Configuration Manager Component Specification

1. Design

Software applications typically manage application level configuration details in “properties” or “ini” files. In the case of large applications or applications composed of distinct software components, there may be numerous configuration files each bound to a particular functional component.

The Configuration Manager component centralizes the management of and access to these files. The benefits of Configuration Manager include:

1) A centralized code base for reading and loading configuration files
2) A system-wide caching strategy of configuration details to limit performance bottlenecks
3) A common interface for accessing and updating application properties

The existing functionality of Configuration Manager supports the following functionality:

- Preload a list of configuration files.
- Add additional configuration files once the configuration manager has started.
- Reload configuration files into memory when properties are modified.
- Ability to refresh the Configuration Manager if configuration files are modified.
- Ability to store the properties data in .properties and .xml files..

Configuration Manager v2.1.4 extends the functionality to allow:

- Use a pluggable source of configuration data. The pluggable sources are defined in .config files that contain the name of classes representing that source and their initial parameters.
- Delete the namespaces from in-memory set of namespaces existing within Configuration Manager.
- Delete the properties and their values both from in-memory set of properties and persistent storage.
- Control the process of loading the namespaces and limit the permissions to load and modify namespaces by classes owning the namespaces only.
- Use properties nested directly or indirectly in other properties. This allows for the representation of properties as a tree with a single root and for the manipulation of sub-trees in “JNDI Context” like manner.
- Maintain the integrity of the underlying files including comments.

The Configuration Manager resolves the concurrent updates to namespace properties in following manner:

1) The temporary copy of the namespace properties should be created with ConfigManager.createTemporaryProperties() method.
2) Any modifications to properties should be made with `addProperty(), setProperty(), removeProperty(), removeValue()` and other methods.
3) When modifications are complete a `ConfigurationManager.commit()` should be invoked to store the modifications in persistent store and make them visible. This method attempts to lock the namespace before saving the properties. If namespace is already locked by another user/process the exception is thrown.

Features new in Configuration Manager 2.1.4

Nested Properties

Configuration Manager 2.1.4 provides the facility to nest properties within other properties. The properties can be organized in a tree-like structure with a single “root” property owning all properties that are in that namespace. The “root” property is a fictitious property having no name that is created with a new instance of Namespace. This “root” property serves as an entry-point to the whole property tree.

In order to reference the nested properties, compound names are supported by this version of Configuration Manager. A compound name is a dot-separated String representing the full tree path to the target property. For example, a String “`com.topcoder.util.config.ConfigManagerFile`” is a compound name of a property that should be treated as : “property `ConfigManagerFile` is nested within property `config` that is nested within property `util` that is nested within property `topcoder` that is nested within property `com`”.

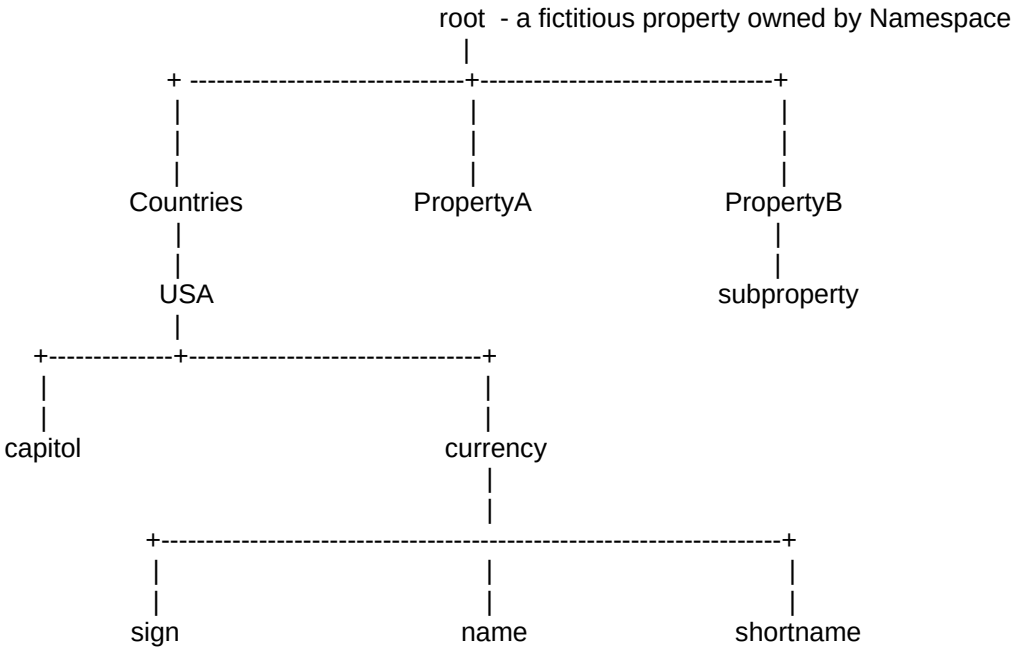
Different properties may have nested properties with same names.

The following example demonstrates a new “nested properties” concept provided by Configuration Manager 2.1.4.

- Assume the configuration file contains the following properties:

```
Countries.USA.currency.sign=$
Countries.USA.currency.name=US Dollar
Countries.USA.currency.shortname=USD
Countries.USA.capitol=Washington
PropertyA=valueA
PropertyB=valueB
PropertyB.subproperty=subproperty_value
```

- After loading the properties into a Namespace object in memory, the Configuration Manager will create following property tree:



File Integrity

Configuration Manager 2.1.4 maintains the order of underlying files (either .properties or .xml files). This includes the order of comments and properties. When loading the properties from configuration files any comments preceding the property declaration are collected in a List and are attached to the property. During the loading of properties, their values are collected in another List. This structure enables the original sequence of properties to be maintained when the properties are saved back to the persistent store.

Pluggable Sources of configuration properties

Configuration Manager 2.1.4 facilitates the use pluggable sources of configuration data. To use such source the `PluggableConfigSource` interface must be implemented. The initial parameters needed to initialize the source should be placed in some file with a filename extension equal to "`.config`". Also, a string like
"classname=<some class implementing PluggableConfigSource>".

Implementations of `PluggableConfigSource` should adhere to the following:

1) They should provide a public non-argument constructor.
2) They should accept the initial parameters from java.util.Properties provided to configure() method.

Swappable implementation of Configuration Manager

Configuration Manager 2.1.4 allows custom implementations of the Configuration Manager logic. To do so, the name of the concrete class that is derived from the ConfigManager class must be

specified as the “implementor” property within the namespace owned by Configuration Manager. If such a class is not specified, is not accessible or any other error occurs while creating an instance of this class, then an instance of DefaultConfigManager is used.

Escaping in “.properties” configuration files

Configuration Manager 2.1.4 supports escaping these characters: \#, \!, \=, \ (slash-space), \\, \:, \n, \r, \t, \uXXXX. For example, supposed there is the following item in “.properties” file:
prop\ \\\:\n\r\t\u0020=value\#\!\=
when it is parsed, key is this java string “prop \\:\n\r\t ”, value is this java string “value#!=”, note that \u0020 is parsed to a space.

1.1 Design Pattern

ConfigManager implements a Singleton pattern.

1.2 Industry Standards

XML 1.0

1.3 Required Algorithms

Most of the source code from Configuration Manager v2.0 may be reused in v2.1.4 with minor modifications.

1) Instantiating the ConfigManager

```
            if (defaultConfigManager!=null) {
                return defaultConfigManager;
            }

            defaultConfigManager = new ConfigManager();

            try {
                ResourceBundle myConfig
                = ResourceBundle.getBundle("com.topcoder.util.config.ConfigManager");

                if (myConfig  != null) {

                    Enumeration en = myConfig.getKeys();

                    while (en.hasMoreElements()) {
                        try {
                            String namespace = (String)en.nextElement();
                            String filename  = myConfig.getString(namespace);

                            if (filename.length() == 0) {
                                defaultConfigManager.add(namespace,CONFIG_XML_FORMAT);
                            } else {
                                String                          extension                          =
filename.substring(filename.lastIndexOf(".")+1);

                                if (!(new File(filename)).isAbsolute()) {
                                    URL url  =
                                defaultConfigManager.getClass().getClassLoader().getResource(filename);

                                    if(url!=null)
                                        filename=url.getFile();
                                    else
                                        throw new ConfigManagerException(
                                                "Couldn't find file " +
                                                "specified by " + filename);
                                }

                                String type=CONFIG_PROPERTIES_FORMAT;
                                if (extension.equalsIgnoreCase("xml")) {

                                //check if it's actually a multi-namespace
                                //XML file
```

```
                        Enumeration e = XMLConfigProperties.getNamespaces(
                                        filename);

                        if(e!=null && e.hasMoreElements())
                            type=CONFIG_MULTIPLE_XML_FORMAT;
                        else
                            type = CONFIG_XML_FORMAT;
                    }

                    defaultConfigManager.add(namespace,
                            filename, type);
                }
            } catch (Exception e) {
            }
        }
    }
} catch (Exception e) {
}

return defaultConfigManager;
```

2) Instantiating the PluggableConfigSource

- Create a java.util.Properties from given .config file
- Get the “classname” property
- Load the class with given name
- Create an instance of that class and cast it to PluggableConfigSource
- Configure this PluggableConfigSource with configure(Properties) method
- Load the properties with load() method.

1.4 Component Class Overview

ConfigManager

This class centralizes the management of and access to applications' configuration details. It provides a common interface for accessing and updating applications' properties. It contains a set of properties namespaces each of which contains a private set of properties.
This is a default implementation of `ConfigManager` that represents a renamed `ConfigManager` class from previous (1.2) version of Configuration Manager. This class is modified to support new features added to Configuration Manager.

Namespace

A properties namespace existing within Configuration Manager. Is described with name uniquely identifying it among other namespaces, format of source containing the properties of namespace, URL pointing to source containing the properties of namespace. All properties are separated into namespaces.

ConfigProperties
An abstract class holding the properties for single namespace. The subclasses of this class are responsible for interacting with underlying persistent storage when loading and saving namespace properties.

PluggableConfigProperties
An extension of ConfigProperties that provides the access to pluggable sources of configuration properties. Manipulates with implementations of PluggableConfigSource.

PluggableConfigSource

An interface specifying the pluggable source of configuration properties. In order to use the source of properties different from .properties and .xml files and implementation of this interface should be provided to PluggableConfigProperties class. Implementors of this interface should provide public non-argument constructor since they will be instantiated using Java Reflection API.

XMLConfigProperties
An extension of ConfigProperties that maintains the storage of properties through XML files.

PropConfigProperties
An extension of ConfigProperties that maintains the storage of properties through standard Java .properties files.

Property
An element of configuration data representing some property. A property may have single or multiple values and zero or more nested subproperties. A property is described with name uniquely identifying it among other properties.

With usage of this class all properties containing within namespace are represented as a “tree”. The Property object containing within ConfigProperties class represents a “root” of such “tree”. This class contains the methods that allow to get any existing nested properties sub-tree. This simplifies the manipulation with properties.

1.5 Component Exception Definitions

ConfigManagerException
Generic Exception thrown by the Config Manager. Thrown when errors specific to the Config Manager occurs.

DuplicatePropertyException
Thrown when there is an attempt to add a nested property to Property with name that already exists within the Property.

UnknownNamespaceException
Thrown from any of ConfigManager’s methods (except existsNamespace()) that is invoked with namespace that does not exist within ConfigManager.
ConfigParserException
Thrown when any exception preventing normal extraction of configuration data occurs. Such exceptions are thrown by subclasses of ConfigProperties during their interaction with underlying source of configuration data.

ConfigLockedException
Exception to be thrown when there is an attempt to lock a namespace but it is already locked by another user.

NamespaceAlreadyExistsException
Thrown when there is an attempt to add a Namespace which already exists in ConfigManager.

UnknownConfigFormatException
Thrown when specified format of configuration properties is not in list of valid values, that are :

- ConfigManager.CONFIG_XML_FORMAT,
- ConfigManager.CONFIG_MULTIPLE_XML_FORMAT,
- ConfigManager.CONFIG_PROPERTIES_FORMAT,
- ConfigManager.CONFIG_PLUGGABLE_FORMAT

2. Environment Requirements

2.1 Environment

2.2 TopCoder Software Components:

None.

2.3 Third Party Components:

- JAXP 1.3

3. Installation and Configuration

3.1 Package Name

com.topcoder.util.config

3.2 Configuration Parameters

com/topcoder/util/config/ConfigManager.properties

This file should be placed into directory accessible from CLASSPATH. It should contain the definitions of namespaces that should be loaded each time the Configuration Manager starts. The format of these definitions is as follows:

<namespace> = <file>, for example :

com.topcoder.util.config.ConfigManager = com/topcoder/util/config/CM.properties

3.3 Dependencies Configuration

None.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Make sure that the JAXP libraries are in the lib/endorsed directory of your JRE.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

// First an instance of ConfigManager should be obtained. This will also

// load the predefined set of namespaces and their properties into

// memory

```
ConfigManager manager = ConfigManager.getInstance();
```

// any additional namespace may be loaded then

```
manager.add("com.topcoder.currency","currency.xml",
      ConfigManager.CONFIG_XML_FORMAT);
```

```
// once the namespace properties are loaded they can be queried
String value = manager.getString("com.topcoder.currency",
       "countries.USA.currency");

// multiple values of same property are also supported
String[] values = manager.getStringArray("com.topcoder.currency",
       "contries.USA.name");

// a new properties may be defined and stored for further use

manager.createTemporaryProperties("com.topcoder.currency");
manager.setProperty("com.topcoder.currency","coutnries.Germany",
       "value");
manager.addToProperty("com.topcoder.currency",
                      "countries.USA.currency.name",
                      "US dollar");
manager.commit("com.topcoder.currency", "user");

// since Configuration Manager 2.1.4 following features are available :
// removing namespaces from in-memory set of loaded namespaces

manager.removeNamespace("com.topcoder.currency");

// checking the permission of current thread to load and modify namespaces
try {
       manager.add("namespace.owned.by.some.class",
              "somefile.xml", ConfigManager.CONFIG_XML_FORMAT);
       manager.createTemporaryProperties("namespace.owned.by.some.class")
       manager.setProperty("namespace.owned.by.some.class", "new.property",
              "value");
       manager.commit("namespace.owned.by.some.class", "user");
} catch(AccessDeniedException e) {
         System.err.println("Hey! Ask a namespace owner to load or "
                           + "modify the namespace");
}

// removing of properties
```

```
manager.createTemporaryProperties("com.topcoder.currency");
manager.removeProperty("com.topcoder.currency", "countries.Utopia");
manager.commit("com.topcoder.currency", "user");
```

// removing of values of properties

```
manager.createTemporaryProperties("com.topcoder.currency");
manager.removeValue("com.topcoder.currency",
                    "countries.USA.currency.name","US dollar");
manager.commit("com.topcoder.currency", "user");
```

// obtaining the object representation of properties tree or sub-tree

// for convenience of querying

```
Property property = manager.getPropertyObject("com.topcoder.currency",
       "countries.USA");
property.getValue("currency.name");
```

// addition of pluggable sources of configuration properties

```
manager.add("pluggable.namespace", "somefile.config",
       ConfigManager.CONFIG_PLUGGABLE_FORMAT );
```

// Demo for escaping in ".properties" files, supposed the

```
manager.add("EscapeEnhancement","test_files/TestEscape.properties",
       ConfigManager.CONFIG_PROPERTIES_FORMAT);
// string s is "hello"
String s = manager.getString("EscapeEnhancement", " \n\rProp3 ==!\t\\:!# ");
```

4.3 Demo

Included in the distribution is a /web directory. This directory contains the administration tool developed for the Configuration Manager 1.0 and 2.0. This has proven to be a useful tool in validating modifications to configuration files. The implementation of the Configuration Manager must contain the modifications to these jsp files required for it to work properly. These files will automatically be packaged in the development submission jar.

To run this tool, execute 'ant dist_web'. This will create a deployable war file that can be executed in any java servlet container.