.*

1. Go to the Integral Web Site at http://www.integraltech.com.au
2. Create a new account by clicking on the login button on the top right hand side of the page
3. Once you have created an account, proceed to the download section under the ConfigNOW menu and download the latest release (or most appropriate release) of the product

### 4.1.3 Running ConfigNOW

1. Start a Windows command prompt and change to your ConfigNOW Home directory, eg –

```
c:\work\configNOW
```

2. Type `configNOW` and press enter
3. The first time ConfigNOW runs, it builds a cache of jar files. You will see a number of messages starting with `*sys-package-mgr*.` This is normal behaviour and only happens the first time the product is run.
4. Finally, you should see a usage message from ConfigNOW similar to:

```
[ConfigNOW] INFO: usage: configNOW <command> [<environment> <config_file>]
```

5. Type `configNOW help` and press enter. This will show a list of available commands.

### 4.1.4 Creating a Simple WebLogic Domain

ConfigNOW uses a configuration file to describe the properties of a domain for a given environment, such as a local or production environment. The basic ConfigNOW distribution contains an environment called `local` containing a configuration file called `simple.properties`. Environments are represented as directories under the `<cfgnow_home>/config` directory.

1. Open the configuration file `<cnfgnow_home>/config/local/simple.properties` in a text editor and modify the properties as follows:

| Property Name | Purpose | Example Setting |
|---|---|---|
| `wls.oracle.home` | Set to your Oracle FMW home directory | `c:/oracle/middleware` |
| `wls.name` | Set to the WebLogic directory, contained within `wls.oracle.home` | `wlserver_10.3` |
| `wls.domain.javahome` | Set to your Weblogic Java home | `${wls.oracle.home}/jrockit_160_05` |
| `wls.domain.dir` | Set to the directory where domains should be created | `${wls.oracle.home}/domains` |
| `wls.domain.name` | Set to the name of the Weblogic domain you wish to create | `my_domain` |

Notice the first line of the configuration file:

```
base=config/templates/wl_as_template.properties
```

This tells ConfigNOW to inherit properties from a base configuration file. You can override any of the properties from the base file by giving the property a value in `simple.properties`. For example, you could change the port the admin server will listen on, from the default of 7001 specified in the base template to 8001 by adding `wls.admin.listener.port=8001` to `simple.properties`.

2. Save and close the simple.properties configuration file.
3. To start the ConfigNOW tool enter:

```
configNow create_domain local simple
```

- `create_domain` is the name of the ConfigNOW command to run
- `local` is the name of the environment being configured
- `simple` is the name of the configuration file (`simple.properties`) describing the environment

**Note:** To avoid any CLASSPATH conflicts please make sure that you run this from a clean environment.

4. The domain will be created in an offline state. The admin server can be started by running the following Windows command:

```
<wls.domain.dir>\<wls.domain.home>\startWeblogic.cmd
```

   The admin server for the domain can also be started via the `start_wls_admin` command – for more information on this, refer to the *ConfigNOW Command Reference*.

5. The admin server is listening on the default port 7001. You can access the console from `http://localhost:7001/console`. The WebLogic user name and password have been set from properties in base template

   `config/templates/wl_as_template.properties`.

#### 4.1.4.1 Validating Creating a Domain

When creating a new domain, there are a number of checks that the `create_domain` command will perform that may cause domain creation to fail, due to a configuration issue. The most likely configuration problems that can occur are:

Domain already exists – if the combination of `wls.domain.dir` and `wls.domain.name` point to a domain that already exists, then `create_domain` will fail with an error indicating that the domain already exists. In this case, confirm the configuration of the existing domain – it may be that the `create_domain` command has been run successfully previously.

If not, shut down all running servers for the domain, delete the existing domain directory, then run the command again.

Could not find Fusion Middleware installation – the `wls.oracle.home` and `wls.bea.home` properties in particular are used to locate the Fusion Middleware installation that ConfigNOW is using to create a new domain. These properties help ConfigNOW find the domain templates used to create a new domain. If these properties are not set, or their values don't point to a valid Fusion Middleware installation, then ConfigNOW will fail to create the new domain.

integral
delivering the complex

# 5 Extending ConfigNOW

While the ConfigNOW tool provides functionality straight out of the box, it has also been designed from the ground up so that it is easy to extend and customize.

Due to the highly customised nature of most client configurations, we would expect that almost all users of ConfigNOW will need to extend the solution at some stage, so it is worthwhile understanding how you go about doing that.

The first thing to be aware of is that you **should not** make changes to the ConfigNOW core which is located in the `[CFGNOW_HOME]/core` directory if you are trying to extend ConfigNOW. The core contains the main code that allows ConfigNOW to operate and it should not need to be modified unless you are creating a patch to fix an issue in the core functionality.

Customisations are achieved by simply adding new scripts or ant build files into the `[CFGNOW_HOME]/custom` directory, which can be completely customized based on your environment.

To achieve this, we use three basic concepts:

- Validators – Which validate configuration files
- Extensions – Allow you to add new commands to ConfigNOW
- Plugins – Allow you to insert logic at key points in the inbuilt commands

ConfigNOW ships with a series of examples for extensions and plugins which can be located in the `[CFGNOW_HOME]/custom` directory.

## 5.1 Validators

The purpose of a validator is to validate the contents of the properties files that you are passing to ConfigNOW to be executed.

The property file inheritance model used by ConfigNOW ensures that property files are merged before the validation process commences. So the validator is always dealing with a single set of properties that it needs to validate.

The process of validation is actually fairly straightforward. However, validators can become complex when there are complex relationships between the various properties and the way that they interrelate.

**IMPORTANT:** Validators are executed against all commands and as such should never be written to be specific to a certain command. Generally, validators should validate for correct formatting of properties (for example, to check that a comma-separated list) or dependencies (for example, if a managed server references a cluster, then that cluster must also be defined). Validators should not check for conditions that relate only to a specific command (for example, to check if a domain home exists). Command-specific validation should simply be performed by the command.

The following is a simple validator that shows some of the key steps required.

**/core/validators/templates.py**

```
from java.io import File
import validation_helper as helper
def run(domainProperties):
return helper.validateList(domainProperties, 'wls.templates')
```

All validators implement a run method and are passed a set of properties when they execute. Properties can be extracted from the set using the `getProperty` command:

```
    machines = domainProperties.getProperty('wls.domain.machines')
```

The role of the validator is simply to validate these properties and return an error code if a validation error is found. It is also important that you log the fact that the validation process has failed, so that the user of ConfigNOW can get some visibility on what has occurred. Logging of any validation errors should occur via the use of the core ConfigNOW logger. For example, this snippet of code taken from one of the core modules shows the logger in use.

**/core/engine/validation_helper.py**

```
from org.apache.log4j import Logger, PropertyConfigurator
log=Logger.getLogger('validation')
def malformed_list_error_msg(propertyName):
log.error('>>> ' + propertyName + ' is malformed.')
```

If validation is successful, the run method should return a status code of 0. If for some reason a validation error has occurred, a status code of 1 should be returned from the run method.

The validator shown above includes the validator_helper module which contains a number of commands to help validate lists of properties and the existence of individual properties within that list.

There are a number of validators that ship with ConfigNOW that you can use as examples and these can all be located in the `[CFGNOW_HOME]/core/validators`. However, remember when you develop your validators they need to be placed in the `[CFGNOW_HOME]/custom/validators` directory as code in the `[CFGNOW_HOME]/core` directory should never be modified.

Validators that are located in the `core/validators or custom/validators` directory are automatically located and executed as part of the ConfigNOW validation process when a command is run.

## 5.2 Extensions

Extensions are mechanisms that allow you to add new commands into the ConfigNOW menu that can be executed as part of a build process. Extensions are the primary mechanism that you use to add site specific functionality into a ConfigNOW configuration.

The ConfigNOW Extensions are a recognised part of the ConfigNOW capability and as such, are supported by properties files models and validators.

**IMPORTANT:** ConfigNOW 4.1 currently has a limitation that means your command name cannot include a full stop in the name. Use the _ character instead.

### 5.2.1 Ant or Jython

When developing extensions for ConfigNOW it is possible to do so using either Ant or Jython as your core development tool.

To develop a Jython based command you simply need to create a new Jython file (a file that has an extension of .py) and save that file into the `[CFGNOW_HOME]/custom/commands` directory. The Jython file must contain a function called 'run' that takes a single parameter (the configuration). When a command is executed, ConfigNOW will call the function and pass in the configuration information for the environment as the parameter. Consider the example.py file below.

**/custom/commands/example.py**

```
def run(config):
    """This is an example custom command"""
    log.debug('-- in example.py')

#   access properties like this -> config.getProperty('my.property')
```

Note how once inside the extension the configuration information can be accessed via the use of the `config.getProperty` method.

By placing the example.py file in to the commands directory, a new command called 'example' (i.e. the name of the file) will be available to ConfigNOW.

A description for a command can be added by defining it according to python doc-string syntax. In the example above, the description is `"This is an example custom command".` The command and description will be displayed in the ConfigNOW help menu which can be executed by typing `'configNOW help'`. A Jython file without a description will not be displayed in the help menu but will still be available as a hidden command.

The steps to define an Ant extension using ConfigNOW are also very simple. To develop an Ant based command, create an Ant build file and place it in the `[CFGNOW_HOME]/custom/commands/ant` directory. ConfigNOW will automatically discover Ant build files and look for targets defined in them. All targets in an Ant file are available as ConfigNOW commands. If a command contains a description it will also be available from the ConfigNOW help menu. If you don't want particular Ant operations to be visible from the help command, then simply don't set the description.

**/custom/commands/ant/example_using_ant.xml**

```
<project name="example" basedir=".">
    <target name="example_using_ant" description="This is an example custom
command, using ANT">
        <echo message="hello from ant" />
        <echo message="wls.oracle.home=${wls.oracle.home}" />
    </target>

    <target name="example_using_ant_stay_hidden">
        <echo message="will not show up in the help menu" />
    </target>
</project>
```

As with Jython extensions, the full set of environmental properties are passed to the extension as part of the process of execution. Within Ant, these properties can be referenced as ant-style property references.

## 5.3 Plug-ins

Plug-ins provide a mechanism to allow you to easily extend built in commands or custom commands for particular environments or purposes.

Because we don't encourage the modification of core capabilities in the ConfigNOW solution, the plug-in approach allows you to perform certain operations either pre or post any defined command or extension in ConfigNOW. Some commands also contain other plug-in points in addition to the default pre and post plug-in points.

Plug-ins can be defined by convention or by configuration.

When defining plug-ins by convention, the naming of the Jython script is the key. The file should be placed in the `[CFGNOW_HOME]/custom/plugins` directory with a file name that is the same as the command to be executed, followed by _pre to execute the plug-in before the command or _post to execute it after the command. The following example shows a Jython script that will be executed before the command `example_using_ant`.

**/custom/plugins/example_using_ant_pre.py**

```
def run():
    log.info('-- in example_using_ant_pre.py - doing some global custom
stuff pre ant task...')
```

The second example shows a Jython script that will be executed after the execution of the `example` command.

**/custom/plugins/example_post.py**

```
def run(config):
    log.debug('in example_post.py - doing some global custom stuff post command...')
    home=config.getProperty('wls.oracle.home')
    if home:
        log.info('wls.oracle.home = ' + home)
```

Both these examples are plug-ins for commands that have been developed as extensions, but will also work for any of the core commands in ConfigNOW.

Plug-ins can also be defined in configuration (the environment properties file) by a plug-in point property definition as below:

`plugins.<command name>.<plug-in point>=<plug-in name>`

where <command_name> is the name of the command,<plug-in point> is the plug-in point (typically pre or post)and <plug-in name> is the name of the plug-in to execute (or a comma-separated list of plug-ins).

Defining plug-ins by configuration instead of convention allows for more than one plug-in to be executed at a certain plug-in point. The plug-ins are executed in the order in which they are defined in the comma-separated list for a plug-in point property definition.

For example, the plug-in point definition below would result in the `configure_db` plug-in being executed followed by the `configure_nodemanager` plug-in after the `create_domain` command has completed.

`plugins.create_domain.post=configure_db,configure_nodemanager`