Brand Mobiliser State Developer's Guide

[PRODUCT DOCUMENTATION]

www.sybase.com

Contents:

Principal Author

Sybase365 - mCommerce

Revision History

Version 1.1 - March 2011

1 Introduction

Brand Mobiliser makes it very easy for companies to mobilise all aspects of their businesses, including: brand awareness, CRM, mobile banking and financial, mobile payment, commerce, and much more. Brand Mobiliser fulfills the increasing demand of mobile customers for more interactions with their mobile applications.

For partners, Brand Mobiliser provides a plug-in mechanism to develop components to enrich the Mobile Interactive Application experiences even more, allowing integration of any 3rd party systems.

This document describes the process for creating those plug-ins.

It is expected that the reader is familiar with the Brand Mobiliser operational environment, the definition of states and applications and is familiar with the web interface associated with creating interactive applications.

It is also assumed that the reader has good knowledge of Java development and the use of development utilities such as Maven and an integrated development environment, such as Eclipse[1] or NetBeans[2] (although there is no pre-requisite on using a specific IDE).

In particular, this document describes the following:

- The public states API and the class library from which custom states will inherit.
- Some sample and example states.
- The process for building a state plugin bundle.
- How to install and configure a plugin bundle into a Brand Mobiliser installation.

Brand Mobiliser is based on a processing core using an OSGi[3] model. This allows packages of Java classes to be separated and dependencies to be carefully described and dynamically managed. It is expected that the reader is at least partly familiar with the concepts of OSGi. However, this document describes all steps necessary to follow in order to develop, install and run a state plugin as an OSGi bundle.

1.1 References

1. Brand Mobiliser User Manual; Version 1.1 – March 2011
2. Brand Mobiliser Development Manual; Version 1.1 – March 2011

[1] http://www.eclipse.org/downloads/

[2] http://netbeans.org/

[3] http://www.osgi.org/Main/HomePage

2 The States API

This section describes the API that should be used to developed new states.

2.1 State Plugin Components

The following diagram outlines the basic component sets that are used to develop Brand Mobiliser state plugins. Each component set relates one-to-one to an OSGi bundle that is used to separate out different aspects of processing. It is expected that sets of states are packaged into separate OSGi bundles.

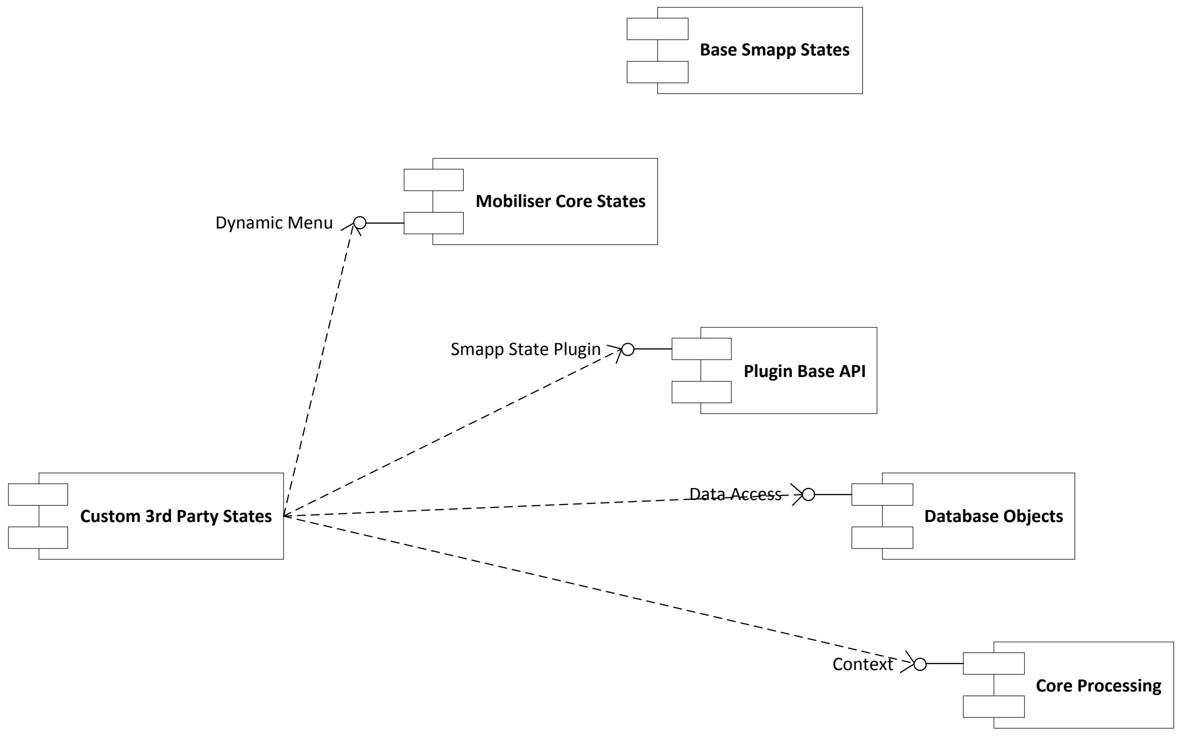


Figure 1. Components For 3rd Party State Development

The components are described as follows:

- Plugin Base API – This bundle contains the packages that form the processing base for any state plugin. All state plugins inherit and use classes from this component. The following are the important packages in this bundle;

  at.ip2.mwiz.processing.plugins.smapp – for the base state API interface

  at.ip2.mwiz.processing.plugins.smapp.state – basic implementation state class to inherit from

  at.ip2.mwiz.processing.plugins.smapp.controls – holds the state attribute classes

  at.ip2.mwiz.processing.plugins.smapp.useful – some utility classes

  Maven:

  <groupId>com.sybase365.mobiliser.brand.plugins</groupId>

  <artifactId>mobiliser-brand-plugin-base</artifactId>

  <name>AIMS :: Object :: Brand Mobiliser Plugin Base API</name>

Filename: mobiliser-brand-plugin-base-*version*.jar

- Database Objects – This bundle holds the reference objects that represent data returned from and stored to the database. The data represent the state's configurations which are defined during the application creation using the application composer or editor.

  Maven:
  <groupId>com.sybase365.mobiliser.brand.database</groupId>
  <artifactId>mobiliser-brand-database</artifactId>
  <name>AIMS :: Object :: Brand Mobiliser Database Objects</name>

  Filename: mobiliser-brand-database-*version*.jar

- Core Object – This bundle provides references to the base processing capability and reference objects. The Core Object is shown as Core Processing in the figure above.

  Maven:
  <groupId>com.sybase365.mobiliser.brand.core</groupId>
  <artifactId>mobiliser-brand-core</artifactId>
  <name>AIMS :: Object :: Brand Mobiliser Core Objects</name>

  Filename: mobiliser-brand-core-*version*.jar

- Base Smapp States – This bundle contains the basic Brand Mobiliser state implementations, such as Send SMS.

  Maven:
  <groupId>com.sybase365.mobiliser.brand.plugins</groupId>
  <artifactId>mobiliser-brand-plugin-smapp</artifactId>
  <name>AIMS :: Object :: Brand Mobiliser Plugin - Base Smapp States</name>

  Filename: mobiliser-brand-plugin-smapp-*version*.jar

- Mobiliser Core States – This bundle contains a set of utility Mobiliser states and the specific states that integrate with the Money Mobiliser Web Services.

  Maven:
  <groupId>com.sybase365.mobiliser.brand.plugins.mobiliser</groupId>
  <artifactId>mobiliser-brand-plugin-core</artifactId>
  <name>AIMS :: Object :: Brand Mobiliser Plugin - Mobiliser Core States</name>

  Filename: mobiliser-brand-plugin-core-*version*.jar

- Custom 3rd Party States – This is the bundle you will implement (the actual name is entirely up to you). The only dependencies you need to reference are the Plugin Base API, the Database Objects and the Core Processing components. Optionally, depending on what your state needs to do, you may inherit from the Mobiliser Core States component; for example, in order to use it's Dynamic Menu capability.

2.2 *State Plugin Class Hierarchy*

The figure below shows the basic state plugin class hierarchy from which all states are built.

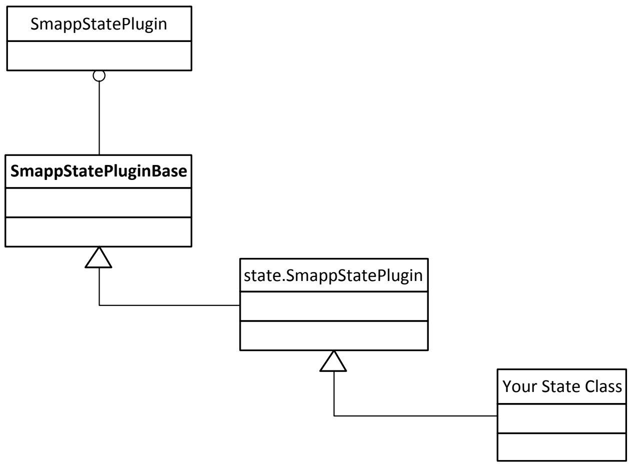


Figure 2. Components For 3rd Party State Development

The following code blocks show the important methods from the class hierarchy.

2.2.1 at.ip2.mwiz.processing.plugins.smapp.SmappStatePlugin

The block below shows selected methods from the state plugin interface from which all plugins will implement (indirectly through the other classes).[4]

at.ip2.mwiz.processing.plugins.smapp.SmappStatePlugin.java:

```
public interface SmappStatePlugin extends PluginInterface {

    String getRevisionString();  (from PluginInterface)

    long getStateId();

    String getStateName();

    boolean isSelectable();

    String getStateNotes();

    @Deprecated
    void webEditorIwfInit(SmappNodeInterface node);

    @Deprecated
    void webEditorLoadValues(SmappNodeInterface node,
```

[4] This interface is used to identify any class as an OSGi state plugin service.

```
                              SmappStateExtended smappState,
                              SmappStateLang smappStateLang)

             throws DBException;

    @Deprecated
    void ~~webEditorSaveValues~~(SmappNodeInterface node,
                              SmappStateExtended smappState,
                              SmappStateLang smappStateLang);

    @Deprecated
    void ~~webEditorModifyChildNodeRow~~(DataGridRow dgr, SmappTransitionExtended st);
...

    public boolean supportsFailTransition();

    public boolean supportsOkTransition();

    public boolean supportsSendSmsMessage();

    public boolean supportsGoToApplication();

...
```

The following describes each different method and what it is used for.

Methods

`String getRevisionString();`

This method is required from PluginInterface. It is used to identify a version and/or change number of the plugin. This method can return any arbitrary String value.

`long getStateId();`

This method returns a unique identification for the state. Every state must provide a unique value to the Brand Mobiliser system, which is stored into the database for each installation where that state is used. This unique value allows the state to be resolved to the same type across installations.

The values between 0 to 100,000 are reserved for Brand Mobiliser and Sybase 365 use.

Partners and other developers must use a unique value above 100,000 for each state they develop.

Note: Do not use a value equal or greater than 1,000,000.

Typically, Money Mobiliser state plugins have allocated values greater than 400,000 and less than 400,999.

`String getStateName();`

This method returns the state name and is what is displayed on the Application Composer and Editor web user interface pages. It is recommended to be unique and may also reference the 3rd party system it integrates with, e.g. 'mBanking Login'.

`boolean isSelectable();`

This method determines if the state is shown in the web user interface drop-down list box for selecting follow-up states.

Deprecated Web Editor Methods

```
void webEditorIwfInit(...);
void webEditorLoadValues(...) throws ...;
void webEditorSaveValues(...);
void webEditorModifyChildNodeRow(...);
```

These methods provide integration into the web user interface for state specific configuration.

Note: These methods are deprecated and should not be overridden. The relate to old web user interface (as shown in the Application Editor web page). The following methods provide state user configuration through a standard meta-data driven interface to separate state UI configuration from its processing.

New Meta-Data Methods

```
String getStateNotes();
```

This is used to define an interface method for the state help text, or short text description of the state.

```
public boolean supportsFailTransition();
```

This method returns true or false value that specifies if the state requires the configuration of an optional Fail transition. The state plugin code will determine if a transition to a 'Fail' state is necessary or not.

```
public boolean supportsOkTransition();
```

This method returns true or false value that specifies if the state requires the configuration of an optional Success, or Ok, transition. The state plugin code will determine if a transition to a 'Success' state is necessary or not.

```
public boolean supportsSendSmsMessage();
```

This method returns true or false value that specifies if the state requires the configuration a text message that will be used as part of the state's processing.

```
public boolean supportsGoToApplication();
```

This method returns true or false value that specifies if the state requires selection of an associated application.

2.2.2 at.ip2.mwiz.processing.plugins.smapp.SmappStatePluginBase

The block below shows selected methods from the abstract state plugin base implementation from which all plugins will inherit.

at.ip2.mwiz.processing.plugins.smapp.SmappStatePluginBase.java:

```
public interface SmappStatePluginBase extends Plugin implements SmappStatePlugin{

    protected SmappStateExtended determineFollowingSmappStateFromPattern
    ( ... ) throws ... { ... }

    protected SmappStateExtended determineFollowingSmappStateFromTransitionType
    ( ... ) throws ... { ... }

...
```

Methods

```
protected SmappStateExtended determineFollowingSmappStateFromPattern
               ( ... ) throws ... { ... }
protected SmappStateExtended determineFollowingSmappStateFromTransitionType
               ( ... ) throws ... { ... }
```

These methods allow the state to decide what is the follow-up state to flow-into.

Note: It is not recommended to use these methods directly. It is expected all custom states will use the helper methods named 'continueXYZ' as described in the next section.

2.2.3 at.ip2.mwiz.processing.plugins.smapp.state.SmappStatePlugin

This class forms the last part of the Brand Mobiliser class hierarchy for states and identifies the point of implementation for all state plugins.

at.ip2.mwiz.processing.plugins.smapp.state.SmappStatePlugin.java:

```
public abstract class SmappStatePlugin extends SmappStatePluginBase {
  protected abstract Attribute[] getStateAttributes();
  protected abstract SmappStateExtended processStateAttr(
      SmappStateProcessingAction action)
      throws MwizProcessingException,DBException, JAXBException,
      IOException, ServiceException, RequiredParameterMissingException;
  @Deprecated
  protected String ~~getStateInfoText~~() { ... }
  public String getStateNotes() { ... }
  public SmappStateProcessingContext getContext() { ... }
  public SmappStateProcessingAction getAction() { ... }
  public SmappStateExtended continueOk() throws DBException { ... }
  public SmappStateExtended continueFail() throws DBException { ... }
  public SmappStateExtended continueFail(String msg) throws DBException { ... }
  public SmappStateExtended continueDyn(String val) throws DBException { ... }
  public SmappStateExtended continueDyn(Integer val) throws DBException { ... }
  public SmappStateExtended continueDyn(Long val) throws DBException { ... }
...
```

The following describes each different method and what it is used for.

Abstract Methods:

`protected abstract Attribute[] getStateAttributes();`

This abstract method must be implemented to return a list of expected input and output attributes that the state will use and/or set during its processing. It is expected that most states will require input beyond the basic customer information that is provided by Brand Mobiliser (e.g. the current user's MSISDN). See the section below for information on how to define required attributes in your state class.

`protected abstract SmappStateExtended processStateAttr(...) throws ...;`

This abstract method should implement the processing of the state. It is called when the application is at the point where this state should be invoked. All input parameters will be resolved and made available through context and attribute API (see below).

On exit of this method the state must supply an SmappStateExtended object that represents the next state depending on the processing that has occurred. The simplest and recommended way of doing this is to use the continueXYZ() methods described below. This allows the state processing to decide if the state is successful, has failed or can continue based on transition using a dynamic value. The configuration of the state is then automatically referenced to find the next state (SmappStateExtended object) and that is returned to the state processing engine.

Deprecated Methods

`protected String **getStateInfoText**() { ... }`

This method returns a description of the state, which will be used in the administration web user interface as help text to aid in the use and configuration of the state. HTML tags may be embedded into the returned String, to allow for formatting to be applied to the text (typical tags would be <p> and <br/>). Please ensure the returned HTML formatted fragment is well-formed.

Note: This method is now deprecated and is replaced by the meta-data method described below, with non-specific formatting information.

New Meta-Data Methods

```
public String getStateNotes() { ... }
```

This method returns the equivalent of what was being returned in the deprecated method above. However, any HTML tags will have to be removed as they will not have any visual affect and will be output as literal strings.

To insert a line break in the text string use the newline character, e.g. "Text.\nText on a newline."

To insert a paragraph break in the text string, use two newline characters consecutively; e.g. "Text.\n\nText in a new paragraph."

Public Methods

```
public SmappStateProcessingContext getContext() { ... }
```

This method is a helper method to access the context associated with the processing of the state. The context identifies information about the current session, including customer MISDN and language, the current state being processed, and associated general information, like the message and message receiver queue used for the application.

In particular it also allows access to a context variable for performing database changes, named 'dao'. This is particularly useful for storing session-based attribute values back to the database, for example;

```
...
  SessionAttribute sessionAtt =
      getContext().dao.getSessionAttributeForKey(
            inIndex.getText(), this.getContext().session);
  if (sessionAtt == null) {
    getContext().dao.saveSessionAttribute(
       inIndex.getText(), "0", getContext().session);
  }
  int index = inIndex.getValue().getInt();
  index++;
  getContext().dao.saveSessionAttribute(
    inIndex.getText(), String.valueOf(index), getContext().session);
...
```

```
public SmappStateProcessingAction getAction() { ... }
```

This method access the action of the state. It is not expected that custom states will use this method.

```
public SmappStateExtended continueOk() throws DBException { ... }
public SmappStateExtended continueFail() throws DBException { ... }
public SmappStateExtended continueFail(String msg) throws DBException { ... }
public SmappStateExtended continueDyn(String val) throws DBException { ... }
public SmappStateExtended continueDyn(Integer val) throws DBException { ... }
public SmappStateExtended continueDyn(Long val) throws DBException { ... }
```

These methods are used as helper methods in sub-classes and custom states to allow the state processing to indicate success, failure or invocation of a follow-up state based on a dynamic value.

If ‘continueOk’ is called, the state associated with the ‘OK’ transition is used as follow-up state.

If ‘continueFail’ is called, the state associated with the ‘Fail’ transition is used as follow-up state.

If ‘continueDyn’ is called, the value passed into the method is evaluated against all keyword/expression transitions specified for the state, to find a matching follow-up state.

Note: If your state code utilitises the ‘continueOk’ and/or ‘continueFail’ methods, your state code must also overide the ‘supportsOkTransition’ and/or ‘supportsFailTransition’ methods and return a boolean true value. It is not required to indicate a dynamic transition is used, as these are optionally available for all state types.

2.3 State Attributes

This section describes how to define input and output attributes and how those attribute values can be accessed or set at runtime.

The diagram below outlines the class hierarchy of state attributes.

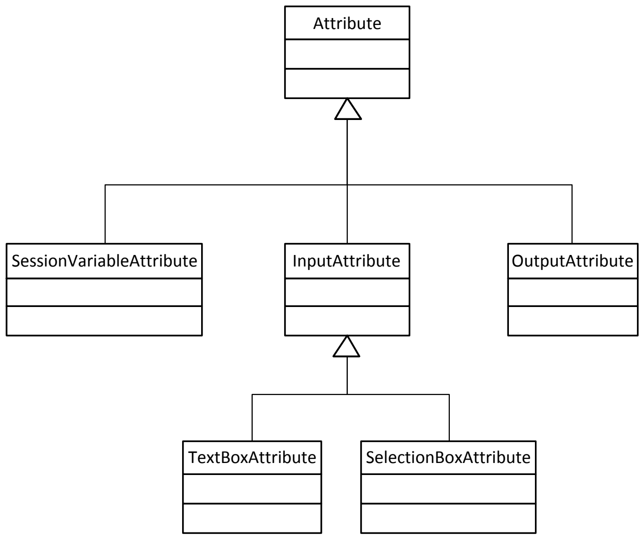


Figure 3. Components For 3rd Party State Development

All attributes are identified by the properties;

- An identifier name - the attribute variable name; this defaults to the session variable name.
- A short description - displayed on the web user interface as a short description of the attribute.

2.3.1 Input Attributes

In addition to the base attribute properties, input attributes also identify an extra property;

- Mandatory value – a true or false value to indicate if the attribute value is mandatory or optional.

Note: On configuration of the input attribute for a state, the administrator has the option to provide either a fixed constant value for the attribute or the value is accessed from a session variable. When the

input variable is mandatory and the attribute value is accessed from a session variable, a run-time error will be thrown if a value for the configured session variable doesn't exist.

As shown in the figure above, there are two different input attribute types:

- TextBoxAttribute

  This provides for the simple case where the input attribute is a single constant value, or a single value accessed from a session variable.
- SelectionBoxAttribute

  This provides for the case where the input is a constant value selected from a list of options. The values for the list is populated in the state code.

To access input attributes use the following methods from InputAttribute class;

Methods

```
public boolean isSet() throws ...
```

Use this method to test if the input attribute is available or not. If the input attribute is a TextBoxAttribute, a constant value or a non-empty session variable value is checked for. If the input attribute is a SelectionBoxAttribute then a choice must have been made from the available options.

```
public InputValue getValue() throws ...;
```

This method returns the input variable through an accessor class. Use the following methods to get the value of a specific type;

```
InputValue.getString();
InputValue.getLong();
InputValue.getInt();
InputValue.getBoolean();
InputValue.getMsisdn();
InputValue.getPhoneNumber();
```

```
public String getText() throws ...;
```

This method returns the value entered by the administrator for the configuration of this input attribute. This will either be a constant value, or the name of a session variable.

Note: You do not normally have to use this method, because the getValue() method will use the getText() method in the correct context (constant or variable), but it maybe useful in some circumstances.

Creating New Input Attribute Instances

The following is an example of creating new input attribute variable instances;

```
...
  protected static final TextBoxAttribute inIndex =
      new TextBoxAttribute("INDEX", "Variable name", false);

  private static Attribute[] stateAttr;

  static {

    stateAttr = new Attribute[]{inIndex};

  }

...
  @Override

  protected Attribute[] getStateAttributes() {

    return stateAttr;
  }
...
```

2.3.2 Output Attributes

Output attributes are session variables whose values are only set by the state and do not require to be set on entry. The output attribute is saved into a session variable as named in the state configuration (default is the output attribute identification name).

To set output attribute value use the following methods from OutputAttribute class;

Methods

```
public void setValue(String val) throws ...
public void setValue(Long val) throws ...
public void setValue(Integer val) throws ...
public void setValue(Boolean val) throws ...
```

Use these methods to set a value of the appropriate type.

```
public String getText() throws ...;
```

This method returns the value entered by the administrator for the configuration of this output attribute. This will be the name of a session variable.

Note: You do not normally have to use this method, because the setValue() method will use the getText() method itself, but it may be useful in some circumstances.

Creating New Output Attribute Instances

The following is an example of creating a new output attribute variable instance;

```
...
  protected static final OutputAttribute outValue =
      new OutputAttribute("DEST_VAR", "Destination");

  private static Attribute[] stateAttr;

  static {
    stateAttr = new Attribute[]{outValue};
  }
...
  @Override
  protected Attribute[] getStateAttributes() {
    return stateAttr;
  }
...
```

2.3.3 Session Variable Attributes

Session variable attributes are a specialised type of both input and output attribute that provides for getting and setting of a list of values, as identified in a java.util.List object.

The List entry objects must implement the BeanConverterInterface. This interface provides the conversion logic for the List entry from an Object into a String and vice versa;

```
package at.ip2.mwiz.processing.plugins.smapp.beans;
```

```
public interface BeanConverterInterface<T> {
  T convert(String value);
  String convert(T object);
}
```

Methods

`public <T extends BeanConverterInterface<T>> void setList(List<T> list) throws ..`

Use this method to set the List of beans into the session variable.

`public <T> List<T> getList(BeanConverterInterface<T> converter) throws ...`

Use this method to get the List of beans from the this session variable using the converter object to change from String value to an List Object value.

Creating New Session Attribute Instances

The following is an example of creating a new session attribute variable instance;

```
...
  protected static final SessionVariableAttribute walletList =
      new SessionVariableAttribute("SESSION_WALLET_LIST", "");
  private static Attribute[] stateAttr;
  static {
    stateAttr = new Attribute[]{inIdentType, inIdent, inFilterPiType,
         inFilterPiClass, inFilterMaxPis, outCurrency,
         outPiType, outPiClass, walletList, outPiStatus};
  }
...
  @Override
  protected Attribute[] getStateAttributes() {
    return stateAttr;
  }
...
```

2.4 Example State

This section describes an example states.

2.4.1 Mobiliser Counter

The following is the code for the Mobiliser Counter example. This class extends the base API class SmappStatePlugin. The code block is annotated with numbers, further information for which is shown below.

SmappStatePbxCounter.java:

```
package at.ip2.mwiz.processing.plugins.implementation.mobiliser;
import java.io.IOException;
import javax.xml.bind.JAXBException;
import javax.xml.rpc.ServiceException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import at.ip2.idf.DBException;
import at.ip2.mwiz.CryptoException;
import at.ip2.mwiz.db.objects.SessionAttribute;
import at.ip2.mwiz.db.objects.smapp.SmappStateExtended;
import at.ip2.mwiz.processing.exceptions.MwizProcessingException;
import at.ip2.mwiz.processing.plugins.smapp.controls.Attribute;
import at.ip2.mwiz.processing.plugins.smapp.controls.TextBoxAttribute;
import at.ip2.mwiz.processing.plugins.smapp.state.RequiredParameterMissingException;
import at.ip2.mwiz.processing.plugins.smapp.state.SmappStatePlugin;
import at.ip2.mwiz.processing.plugins.smapp.SmappNodeInterface;
import at.ip2.mwiz.processing.plugins.smapp.SmappStateProcessingAction;

public class SmappStatePbxCounter extends SmappStatePlugin {

  protected static final Logger LOG = LoggerFactory.getLogger(SmappStatePbxCounter.class);

  protected static final TextBoxAttribute inIndex =
        new TextBoxAttribute("INDEX", "Variable name", false);

  private static Attribute[] stateAttr;                                1

  static {
    stateAttr = new Attribute[]{inIndex};
  }

  @Override
  protected String getStateNotes() {
    return "A counter state. If executed, it increases the given variable by 1.\n"
      + "The state automatically initiates a new variable with 0 and adds 1\n"
      + "Use the following follow up states:\n"
      + "- OK: Not in use.\n"
      + "- FAIL: If an error occurs.\n"
      + "- Dyn: The current index count.";                           2
  }

  @Deprecated
  protected String ~~getStateInfoText~~() {
    return "A counter state. If executed, it increases the given variable by 1. <br />"
      + "The state automatically initiates a new variable with 0 and adds 1<br />"
      + "<b>Use the following follow up states:</b><br />"
      + "- FAIL: If an error occurs.<br />"
      + "- OK: Not in use.<br />"
      + "- Dyn: The current index count.";
  }

  @Override
  public String getRevisionString() {
    return "1.0.0-SNAPSHOT";
  }

  @Override
  protected Attribute[] getStateAttributes() {
    return stateAttr;                                        3
  }

  @Override
  public long getStateId() {
    return SmappStateMobiliserBase.STATE_COUNTER;            4
  }

  @Deprecated
  public void ~~webEditorIwfInit~~(SmappNodeInterface node) {

    super.webEditorIwfInit(node);
    node.getPluginControlsObject().containerResultTransitionOk.setVisible(false);

  }

  public boolean supportsOkTransition() {
    return false;                                            5
  }

  public boolean supportsFailTransition() {
    return true;
  }
  @Override
  public String getStateName() {
    return "Mobiliser Counter";             6
  }

  @Override
  protected SmappStateExtended processStateAttr(
      SmappStateProcessingAction action) throws MwizProcessingException,
      DBException, JAXBException, IOException, ServiceException,
      RequiredParameterMissingException {

    SessionAttribute sessionAtt =
      getContext().dao.getSessionAttributeForKey(inIndex.getText(),
        this.getContext().session);
                                                                     7
    try {
      if (sessionAtt == null) {

        getContext().dao.saveSessionAttribute(inIndex.getText(), "0",
            getContext().session);

      }
      int index = inIndex.getValue().getInt();
      index++;

      getContext().dao.saveSessionAttribute(
        inIndex.getText(),
        String.valueOf(index),
        getContext().session);

      return continueDyn(index);                   8
    }
    catch (CryptoException ce) {
      log.error("CryptoException!", ce);
    }
    catch (Exception e) {
      log.error("Exception!", e);
    }
    return continueFail();                  9
  }

}
```

1 Create the mandatory input attribute named ‘INDEX’.

2 Return the ‘About’ text for this state, using the new meta-data method ‘getStateNotes()’.

3 Return all the attributes.

4 Return the unique state ID (referenced in another class, with all other states for this bundle).

5 Override the meta-data methods ‘supportsOkTransition’ and ‘supportsFailTransition’. In this case ‘Ok’ is not used and ‘Fail’ is used.

6 Return a unique descriptive state name.

7 Implement the logic of the state processing.
This state takes the configured text of the input attribute ‘INDEX’ and uses that to lookup a session variable with the text name.

It then gets the integer value increments it by one, then uses the processing context to retrieve a data access object to save the value back to the database to be persisted for this session.

8 Continue using the index value as the dynamic transition follow-up state.

9 Continue using the ‘Fail’ transition follow-up state.

3 Building State Plugin Bundles

The process of building a state plugin requires the conformance to the standard software development processes that are used in Brand Mobiliser.

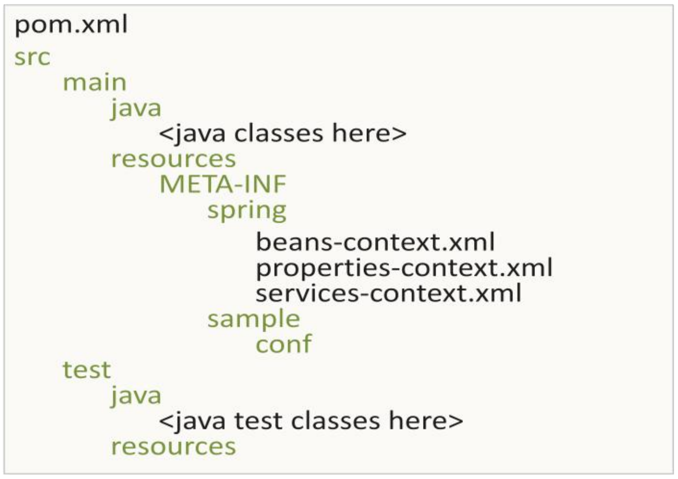
In particular, the build process uses **Apache Maven** as the software project management and build process. It is assumed that the reader if familiar with the basic concepts of Maven.

Note: It is also assumed that the state developer has access to a Maven artifact repository containing the Brand Mobiliser dependencies. If you require access to the Sybase 365 repository or you are missing transient dependencies, please contact Sybase 365 mCommerce support for more information.

Brand Mobiliser also uses the following major software mechanisms;

- The Spring Framework - for application context and dependency injection.
- Spring Dynamic Modules (Spring DM) – for abstraction of OSGi mechanisms.
- OSGi Services – for simple software service publication and consumption.
- OSGi Configuration Admin – for container based configuration of services and components.

The figure below shows both the directory and file structure for a typical Brand Mobiliser plugin implemented as a Maven software module.



```
pom.xml
src
    main
        java
            <java classes here>
        resources
            META-INF
                spring
                    beans-context.xml
                    properties-context.xml
                    services-context.xml
                sample
                    conf
    test
        java
            <java test classes here>
        resources
```

Figure 4. State Plugin Project Directory Structure

The following sections describe the use of the major software mechanisms and the required files as listed above.

3.1 *Maven Project File*

The Maven project file (pom.xml) contains all the required information for the Maven tool to build and create a Maven artifact that can be installed into Brand Mobiliser. There are specific requirements on the contents, so that an OSGi compliant bundle is configured correctly for the artifact.

The sample project file below shows a simple example, one that will be used to describe the build process in this section.

pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.sybase365.mobiliser.brand.poms</groupId>
    <artifactId>spring-parent</artifactId>
    <version>1.0.4</version>
  </parent>

  <groupId>com.sybase365.mobiliser.brand.plugins.sample</groupId>
  <artifactId>mobiliser-brand-plugin-sample</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>bundle</packaging>
  <name>AIMS :: Object :: Brand Mobiliser Plugin - Sample State</name>      1

  <properties>

    <bundle.namespace>${pom.groupId}</bundle.namespace>
    <bundle.symbolicName>${bundle.namespace}.${pom.artifactId}</bundle.symbolicName>

  </properties>
                                                                                 2
  <build>
    <plugins>
      <!-- Create an OSGi Bundle Manifest -->
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>      3
        <extensions>true</extensions>
        <configuration>
          <instructions>
            <manifestLocation>META-INF</manifestLocation>
            <Bundle-Category>object</Bundle-Category>
            <Bundle-SymbolicName>${bundle.symbolicName}</Bundle-SymbolicName>
            <Bundle-Version>${pom.version}</Bundle-Version>
            <Embed-Dependency>
              <!--Enter here artifact id of dependencies to
be packaged inside this artifact -->
            </Embed-Dependency>
            <Export-Package>
              <!--Enter here java packages that you may want to be made available to other
bundles -->
            </Export-Package>
            <!--
             | Note: When you develop your own additional classes within this object
bundle,
             include the package names of the classes in either the Export-Package or the
            Private-Package, otherwise it will not be included in the bundle.
            -->
            <Private-Package>
               com.sybase365.mobiliser.brand.plugins.sample.impl
            </Private-Package>
            <DynamicImport-Package>
              *
              ,javax.xml.bind
            </DynamicImport-Package>
            <!--
             | Note: If you use other only referenced from spring context then include them
in
            the Import-Package instruction here.  The * instruction will ensure any
directly
            imported packages in supporting classes are include automatically, but Spring
            context referenced ones need explicit reference.
            -->
            <Import-Package>
              ,at.ip2.mwiz
              ,at.ip2.mwiz.objects
              ,at.ip2.mwiz.processing
              ,at.ip2.mwiz.processing.exceptions
              ,at.ip2.mwiz.processing.plugins.charging
              ,at.ip2.mwiz.processing.plugins.smapp
              ,at.ip2.mwiz.processing.plugins.smapp.beans
              ,at.ip2.mwiz.processing.plugins.smapp.controls
              ,at.ip2.mwiz.processing.plugins.smapp.state
              ,at.ip2.mwiz.processing.plugins.useful
              ,at.ip2.mwiz.db
              ,at.ip2.mwiz.db.implementation
              ,at.ip2.mwiz.db.objects
              ,at.ip2.mwiz.db.objects.campaigns
              ,at.ip2.mwiz.db.objects.smapp
              ,at.ip2.mwiz.db.enums
              ,at.ip2.idf
              ,at.ip2.iwf
              ,at.ip2.iwf.webcontrols
              ,org.osgi.service.cm;resolution:=mandatory
                ,org.apache.commons.logging
              ,org.slf4j;version="[1.5.6,1.5.6]"
              ,org.springframework.beans.factory.annotation
            </Import-Package>
            <!--
             | Each module can override these defaults in an optional osgi.bnd file
            -->
            <_include>-osgi.bnd</_include>
            <ARF-Bundle-Template>/META-INF/sample/config</ARF-Bundle-Template>
          </instructions>

          <!-- Generate a local repository.xml for the artifact to enable obr testing -->
          <obrRepository>${project.build.directory}/repository.xml</obrRepository>

        </configuration>

      </plugin>

    </plugins>

  </build>


  <dependencies>

    <dependency>
      <groupId>com.sybase365.mobiliser.brand.core</groupId>        4
      <artifactId>mobiliser-brand-core</artifactId>
      <version>1.1.0</version>
    </dependency>

    <dependency>
      <groupId>com.sybase365.mobiliser.brand.database</groupId>
      <artifactId>mobiliser-brand-database</artifactId>
      <version>1.1.0</version>
    </dependency>

    <dependency>
      <groupId>com.sybase365.mobiliser.brand.plugins</groupId>
      <artifactId>mobiliser-brand-plugin-base</artifactId>
      <version>1.1.0 </version>
    </dependency>

    <dependency>
      <groupId>javax.xml.rpc</groupId>
      <artifactId>com.springsource.javax.xml.rpc</artifactId>
      <version>1.1.0</version>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>org.junit</groupId>
```

```
      <artifactId>com.springsource.org.junit</artifactId>
      <version>4.4.0</version>
      <scope>provided</scope>
    </dependency>

  </dependencies>

</project>
```

1 This project file inherits from a parent, which is the standard Brand Mobiliser Spring-based parent module. This parent specifies the appropriate versions for transient dependencies.

Note: This project file identifies the package type as '*bundle*'. This means that an OSGi compliant bundle is to be built and the 'maven-bundle-plugin' must be used to do this.

2 The properties section identifies fixed values that are used in this project file.

3 This section contains the configuration of the OSGi bundle that is to be created.

Note: The configuration of this section is critical to the correct building, installation and running of the sample plugin. This is described in more detail in a section below.

4 This section contains the standard dependency section of a Maven project file. This list contains the core minimum – it is expected you may have additional dependencies or artifacts that will be referenced.

3.1.1 OSGi Bundle Configuration

To create a correct OSGi bundle you need to specify the basic requirements of the processing.

The Brand Mobiliser OSGi runtime provides the wiring of bundles to other bundles. OSGi provides an compartmentalized sets of JAR files that are dynamically made available to other bundles (JAR files) on request.

The extract of the Maven project file below shows the important sections of the 'maven-bundle-plugin' and are described more below.

pom.xml:

```
...
  <build>
    <plugins>
      <!-- Create an OSGi Bundle Manifest -->
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <extensions>true</extensions>
        <configuration>
          <instructions>
            <manifestLocation>META-INF</manifestLocation>
            <Bundle-Category>object</Bundle-Category>
            <Bundle-SymbolicName>${bundle.symbolicName}</Bundle-SymbolicName>
            <Bundle-Version>${pom.version}</Bundle-Version>
            <Embed-Dependency>
            <!-- Enter here artifact id of dependencies to be packaged inside
                this artifact ->
            </Embed-Dependency>
            <Export-Package>
            <!-- Enter here java packages that you want to be made available to
                other bundles -->
            </Export-Package>
            <Private-Package>
            <!-- Enter here java packages that are private to this bundle only -->
            </Private-Package>
            <DynamicImport-Package>
            <!-- Enter here java packages that are private to this bundle only -->
            </DynamicImport-Package>
            <Import-Package>
            <!-- Enter here java packages that are external and required by this bundle -->
            </Import-Package>
          </instructions>

        </configuration>

      </plugin>
    </plugins>
  </build>

...
```

Bundle Symbolic Name and Version

The bundle symbolic name and version is used by other bundles when referencing this bundle. It is also shown in the configuration/debugging output to aid identification of the bundle.

Embed Dependency

When your bundle requires the use of another JAR package, you have the option to resolve the JAR package from the OSGi environment or to embed the JAR package as a local-only dependency.

For Brand Mobiliser state plugin development, all references to Brand Mobiliser bundles must be resolved through the OSGi environment.

For other JARs you may want to reference that are not already provided by the Brand Mobiliser runtime, it is recommended you embed these JARs into your bundle. To do this you enter a list of artifact names in this section.

Note: do not separate artifacts with new-line characters in this section – keep the list on one line.

Export Package

The export package section allows you to place certain Java classes to be made available to the OSGi runtime environment , through entry of the package names in this section.

Note: For Brand Mobiliser state plugin development, you do not have to export any packages, even though the runtime environment uses these packages. This is because the plugins are referenced in the environment as OSGi services, which are resolved in a different way.

Private Package

The private package section allows you to specify the package names of your Java classes to be included in the OSGi bundle.

Note: You must specify all package names in this section, otherwise they will not be included in the OSGi bundle and you will get 'ClassNotFound' exceptions at runtime.

Dynamic Import Package

The dynamic import package allows the specification of Java packages required by this bundle to be resolved dynamically, at runtime, rather than statically when the bundle is started.

Note: It is generally not recommended to use the dynamic import package section, because missing packages are not reported until they are referenced at runtime. Instead, use the Import Package section for all known package requirements.

Import Package

The import package allows the specification of Java packages required by this bundle to be resolved statically. When this bundle is started all imported packages are checked for and resolved from the OSGi environment. If there are missing packages, the bundle will not start and an error will be produced in the runtime error logs.

3.2 Spring Configuration

The files under the directory;

src/main/resources/META-INF/spring

contain the Spring configuration for a bundle.

Although Spring configuration .xml files may be named in any way understood by the Spring environment, we implement a naming convention for separation of Spring wiring, properties and services:

- beans-context.xml – This file contains the creation of standard Spring Framework beans.
- properties-context.xml – This file identifies any properties used in the bean configuration.
- services-context.xml – This file identifies the OSGi services that are exposed to the OSGI environment by this bundle.

Each of these is described in detail below.

3.2.1 Spring Beans

At a minimum a Spring Bean must be defined for each and every state created in the bundle.

In additional each Spring Bean may be configured with properties or other beans created that support the operation of the states.

beans-context.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns=http://www.springframework.org/schema/beans
      xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xmlns:osgix=http://www.springframework.org/schema/osgi-compendium
      xmlns:context=http://www.springframework.org/schema/context
      xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                       http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2.5.xsd
                       http://www.springframework.org/schema/osgi-compendium
    http://www.springframework.org/schema/osgi-compendium/spring-osgi-compendium-1.2.xsd">

  <!--
  ***********************************
  Beans Configuration
  ***********************************
  -->

  <!-- Auto-wire the configuration into the following beans -->
  <context:annotation-config />

    <bean id="SmappStateSample"
```

```
      class="com.sybase365.mobiliser.brand.plugins.sample.impl.SmappStateSample">

    <property name="version" value="${sample.version}"/>

  </bean>
</beans>
```

This example above shows a Spring Bean created for the state 'SmappStateSample'. It also has a property value injected into it, which references a version value from a property.

3.2.2 Bean Properties

Bundles may also receive property value that are provided by the runtime environment.

Brand Mobiliser uses the Spring property mechanism to inject property values into beans. These values are sourced from the standard OSGi Configuration Administration service.

For Brand Mobiliser, this consists of a configuration properties that is loaded at runtime by the AIMS OSGi bundles.

properties-context.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns=http://www.springframework.org/schema/beans
       xmlns:ctx=http://www.springframework.org/schema/context
       xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
       xmlns:osgi=http://www.springframework.org/schema/osgi
       xmlns:osgix=http://www.springframework.org/schema/osgi-compendium
       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                        http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
                        http://www.springframework.org/schema/osgi
    http://www.springframework.org/schema/osgi/spring-osgi-1.2.xsd
                        http://www.springframework.org/schema/osgi-compendium
    http://www.springframework.org/schema/osgi-compendium/spring-osgi-compendium-1.2.xsd">

  <!--
  ***********************************
  Properties Configuration
  ***********************************
  -->

    <!-- This configuration will not be seen until we run inside of OSGi -->    1
    <osgix:cm-properties id="sample-plugin-cfg" persistent-id="service.sample.plugin">

    <prop key="sample.version">1.0</prop>    2

  </osgix:cm-properties>

  <ctx:property-placeholder properties-ref="sample-plugin-cfg"/>    3

</beans>
```

1 This properties file uses the Spring OSGi compendium namespace to create a configuration admin id that is exposed to the OSGi runtime environment and made available to the Spring Framework environment as a bean. The 'persistent-id' identifies the file name that will hold the loaded value for these properties.

2 The list of properties reference by each <prop> element contain default values for property keys used elsewhere in the Spring configuration.

3 The standard Spring framework context namespace is used to identify the source of a property-placeholder. This allows the properties referenced and created from the configuration administration system to be made available to the Spring environment.

Note: Configuration of state plugins through this mechanism is a recommended approach. However, in certain instances, use of the Mobiliser Preferences services may be preferred, depending on the type of information being referenced. However, using Mobiliser Preferences is entirely dependent on the state code and will need to provided for.

Note: Configuration through OSGi configuration administration is particularly suited for per-instance configuration of system values, like URL locations of external services, for example.

For convention, it is recommended that a sample property file is provided in the bundle to aid the system configuration.

For the example above, a sample property file may be created as;

`src/main/resources/META-INF/sample/conf/service.sample.plugin.properties`

In this case, it would hold the following:

service.sample.plugin.properties:

```
sample.version=1.0
```

3.2.3 Services

At a minimum a Spring DM service must be defined for each and every state created in the bundle.

This mechanism frees the developer from the burden of programmatically installing the state plugin as an OSGi service. Instead, the Spring DM mechanism provided in the AIMS OSGi server uses the Spring configuration for publishing services.

Each State Plugin configured through the Spring DM configuration is exposed to the OSGi runtime environment as a service that implements the base SMAPP state plugin interface;

`at.ip2.mwiz.processing.plugins.smapp.SmappStatePlugin`

An example is shown below.

services-context.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns=http://www.springframework.org/schema/beans
      xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xmlns:osgi=http://www.springframework.org/schema/osgi
      xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                       http://www.springframework.org/schema/osgi
       http://www.springframework.org/schema/osgi/spring-osgi-1.2.xsd">

  <!--
  ***********************************
  OSGi Services Configuration
  ***********************************
  -->
  <osgi:service id="SmappStateSampleService"
         ref="SmappStateSample"
         interface="at.ip2.mwiz.processing.plugins.smapp.SmappStatePlugin" />

</beans>
```

Note: The Service id cannot be the same as the Spring Bean id. For simplicity and as a convention, use the related Spring Bean id with the suffix ‘Service’.

3.3 Building the Project

If the project is setup correctly, the project may be built (and unit tested) using the Maven tool:

```
C:\Temp\plugin-sample>mvn -version
Apache Maven 2.2.1 (r801777; 2009-08-06 20:16:01+0100)
Java version: 1.6.0_20
Java home: C:\Program Files\Java\jdk1.6.0_20\jre
Default locale: en_GB, platform encoding: Cp1252
OS name: "windows xp" version: "5.1" arch: "x86" Family: "windows"


C:\Temp\plugin-sample>mvn clean install

[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Building AIMS :: Object :: Brand Mobiliser Plugin - Sample State
[INFO]    task-segment: [clean, install]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
[INFO] Deleting directory C:\Temp\plugin-sample\target
[INFO] [resources:resources {execution: default-resources}]
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 4 resources
[INFO] [compiler:compile {execution: default-compile}]
[INFO] Compiling 1 source file to C:\Temp\plugin-sample\target\classes
[INFO] [resources:testResources {execution: default-testResources}]
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 4 resources
[INFO] Copying 1 resource
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] Nothing to compile - all classes are up to date
[INFO] [surefire:test {execution: default-test}]
[INFO] Surefire report directory: C:\Temp\plugin-sample\target\surefire-reports
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
There are no tests to run.

Results :

Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

[INFO] [bundle:bundle {execution: default-bundle}]
[WARNING] Warning building bundle com.sybase365.mobiliser.brand.plugins.sample:mobiliser-
brand-plugin-sample:bundle:1.0.0-SNAPSHOT : No translation found for macro: sample.version
[INFO] Preparing source:jar
[WARNING] Removing: jar from forked lifecycle, to prevent recursive invocation.
[INFO] No goals needed for project - skipping
[INFO] [source:jar {execution: attach-javasources}]
[INFO] Building jar: C:\Temp\plugin-sample\target\mobiliser-brand-plugin-sample-1.0.0-
SNAPSHOT-sources.jar
[INFO] [install:install {execution: default-install}]
[INFO] Installing C:\Temp\plugin-sample\target\mobiliser-brand-plugin-sample-1.0.0-
SNAPSHOT.jar to C:\Documents and
Settings\msw\.m2\repository\com\sybase365\mobiliser\brand\plugins\sample\mobiliser-brand-
plugin-sample\1.0.0-SNAPSHOT\mobiliser-brand-plugin-sample-1.0.0-SNAPSHOT.jar
[INFO] Installing C:\Temp\plugin-sample\target\mobiliser-brand-plugin-sample-1.0.0-
SNAPSHOT-sources.jar to C:\Documents and
Settings\msw\.m2\repository\com\sybase365\mobiliser\brand\plugins\sample\mobiliser-brand-
plugin-sample\1.0.0-SNAPSHOT\mobiliser-brand-plugin-sample-1.0.0-SNAPSHOT-sources.jar
[INFO] [bundle:install {execution: default-install}]
[INFO] Writing OBR metadata
[INFO] Installing
/C:/Documents%20and%20Settings/msw/.m2/repository/com/sybase365/mobiliser/brand/plugins/sam
```

```
ple/mobiliser-brand-plugin-sample/1.0.0-SNAPSHOT/mobiliser-brand-plugin-sample-1.0.0-SNAPSH

OT.jar
[INFO] Writing OBR metadata
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3 seconds
[INFO] Finished at: Tue Jan 04 15:04:07 GMT 2011
[INFO] Final Memory: 27M/65M
[INFO] ------------------------------------------------------------------------
```

4 Installing and Configuring States

This section describes how to install a State Plugin bundle into the Brand Mobiliser runtime environment.

4.1 Installation

To install the State Plugin bundle, the target '.jar' file must be copied into the Brand Mobiliser directory structure, under

`<home>/bundle/application`

This directory contains all the bundles that are installed as part of the Brand Mobiliser runtime application environment. However, this is not a mandatory requirement, and if you wish you can place the target '.jar' file anywhere on the local file system, which is accessible as a 'file:' reference from the Brand Mobiliser server.

Note: All of the AIMS *system* bundles are installed in the directory:

`<home>/bundle`

After the file is made available to the same file system as the Brand Mobiliser runtime, edit the standard configuration properties file of Brand Mobiliser and add the file to the list of known bundles that are to be auto-started. The following file fragment highlights the line added in order to install the bundle as created as an example in the last section:

<home>/conf/config.properties:

```
...
felix.auto.start.9 = \
 file:bundle/application/smppapi-1.0.1.jar \
 file:bundle/application/mobiliser-brand-plugin-sample-1.0.0-SNAPSHOT.jar \
 file:bundle/application/mobiliser-brand-plugin-base-1.1.0.jar \
 file:bundle/application/mobiliser-brand-plugin-smapp-1.1.0.jar \
 file:bundle/application/mobiliser-brand-plugin-smapp-sample-1.1.0.jar \
 file:bundle/application/mobiliser-brand-jms-messages-1.1.0.jar \
 file:bundle/application/mobiliser-brand-plugin-channel-1.1.0.jar \
 file:bundle/application/mobiliser-brand-plugin-core-1.1.0.jar \
 file:bundle/application/mobiliser-brand-processing-1.1.0.jar
...
```

Note: All bundles are referenced in this config.properties file. The default setting for the AIMS OSGi Brand Mobiliser server is to re-initialise it's Felix bundle cache on each and every restart. This means all cached bundles under

<home>/.felix-cache

Will be deleted and each bundle referenced in the config.properties file will be re-read. Although this causes startup of the server to be more extended, it allows for a consistent runtime environment to be created based on configuration. OSGi allows for bundles to be dynamically added and removed at runtime. However, for a stable product it is recommended a consistent runtime environment organised by the config.properties is preferred.

After starting (or restarting) the server, there should be no errors, such as ClassNotFound exceptions or package/bundle wiring problems. If there are, you should revisit the OSGi configuration and examine your import/private/dynamic package specifications.

Note: If your bundle requires configuration before it can be run, refer to the next section ('Configuration') before starting/restarting your server.

To ensure your bundle has resolved (package wiring can be fulfilled) and started you have the option to use either the local Telnet interface, or the AIMS system web console.

4.1.1 Telnet Interface

Using the Telnet interface, connect to localhost port 5365 and run the ‘ps’ command as shown below.

Note: The Telnet interface is only listening on the localhost interface, to ensure security of the runtime environment is not compromised.

Somewhere in the output of the command should be the name of the state bundle.

```
C:\>telnet localhost 5365

Felix Remote Shell Console:
============================

-> ps
START LEVEL 20
   ID   State         Level  Name
[   0] [Active     ] [    0] System Bundle (3.0.9)
[   1] [Active     ] [   20] ARF :: System :: arf-sys (0.3.0)
[   2] [Active     ] [   20] ARF :: System :: arf-util-commands (0.3.0)
[   3] [Active     ] [   20] ARF :: System :: cm-bridge (0.3.0)
[   4] [Active     ] [   20] ARF :: System :: cm-command (0.3.0)
[   5] [Active     ] [   20] Java Activation API (1.1.1)
[   6] [Active     ] [   20] Java Messaging System API (1.1.0)
[   7] [Active     ] [   20] J2EE Application Management Specification (1.0.1)
[   8] [Active     ] [   20] Java Transaction API (1.1.0)
[   9] [Active     ] [   20] Java SOAP API (1.3.0)
[  10] [Active     ] [   20] AOP Alliance API (1.0.0)
[  11] [Active     ] [   20] Apache ActiveMQ Core (5.3.0)
[  12] [Active     ] [   20] Apache ActiveMQ - KahaDB (5.3.0)

...
[  42] [Active     ] [    9] AIMS :: 3rd Party :: SMPPAPI (ie.omk) (1.1.0)
[  43] [Active     ] [    9] AIMS :: Object :: Brand Mobiliser Plugin - Sample
State (1.0.0.SNAPSHOT)
[  44] [Active     ] [    9] AIMS :: Object :: Brand Mobiliser Plugin Base API (1.1.0)

...

->
```

This bundle should report a state of ‘Active’.

4.1.2 Web Console

Using a Web Browser, connect to the address;

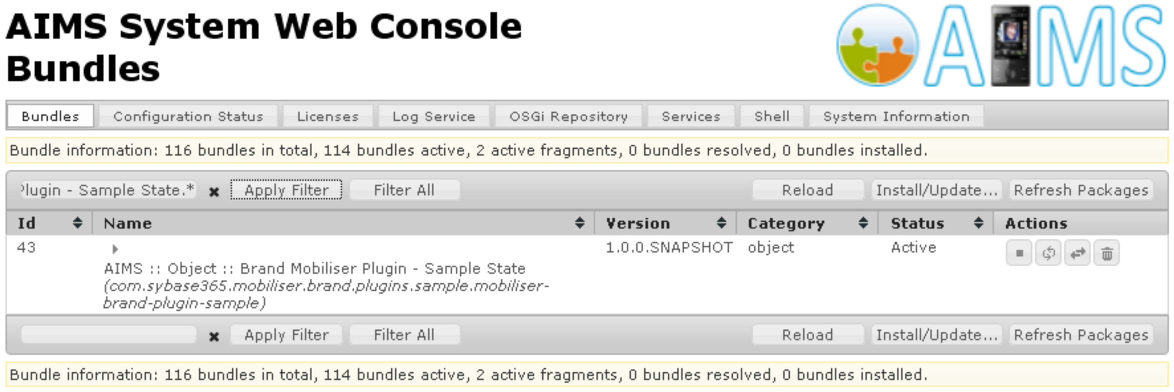
`http://localhost:8080/system/console`

Note: The Web Console restricts access based on a list of allowable IP addresses, to ensure security of the runtime environment is not compromised. By default, only localhost (127.0.0.1) is allowed to access. To add alternative allowable IP addresses, update the Configuration Administration file named ‘org.apache.felix.webconsole.internal.servlet.OsgiManager.properties’ and add the allowable IP address to the ‘allowed.ip.list’. (This property is a comma separated list of IP addresses.)

When prompted for the ‘AIMS Management Console’ username and password use the values;

User Name:	**sybase365**
Password:	**fr4nt1c**

Under the ‘Bundles’ tab will be a list of all bundles; use the Filter criteria to display only those bundles with ‘Plugin’ in their name by using the regular expression ‘.*Plugin.*’.



AIMS System Web Console
Bundles

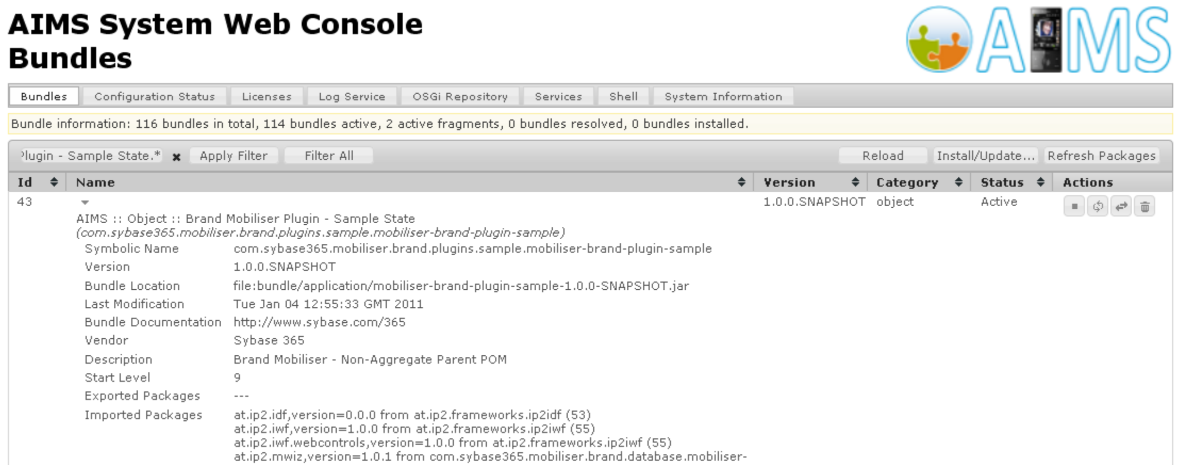
Bundle information: 116 bundles in total, 114 bundles active, 2 active fragments, 0 bundles resolved, 0 bundles installed.

Id	Name	Version	Category	Status	Actions
43	AIMS :: Object :: Brand Mobiliser Plugin - Sample State *(com.sybase365.mobiliser.brand.plugins.sample.mobiliser-brand-plugin-sample)*	1.0.0.SNAPSHOT	object	Active	

Bundle information: 116 bundles in total, 114 bundles active, 2 active fragments, 0 bundles resolved, 0 bundles installed.

This bundle should report a state of ‘Active’.

To view the details of your bundle, click on the bundle name and you will be displayed all the bundle meta-data created by the Maven Bundle Plugin (from the bundle’s Manifest file), together with all the package wiring (exported/imported packages and from where) and services information (the interface of the exported services).



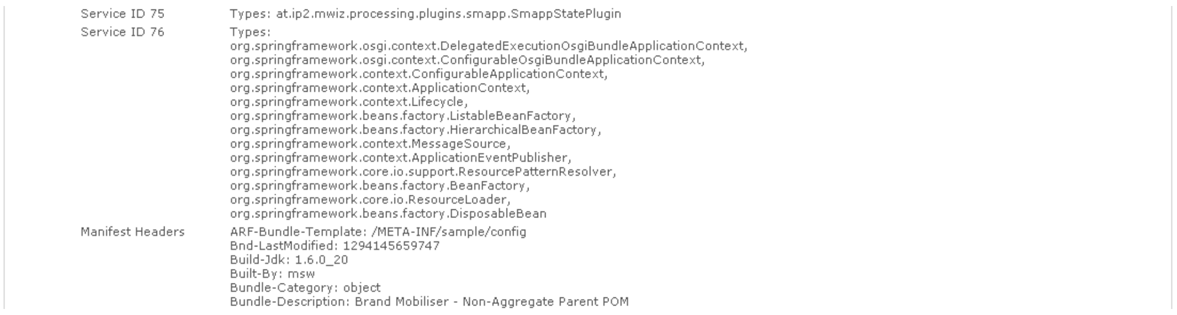
AIMS System Web Console
Bundles

Bundle information: 116 bundles in total, 114 bundles active, 2 active fragments, 0 bundles resolved, 0 bundles installed.

Id	Name	Version	Category	Status	Actions
43	AIMS :: Object :: Brand Mobiliser Plugin - Sample State *(com.sybase365.mobiliser.brand.plugins.sample.mobiliser-brand-plugin-sample)*	1.0.0.SNAPSHOT	object	Active	

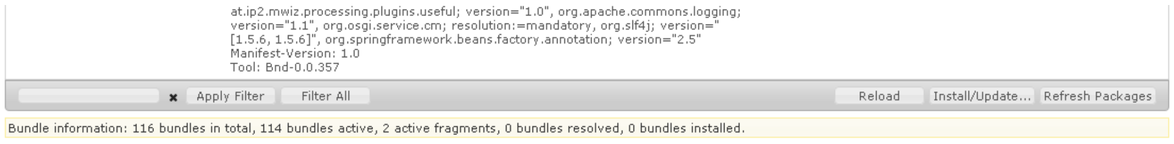
```
Symbolic Name          com.sybase365.mobiliser.brand.plugins.sample.mobiliser-brand-plugin-sample
Version                1.0.0.SNAPSHOT
Bundle Location        file:bundle/application/mobiliser-brand-plugin-sample-1.0.0-SNAPSHOT.jar
Last Modification      Tue Jan 04 12:55:33 GMT 2011
Bundle Documentation   http://www.sybase.com/365
Vendor                 Sybase 365
Description            Brand Mobiliser - Non-Aggregate Parent POM
Start Level            9
Exported Packages      ---
Imported Packages      at.ip2.idf,version=0.0.0 from at.ip2.frameworks.ip2idf (53)
                       at.ip2.iwf,version=1.0.0 from at.ip2.frameworks.ip2iwf (55)
                       at.ip2.iwf.webcontrols,version=1.0.0 from at.ip2.frameworks.ip2iwf (55)
                       at.ip2.mwiz,version=1.0.1 from com.sybase365.mobiliser.brand.database.mobiliser-
```

…



```
Service ID 75          Types: at.ip2.mwiz.processing.plugins.smapp.SmappStatePlugin
Service ID 76          Types:
                       org.springframework.osgi.context.DelegatedExecutionOsgiBundleApplicationContext,
                       org.springframework.osgi.context.ConfigurableOsgiBundleApplicationContext,
                       org.springframework.context.ConfigurableApplicationContext,
                       org.springframework.context.ApplicationContext,
                       org.springframework.context.Lifecycle,
                       org.springframework.beans.factory.ListableBeanFactory,
                       org.springframework.beans.factory.HierarchicalBeanFactory,
                       org.springframework.context.MessageSource,
                       org.springframework.context.ApplicationEventPublisher,
                       org.springframework.core.io.support.ResourcePatternResolver,
                       org.springframework.beans.factory.BeanFactory,
                       org.springframework.core.io.ResourceLoader,
                       org.springframework.beans.factory.DisposableBean
Manifest Headers       ARF-Bundle-Template: /META-INF/sample/config
                       Bnd-LastModified: 1294145659747
                       Build-Jdk: 1.6.0_20
                       Built-By: msw
                       Bundle-Category: object
                       Bundle-Description: Brand Mobiliser - Non-Aggregate Parent POM
```

…



```
at.ip2.mwiz.processing.plugins.useful; version="1.0", org.apache.commons.logging;
version="1.1", org.osgi.service.cm; resolution:=mandatory, org.slf4j; version="
[1.5.6, 1.5.6]", org.springframework.beans.factory.annotation; version="2.5"
Manifest-Version: 1.0
Tool: Bnd-0.0.357
```

Bundle information: 116 bundles in total, 114 bundles active, 2 active fragments, 0 bundles resolved, 0 bundles installed.

4.2 Configuration

The optional configuration of the your State Plugin bundle will depend on any configuration administration/property file references, as described in the previous section.

In the example in the previous section, the file;

service.sample.plugin.properties:

```
sample.version=1.0
```

Shoud be placed into the directory;

`<home>/conf/cfgload`

This will get loaded into the OSGi configuration admin system and the file will be move into the directory;

`<home>/conf/cfgbackup`

Note: After each server start or restart, the contents of the 'cfgbackup' directory and moved into 'cfgload' directory so that all properties are associated with the re-initialised felix bundle cache.

To test the configuration has been read, you can use either the local Telnet interface, or the AIMS system web console.

4.2.1 Telnet Interface

Using the Telnet interface, connect to localhost port 5365 and run the 'cm list' command as shown below.

Somewhere in the output of the command should be the name of the state bundle.

```
C:\>telnet localhost 5365

Felix Remote Shell Console:
============================

->cm list

Configuration list:

service.sample.plugin                              file:bundle/application/mobiliser-brand-
plugin-sample-1.0.0-SNAPSHOT.jar

service.dsprovider                                 file:bundle/application/mwiz-osgi-
services-1.0.0.jar

service.mobiliser.plugin                           file:bundle/application/mobiliser-brand-
plugin-core-1.0.1.jar

org.ops4j.pax.logging                              file:bin/pax-logging-service-1.5.0.jar

service.coreprocessing                             file:bundle/application/mobiliser-brand-
processing-1.0.0.jar

com.sybase365.arf.management.system.responder      file:/C:/AIMS/Tests/aims-brand-
mobiliser-1.0.0/bundle/mm-responder-0.3.0.jar
```

```
->
```

In the list you should see your ‘service pid’.

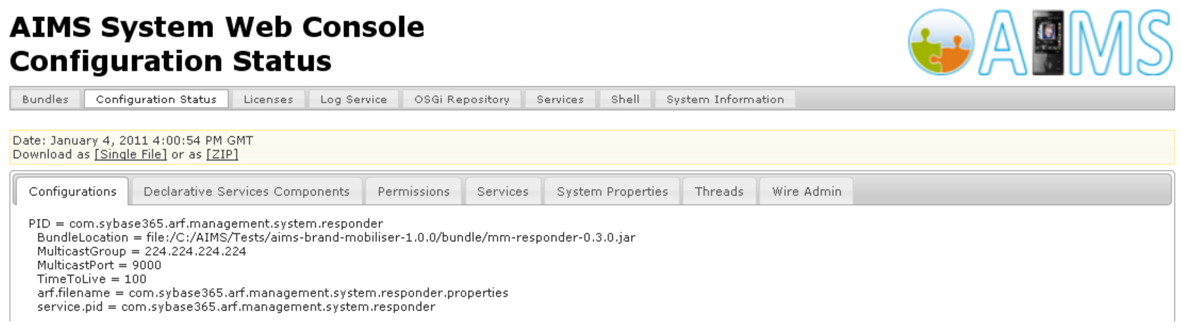
4.2.2 Web Console

Using a Web Browser, connect to the address;

`http://localhost:8080/system/console`

When prompted for the username and password use the values supplied previously in this document.

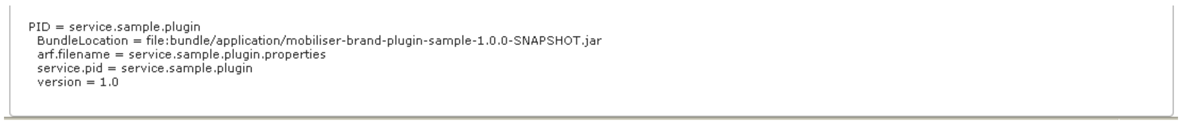
Under the ‘Configuration Status’ tab and ‘Configurations’ sub-tab will be a list of all configurations.



```
Date: January 4, 2011 4:00:54 PM GMT
Download as [Single File] or as [ZIP]
```

```
PID = com.sybase365.arf.management.system.responder
  BundleLocation = file:/C:/AIMS/Tests/aims-brand-mobiliser-1.0.0/bundle/mm-responder-0.3.0.jar
  MulticastGroup = 224.224.224.224
  MulticastPort = 9000
  TimeToLive = 100
  arf.filename = com.sybase365.arf.management.system.responder.properties
  service.pid = com.sybase365.arf.management.system.responder
```

…



```
PID = service.sample.plugin
  BundleLocation = file:bundle/application/mobiliser-brand-plugin-sample-1.0.0-SNAPSHOT.jar
  arf.filename = service.sample.plugin.properties
  service.pid = service.sample.plugin
  version = 1.0
```

DOCUMENT ID: DC01689-01-0110-01
LAST REVISED: March 2011