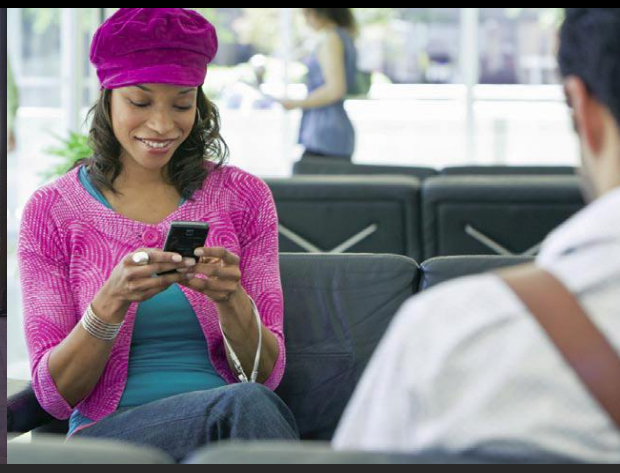
SYBASE®
An SAP® Company

Brand Mobiliser
Using the USSD States

[PRODUCT DOCUMENTATION]





www.sybase.com

Contents:

1 Introduction 1
1.1 References 1
2 USSD States 2
2.1 Send USSD Input 3
2.2 Send USSD Text 3
2.3 Send USSD Menu 5

Principal Author

Sybase365 - mCommerce

Revision History

Version 1.2 - September 2011

Version 1.1 - June 2011

1 Introduction

Brand Mobiliser supports two internal channels where messages may be received and sent and interactive applications will be executed:

- SMPP – For standard SMS.
- JMS – For interface to additional channels.

The JMS interface is specifically used to communicate to the Channel Manager, currently a separately running component of the Mobiliser stack, external to the Brand Mobiliser installation.

The Channel Manager can be used to interface to a number of standard interfaces, such as;

- Jabber/XMPP
- MSN, and
- HTTP

but also can be customised to communicate to interfaces for which standards vary, such as;

- USSD, and
- non-standard SMPP.

This document describes the options within Brand Mobiliser for generating interactive applications that use all the features and functions of USSD.

This document assumes that the audience is familiar with the concepts of Brand Mobiliser Application development, as described in [1], and has a basic knowledge of the Brand Mobiliser State Development API, as described in [2].

1.1 References

1. Brand Mobiliser Development Manual; Version 1.2 – September 2011
2. Brand Mobiliser State Developers Guide; Version 1.2 – September 2011

2 USSD States

The section describes how to build applications that used the sample USSD states.

Installation

To install the USSD states requires the following;

1. Check that the JAR file for the USSD states is installed in the relevant Brand installation location.

Ensure the Jar file;

`mobiliser-brand-plugin-ussd-1.2.0.jar`

Is in the location;

`<brandroot>/application/bundle/.`

2. Update the list of install plugin bundles.

Open the file in your chosen editor;

`<brandroot>/conf/config.properties`

And, add the following line, in bold below, to this file;

```
...
 felix.auto.start.9 = \
 file:bundle/application/smppapi-1.0.2.jar \
 file:bundle/application/mobiliser-brand-plugin-base-1.2.0.jar \
 file:bundle/application/mobiliser-brand-plugin-smapp-1.2.0.jar \
 file:bundle/application/mobiliser-brand-plugin-ussd-1.2.0.jar \
 file:bundle/application/mobiliser-brand-jms-messages-1.2.0.jar \
 file:bundle/application/mobiliser-brand-plugin-channel-1.2.0.jar \
 file:bundle/application/mobiliser-brand-plugin-core-1.2.0.jar \
 file:bundle/application/mobiliser-brand-processing-1.2.0.jar
...
```

Start or restart your Brand server for the plugins to be recognised. If states are installed correctly, they will appear in the Follow-Up state list drop-down, as below.

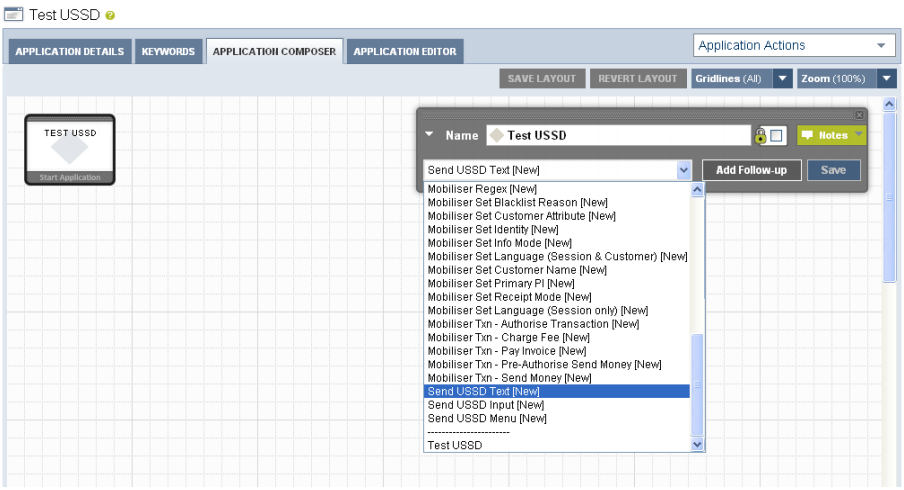


Figure 1. Installed Sample USSD States

Note: All of the following states are interoperable across the SMPP and JMS channels. When a SMPP channel is configured, the message is processed in the same way as plain SMS message. If a JMS channel

is configured, the message is processed by the Channel Manager understanding the underlying JMS object type.

2.1 Send USSD Input

This state is used to send a prompt for input.

The input prompt is a regular message that can be specified in the state configuration.

You may optionally specify an input validation value and a mask input setting.

- The input validation value and/or URL will be some form of validation of response values expected.
- The mask input will allow response inputs made by the subscriber to be masked on the phone.

Note: The use of input validation and mask input is dependent on the Channel Manager interfaces into the USSD Gateway. Please refer to your USSD Gateway capability and Channel Manager interfaces as to whether input validation and mask input are supported.

The following diagram shows an Application Composer view of an application using a 'Send USSD Input' state.

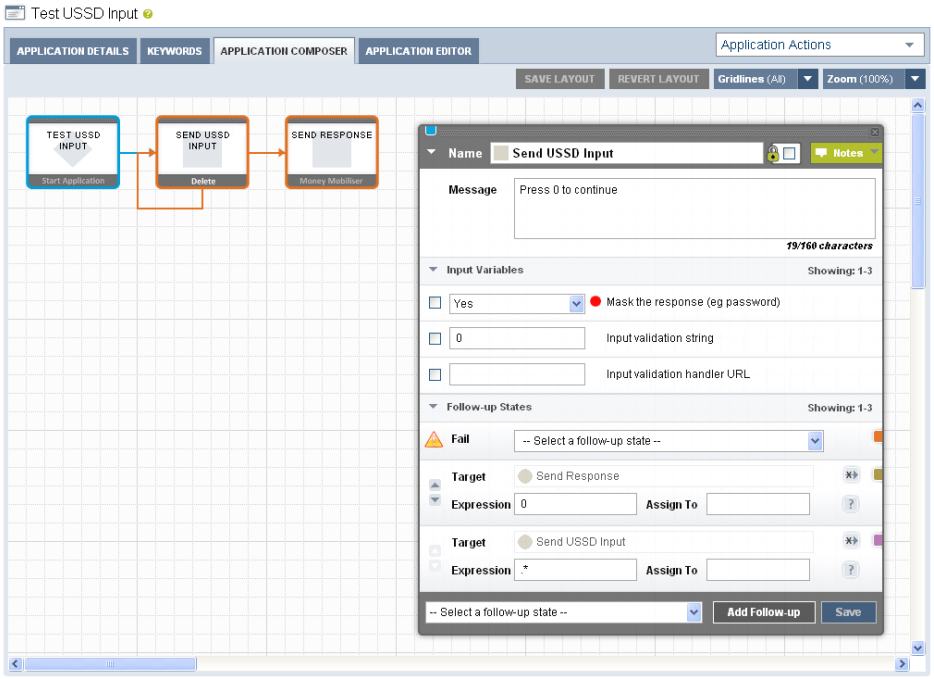


Figure 2. Sample Send USSD Input State

The Send Ussd Input state allows the use of the on 'Fail' follow-up. This follow-up state is only used if there is an internal problem formatting the state text.

Other follow-up states will be used to branch based on the response received back from the subscriber.

2.2 Send USSD Text

This state is used to send a simple text notification to the subscriber.

The text is sent as a notification that requires no subscriber input other than following a simple 'accept' or 'confirm'. This confirmation maybe sent back to Channel Manager.

The following diagram shows an Application Composer view of an application using a 'Send USSD Text' state.

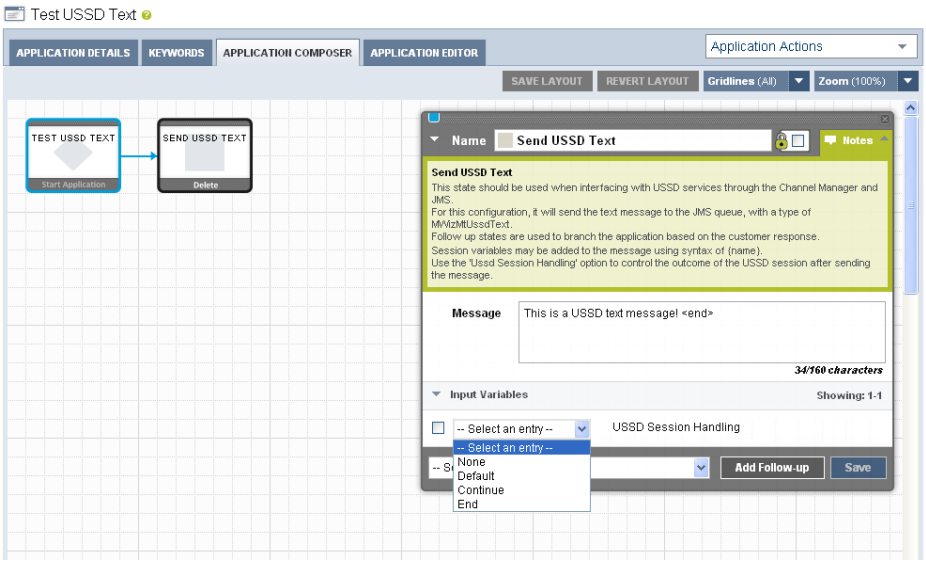


Figure 3. Sample Send USSD Text State

The 'Send USSD Text' allows the configuration of a option to specify how any associated USSD session is managed by Channel Manager.

Note: This configuration option is only relevant when Channel Manager is customised and configured to manage USSD session information.

The options for the 'USSD Session Handling' are;

None – This is the standard option, which will be used in no other option is selected. With this option no specific processing for USSD session handling is performed.

Default – When this option is selected, the type of session handling is based on the follow-up transitions from this state;

If there are no follow-up transitions from this state - then the application terminates and the user's Brand Mobiliser session will terminate, which will mean that Channel Manager should instruct the USSD Gateway it is interfacing with to also terminate the USSD session for this user.

If there are follow-up transitions from this state - then the application continues and the user's Brand Mobiliser session will also continue, which will mean that Channel Manager should instruct the USSD Gateway it is interfacing with to continue the USSD session for this user.

Continue – this option can be used to override the default behaviour, so that Channel Manager should instruct the USSD Gateway it is interfacing with to continue the USSD session for this user, regardless of whether there are follow-up transitions or not.

End – this option can be used to override the default behaviour, so that Channel Manager should instruct the USSD Gateway it is interfacing with to also terminate the USSD session for this user, regardless of whether there are follow-up transitions or not.

Note: In the situation where this type of state will require Channel Manager to end the USSD session, this state will suffix the text of the message sent with the text "[$[End]$]". This must be used by the Channel Manager interacting with the USSD Gateway to instruct it to end the USSD session. It is expected that Channel Manager will strip off this text before passing the message text onto the USSD Gateway.

2.3 Send USSD Menu

This type of state allows the creation of a dynamic menu which will be presented to the subscriber as a series of options with relevant responses. The menu is formed from:

- Header text – this is the text that is provided in the text input field on the state editor panel.
- Options – these are provided programmatically by instances of this state type, as provided by a state developer.
- Paging Options – this type of state will automatically end 'Next' and 'Previous' options to the menu list if the number of options go beyond the allowable number.
- End Option – an option can be added to each menu which is the 'end' or 'exit' option.

This state type is actually an abstract type of which instances can be developed, so that the dynamic menu capability can be developed.

This state is similar to the 'Send USSD Input' state because it expects a response from the subscriber. However, the response is expected to be one of the menu options.

The following diagram shows an Application Composer view of an application using a sample implementation of the 'Send USSD Menu' state.

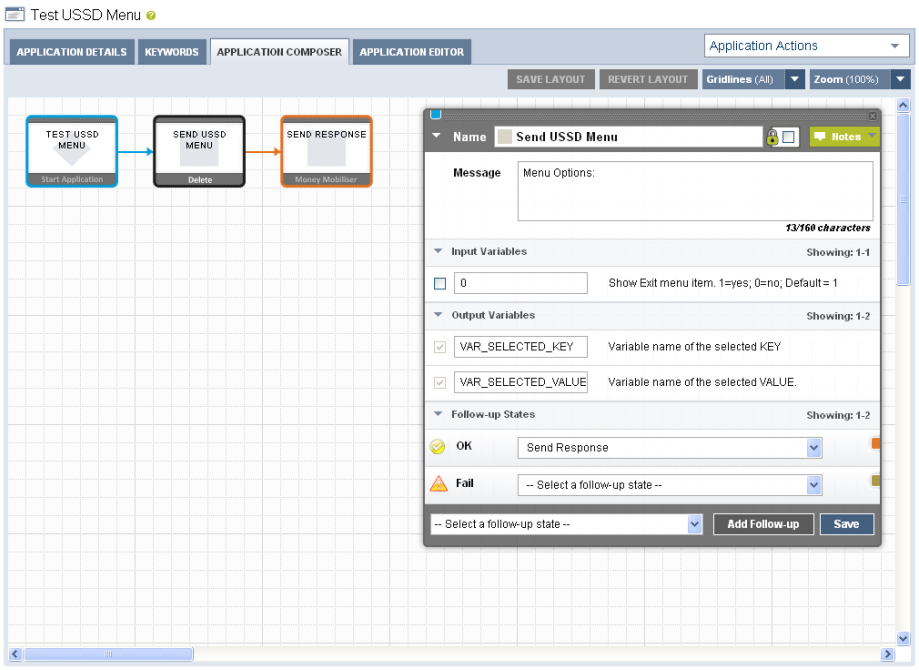


Figure 4. Sample Send USSD Menu Application

The configuration of the 'Send USSD Menu state' comprises of:

Message:

The text for the menu header.

Input Variables:

The standard input variables as described by the abstract dynamic menu are;

Show Exit menu item. – Show or hide the menu item for end/exit.

Output Variables:

The standard output variables as described by the abstract dynamic menu are;

Variable name of the selected KEY – the session variable where the selected option key will be placed.

Variable name of the selected VALUE – the session variable where the selection option value will be placed.

Follow-Up States:

OK – This transition is normally used when the menu is created successfully and the user has sent a valid response key.

Fail – This transition is only used when there is an internal processing error of the abstract dynamic menu.

Note: Dynamic transitions MAY be implemented, but they will not be followed unless specifically coded for in the implementation of the menu (as described below).

2.3.1 Implementing the Send USSD Menu

To provide an implementation of the abstract menu requires creating a state that extends from the class:

at.ip2.mwiz.processing.plugins.implementation.ussd. AbstractDynamicUssdMenu

This abstract super-class manages all the creation and structuring of the message. It relies on overriden and implemented abstract methods to provide the information required.

The following methods are required to be implemented. Refer to the method comments for more information.

```
...

/*
 * The normal state attribute list is already set in the
protected abstract Attribute[] getStateAttributeList();

/*
 * Do any initialization of the dynamic list, possibly based on subscriber information
 */
protected abstract SmappState init(SmappStateProcessingAction action)
      throws MwizProcessingException, DBException, JAXBException,
      IOException, ServiceException, RequiredParameterMissingException;

/*
 * Return the list of options in a format [[key,text],...]
 */
protected abstract List<KeyValuePair<String, String>> getMenuList()
      throws NumberFormatException, DBException,
      RequiredParameterMissingException;

/*
 * Allow the branching of processing based on selected key.
 * If you want to use the configured dynamic following transitions, simply
 * override this method and 'return continueDyn(key)', otherwise override
 * this method and 'return null' to follow the OK transition when the user
```

```
 * selected an entry.
 */
protected abstract SmappState saveSessionVariables(
      SmappStateProcessingContext context, String key, String value)
      throws MwizProcessingException, DBException,
      RequiredParameterMissingException;

...
```

2.3.2 Sample USSD Menu Code

As an example of a complete implementation of the 'Send USSD Menu' the following code can be used as a reference.

This code produces a menu based on the option values; "Option 1", "Option 2", "Option 3" and "Option 4".

at.ip2.mwiz.processing.plugins.implementation.ussd.SmappStateSendUssdMenu:

```
package at.ip2.mwiz.processing.plugins.implementation.ussd;

import at.ip2.mwiz.processing.plugins.useful.KeyValuePair;
import at.ip2.mwiz.processing.exceptions.MwizProcessingException;
import at.ip2.mwiz.processing.plugins.smapp.controls.Attribute;
import at.ip2.mwiz.processing.plugins.smapp.state.RequiredParameterMissingException;
import at.ip2.mwiz.processing.plugins.smapp.SmappStateProcessingAction;
import at.ip2.mwiz.processing.plugins.smapp.SmappStateProcessingContext;

import com.sybase365.mobiliser.brand.dao.DBException;
import com.sybase365.mobiliser.brand.jpa.SmappState;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Arrays;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import javax.xml.bind.JAXBException;
import javax.xml.rpc.ServiceException;

/**
 * Loads all available Languages and puts it into a menu
 *
 */
public class SmappStateSendUssdMenu extends AbstractDynamicUssdMenu {

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

  private static final String[] OPTIONS =
         { "Option 1", "Option 2", "Option 3", "Option 4" };

  privateList<String> listOfOptions = Arrays.asList(OPTIONS);

  private static Attribute[] stateAttr;

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

  @Override
  protected Attribute[] getStateAttributeList() {
```

```
    return stateAttr;
  }

  @Override
  public long getStateId() {
    return 485002L;
  }

  @Override
  public String getStateName() {
    return "Send USSD Menu";
  }

  @Override
  public String getStateNotes() {
    return "This state generates a sample USSD Menu.\n"
        + "Use the following follow up states:\n"
        + "- OK: If user selected a menu item.\n"
        + "- FAIL: If an error occurs.";
  }

  @Override
  protected String getStateInfoText() {

    return "This state generates a sample USSD Menu.<br/>"
        + "Use the following follow up states:<br />"
        + "- FAIL: If an error occurs.<br />"
        + "- OK: If user selected a menu item.";
  }

  @Override
  public boolean supportsOkTransition() {
    return true;
  }

  @Override
  public boolean supportsFailTransition() {
    return true;
  }

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

    if (listOfOptions == null) {
      return continueFail();
    }

    return null;
  }

  @Override
  protected List<KeyValuePair<String, String>> getMenuList()
      throws NumberFormatException, DBException,
      RequiredParameterMissingException {

    List<KeyValuePair<String, String>> list =
            new ArrayList<KeyValuePair<String, String>>();

    int optionNumber = 1;

    for (String option : listOfOptions) {
      KeyValuePair<String, String> keyVal = new KeyValuePair<String, String>();
      keyVal.setKey(Integer.toString(optionNumber));
      keyVal.setValue(option);
      list.add(keyVal);
      optionNumber++;
    }
```

```
    return list;
  }

  @Override
  protected SmappState saveSessionVariables(
     SmappStateProcessingContext context, String key, String value)
     throws MwizProcessingException, DBException,
     RequiredParameterMissingException {

    return null;
  }
}
```

SYBASE, INC.
WORLDWIDE HEADQUARTERS
ONE SYBASE DRIVE
DUBLIN, CA 94568-7902 USA
Tel: 1 800 8 SYBASE

www.sybase.com

DOCUMENT ID: DC60006-01-0120-01
LAST REVISED: September 2011