# Configuration Management

XperienCentral provides a Configuration Management service. This service is used to store environment specific settings (hostname, for example). The settings of the Configuration Management service can be edited through the Setup Tool (`/web/setup`). This topic explains how to add a property to the Configuration Management service and how to retrieve the contents of the properties in the Java code. When creating and reading configuration entries, be sure that you conform to the plugin development guidelines (G046 and G152 in particular). When you follow the steps explained in this topic, your plugin will conform to these guidelines.

### In This Topic



---

## Adding Properties to the Configuration Management

Beginning with XperienCentral.version 10.4, it is possible to add properties to the Configuration Management through a plugin. This topic explains the steps you need to take to create two new extra properties. One extra property is an array (a list of server names). The second extra property is the `portnumber`.

### Create config_metatype.xml

Create a new file in the root of the resources folder; the new file is called `config_metatype.xml`. This file defines the properties that can be used and groups them into a set. The contents of `config_metatype.xml` should look like this in order to create support for the two new properties:

```
<?xml version="1.0"
encoding="UTF-8"?>
<metatype:MetaData xmlns:metatype="http://www.osgi.org/xmlns/metatype/v1.0.0">

        <OCD name="My Configuration Set" id="my_config_set">
                <AD name="Server names" id="my_config_entry_server_names" type="String" cardinality="-1" />
                <AD name="Portnumber" id="my_config_entry_portnumber" type="String" cardinality="1" />
        </OCD>

        <Designate pid="my_config_set">
                <Object ocdref="my_config_set" />
        </Designate>

</metatype:MetaData>
```

The cardinality can be set for each property. `Cardinality '-1'` means that the property can contain more than one value. `Cardinality '1'` means that the property can only hold one value. Be sure that the id of the configuration set (`my_config_set`) is prefixed by the plugin ID, as described in guideline G152.

### Create config.xml

While the file `config_metatype.xml` can be compared to an interface, the file config.xml can be compared to an implementation of this interface. The `config.xml` file should also be placed in the root of the resources folder. The `config.xml` file contains default values for the properties as defined in `config_metatype.xml`. The `config.xml` for the above example looks like this:

```
<configurations>
        <config name="default" definition="my_config_set" >
                <entry name="my_config_entry_server_names" value="www.domain1.com,www.domain2.com" />
                <entry name="my_config_entry_portnumber" value="8080" />
    </config>
</configurations>
```

The values for a property that can hold multiple values should be entered in a comma-separated fashion.

### Register the Configuration Set

By just creating the two XML files the Configuration Management service still doesn't know about the new properties. There are two methods in the Configuration Management service that can be called that will process the two XML files and register the new properties:

```
configurationManagement.addConfigurationSetDefinition(getClass().getResource("/config_metatype.xml"));
configurationManagement.parseAndAddConfigurationSets(getClass().getResource("/config.xml"), false);
```

The boolean used with the `parseAndAddConfigurationSets` method decides whether the values in the config.xml should overwrite the present values. If set to false, values in the Configuration Management service will not be overwritten by the values from the `config.xml`. If set to true, the values from the `config.xml` will replace the values from the Configuration Management service.

The two methods as described above should only be called when the plugin is loaded. The best place for these lines is in the `start()` method of a configuration service. Remember that this configuration set is created by the plugin and should thus be removed automatically when the plugin is purged. So the purge method should delete the configuration set.

Back to Top

---

## Using Custom Configuration Properties

This part describes in detail how to use properties that were added to the Configuration Management service. The following steps have to be taken to make the properties available:

- Create a configuration service that has access to the Configuration Management service;
- The component that needs access to the configuration properties depends on the configuration service.

### Define a Service Component in the Activator

In the Activator, a service component is defined. The service component will:

- Depend on the Configuration Management service;
- Register the configuration sets to the Configuration Management service;
- Offer getters to access the properties.

The definition of this service, the HelloWorld Configuration service (HWC service), in the Activator looks like this:

```
ServiceComponentDefinitionImpl serviceDef = new ServiceComponentDefinitionImpl( false );
serviceDef.setId("nl.gx.helloworld.configurationservice");
serviceDef.setName("HelloWorld configuration service.");
serviceDef.setDescription( "Service which contains the configuration for this plugin." );
serviceDef.setTypeId( ServiceComponentType.class.getName() );
serviceDef.setProperties( new Hashtable<String, String>() );
serviceDef.setImplementationClassName(HelloWorldConfigurationServiceImpl.class.getName() );
serviceDef.setInterfaceClassNames( new String[] {HelloWorldConfigurationService.class.getName()} );
serviceDef.setWrapperClassNames( new String[] {} );
ComponentDependencyImpl serviceDependency = new ComponentDependencyImpl();
serviceDependency.setServiceName(ConfigurationManagement.class.getName());
serviceDependency.setRequired(true);
serviceDef.setDependencies(new ComponentDependency[] { serviceDependency });
```

### Define an Interface for the HWC Service

The interface for the HWC (HelloWorld Configuration) service component is very small. The getters in the interface are based on the XML examples earlier in this topic. A basic interface looks like this:

```
public interface HelloWorldConfigurationService {
	public String[] getServerNames();
	public String getPortnumber();
}
```

## Create an Implementation of the Interface

The implementation of the interface has a dependency on the Configuration Management service so it only needs a private member of the `ConfigurationManagement` type to have access to the service. The implementation of the `HelloWorldConfigurationService` interface looks like this:

```
public final class HelloWorldConfigurationServiceImpl extends
        SimpleServiceComponent implements HelloWorldConfigurationService {

        private final static Logger LOG = Logger.getLogger(HelloWorldConfigurationServiceImpl.class.getName());

        private ConfigurationManagement myConfigService;

        /**
        * Callback method for the dependency manager, called when the bundle isstarted. This start method
registers the properties
        * to the ConfigurationManagement service.
        */

  public void onStart() {
        // Note: when the configuration set already exists, its existing properties will not be affected.
        myConfigService.addConfigurationSetDefinition(
                  getClass().getResource("/config_metatype.xml"));
        myConfigService.parseAndAddConfigurationSets(
                getClass().getResource("/config.xml"), false);
  }

        /**
        * Callback method for the dependency manager, called when the bundle is purged. This purge method
removes the configuration set
        * created in the start method.
        */

public boolean onPurge() {
        myConfigService.removeConfigurationSet("my_config_set");
    return true;
  }

public String getPortnumber() {
        try {
                return myConfigService.getParameter(
                        "my_config_set.my_config_entry_portnumber");
        }        catch (ConfigurationManagementException e) {
                LOG.log(Level.WARNING, "Could not find parameter'my_config_set.my_config_entry_portnumber' from
the ConfigurationManagement service", e);
                return null;
        }

  }

public String[] getServerNames() {
        try {
                return myConfigService.getParameters(“my_config_set.my_config_entry_server_names");
        }        catch (ConfigurationManagementException e) {
                LOG.log(Level.WARNING, "Could not find parameter'my_config_set.my_config_entry_server_names'
from the ConfigurationManagement service", e);
                return null;
                }
    }
   }
 }
}
```

## Add a Dependency to the Configuration Service

The components that want access to the properties need to create a dependency on the HWC service that was created earlier in this topic. If, for example, an element needs access to the properties, the dependency would look like this:

```
ComponentDependencyImpl servDep = new
ComponentDependencyImpl();
servDep.setServiceName(HelloWorldConfigurationService.class.getName());
servDep.setRequired(true);
elementDefinition.setDependencies(new
ComponentDependency[] { servDep });
```

Add this dependency to the definition of a component in the Activator.

## Use the Dependency in the Implementation

The dependency to the HWC service can now be used in the element implementation:

```
public class CustomElementImpl extends ElementBase
implements CustomElement {
        private final static Logger LOG = Logger.getLogger(Activator.class.getName());
    private HelloWorldConfigurationService hwcs;

    public void setHelloWorldConfigurationService(
        HelloWorldConfigurationService hwcs) {
                this.hwcs = hwcs;
        String ServerNames = "";
        for (String ServerName : hwcs.getServerNames()) {
                        ServerNames += ServerName + ";";
                }
        LOG.log(Level.INFO, "Values of the custom properties are:\n Server names:" + ServerNames + "\n
Portnumber : " + cs.getPortnumber());
                }
}
```

The above code will result in a log message each time an instance of the element is being placed or refreshed on a page:

```
INFO: Values of the custom properties are:
  Server names:
www.domain1.com;www.domain2.com; Portnumber  : 8080
```

## Changing the Values of the Properties

Once the properties are present in the Configuration Management service, the value of the properties can be changed through the /web/setup tool. Log onto the Setup Tool (`/web/setup`) and navigate to the General configuration tab. Your configuration set should be present between the default configuration sets (the sets are alphabetically ordered).

Back to Top