Intel Cloud Integrity Technology 3.0

**Configuration Settings**

# Background

Mt Wilson is both an enterprise product and a datacenter product. Customers require varying degrees of automation, flexibility, and security for its configuration. This blueprint generally describes how to configure Mt Wilson and related applications.

Many applications have a file or database table with configuration settings and administrators typically prefer to edit these from the user interface or script changes in the command line.

In addition, while many settings can be represented as name-value pairs, some settings require a different format and there needs to be a way to store and access these from the user interface and the command line.

# Architecture

The environment provided by the operating system is used to bootstrap the configuration and determine the home directory, configuration directory, and features directory. Without environment settings, the application uses built-in defaults.

Using the folder locations determined by the environment or defaults, the application configuration is loaded (and decrypted if necessary) and is available to all components via a static configuration factory. The application configuration is limited to key-value pairs.

Any other type of configuration, such as a keystore file or other resource, is by its nature non-generic and belongs to a specific feature. Each feature can load its specific configuration files either from the application configuration folder or from its own feature folder. Use of the application configuration folder must be kept to a minimum.

The mtwilson-util-configuration project defines a general API for Configuration objects and related utility classes that is the preferred method for defining and sharing configuration information throughout Mt Wilson.

# Implementation

The core functionality (API and user interface) is implemented in mtwilson-core-configuration-settings-ws-v2. To add it as a dependency in Maven projects:

<dependency>

<groupId>com.intel.mtwilson.services</groupId>

<artifactId>mtwilson-core-configuration-settings-ws-v2</artifactId>

<classifier>feature</classifier>

<type>zip</type>

</dependency>

The Configuration Java API it relies on TarGzipBuilder which is provided by:

<dependency>

<groupId>com.intel.mtwilson.util</groupId>

<artifactId>mtwilson-util-configuration</artifactId>

<type>jar</type>

</dependency>

The JAX-RS endpoint is:

GET /v2/configuration-settings

Content-Type: application/json

The endpoint follows the v2 conventions.

To retrieve a setting using the command line:

mtwilson config [setting\_name]

To update a property using the command line:

mtwilson config [setting\_name] [setting\_value]

To retrieve all settings using the command line:

mtwilson export-config [outfile] [--stdout]

To import settings from another file using the command line:

mtwilson import-config [infile] [--stdin]

## Environment Variables

All use of environment variables can be found by looking for usage of java.lang.System.getenv and com.intel.mtwilson.Environment.get.

The System.getenv function can be used to retrieve all environment variables.

The Environment.get functions are used to retrieve application-specific environment variables. These are defined by a prefix. The default prefix is “MTWILSON\_” and an example variable is “MTWILSON\_HOME” which can be obtained by calling Environment.get(“HOME”). This use is encouraged because it abstracts the application name and allows the same feature to work consistently across different applications such as Mt Wilson attestation server, Key Server, Key Server Proxy, Trust Director, and Trust Agent. These applications can then define application-specific variables in accordance with industry conventions, run simultaneously on the same server, share the same launch and configuration code, yet be configured separately.

## Configuration Directory

The main configuration directory is a subdirectory named “configuration” under the application’s home directory (MTWILSON\_HOME, DIRECTOR\_HOME, etc.). The location can be changed by setting the environment variable MTWILSON\_CONFIGURATION (or DIRECTOR\_CONFIGURATION, etc). See the launcher blueprint for more information. From Java, this is accessed as Folders.configuration().

Each feature can also have its own configuration directory. From Java, this is accessed as Folders.configuration(featureId).

## Configuration File

All configuration settings that can be represented as name-value pairs are stored in a properties file in the main configuration directory.

The name of the configuration file is “mtwilson.properties” unless the application has defined a different value for “mtwilson.configuration.file” in /com/intel/mtwilson/application.properties. See the launcher blueprint for more information about the application.properties classpath resource.

The main configuration file may be encrypted. It will be loaded (and decrypted if necessary) by the launcher.

## Configuration Java API

The Configuration interface contains the following methods:

keys() returns Set<String> of all available configuration keys

get(String key) returns String with value of the requested key or null if the key is not defined

get(String key, String defaultValue) returns String with value of the requested key or defaultValue if the key is not defined

set(String key, String value) sets the value for the requested key and does not return anything

isEditable() returns boolean with true if set(key,value) should work and false if set(key,value) will throw an UnsupportedOperationException

The mtwilson-util-configuration project contains some useful decorators such as a ReadonlyConfiguration, a ValveConfiguration, a LayeredConfiguration, and a CommonsConfigurationDecorator (which provides a commons-configuration front-end to the Configuration API).

## User Interface

The mtwilson-configuration-settings-ws-v2 project contains a “Settings” tab that integrates into the mtwilson-core-html5 navigation bar to provide a page from which the user can edit the name-value pairs.

Features that require additional configuration, such as managing a set of trusted certificates, an extension point is defined for adding links to separate pages where those features provide the necessary user interface.

The extension point is /mtwilson-core-configuration/more\_settings.json with the following format:

{

"items": [

{ "label": "SAML Certificates", "target\_tab": "saml\_certificates" },

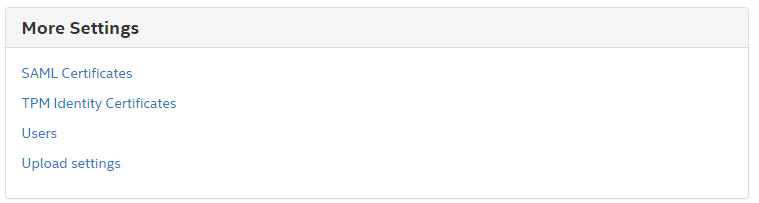
{ "label": "Upload Configuration Data Bundle", "target\_tab": "upload\_data\_bundle" }

]

}

The label is what will be shown in the user interface, and the target\_tab is the identifier of the content tab (declared in /mtwilson-core-html5/content/main.json) that should be shown when the user clicks on the label.

In the user interface, the links are sorted alphabetically. An example screenshot below (before sorting was implemented):



## Exporting Configuration to a Data Bundle

See the data bundle blueprint for more information, with an example implementation in mtwilson features/mtwilson-export-data-bundle.

## Importing Configuration from a Data Bundle

See the data bundle blueprint for more information, with upload functionality provided by mtwilson-core-data-bundle an example receiver implementation in key broker features/kms-saml.

## Security

The configuration file may be encrypted using the application’s master password. However, even with the encryption feature passwords should be stored separately in the password vault, in order to prevent leaking the passwords when an administrator prints the configuration, accesses it from the user interface, or makes a backup copy on the server.

## How to add a More Settings link for a feature

The following instructions refer to files in the feature’s source directory. Here is an example from kms-saml:

src/main/html5/mtwilson-core-configuration

+ more\_settings.json

src/main/html5/mtwilson-core-html5/content

+ main.json

+ saml\_certificates.html

+ tpm\_identity\_certificates.html

Example content of src/main/html5/mtwilson-core-configuration/more\_settings.json:

{

"items": [

{ "label": "SAML Certificates", "target\_tab": "saml\_certificates" },

{ "label": "TPM Identity Certificates", "target\_tab": "tpm\_identity\_certificates" }

]

}

Example content of src/main/html5/mtwilson-core-html5/content/main.json:

{

"items": [

{ "href": "/html5/features/kms-saml/mtwilson-core-html5/content/saml\_certificates.html", "target\_tab": "saml\_certificates" },

{ "href": "/html5/features/kms-saml/mtwilson-core-html5/content/tpm\_identity\_certificates.html", "target\_tab": "tpm\_identity\_certificates" }

]

}

And the corresponding HTML files are made the same as any other tabs that might have been registered with the navigation bar instead of with settings.

Example content of src/main/html5/mtwilson-core-html5/content/saml\_certificates.html:

<!DOCTYPE html>

<html>

<head>

<title>SAML Certificates</title>

</head>

<body>

<div class="row">

<div class="col-md-2"></div>

<div class="col-md-8" style="margin-top: 10px;">

<div class="panel panel-default" data-bind="with: samlCertificatesViewModel">

<div class="panel-heading">

<ul class="list-inline pull-right">

<li><a title="Refresh" data-toggle="modal" href="#" data-bind="click: searchSamlCertificates"><span class="glyphicon glyphicon-refresh"></span> Refresh</a></li>

<li><a title="Add SAML Certificate" data-toggle="modal" href="#addSamlCertificateModalDialog"><span class="glyphicon glyphicon-plus-sign"></span> Add SAML Certificate</a></li>

</ul>

<h1>SAML Certificates</h1>

</div>

<div class="panel-body">

<p><span data-bind="text: samlCertificates().length"></span> SAML Certificates</p>

<table class="table table-striped" data-bind="visible: samlCertificates().length > 0">

---------------------[snip]---------------------

</table>

</div>

</div>

</div>

<div class="col-md-2"></div>

</div>

---------------------[snip]---------------------

</body>

</html>

## Mt Wilson Prior Implementation

In Mt Wilson, the application configuration is available via a static function My.configuration() and is provided in a read-only view. Setup tasks or commands that need to edit the configuration can load it using the ConfigurationProvider, make changes, and then store it using the ConfigurationProvider. If this is done within the running application, the static configuration should be reloaded by calling My.reset(). All subsequent reads of My.configuration() will yield the new configuration… but because some components may cache configuration settings, the only way to ensure that all components are using new settings is by restarting the application.

# Opens