IBM Unica Campaign
Version 8.5.x Publication Date: June 7, 2011

Validation PDK Guide

Table of Contents

Contacting IBM Unica Technical Support

This section provides information about contacting IBM Unica Technical Support.

The key topics in this section are:

Contacting IBM Unica technical support

If you encounter a problem that you cannot resolve by consulting the documentation, your company's designated support contact can log a call with Unica technical support. Use the information in this section to ensure that your problem is resolved efficiently and successfully.

If you are not a designated support contact at your company, contact your IBM Unica Campaign administrator for information.

Information you should gather

Before you contact IBM Unica technical support, you should gather the following information:

- A brief description of the nature of your issue.
- Detailed error messages you see when the issue occurs.
- Detailed steps to reproduce the issue.
- Related log files, session files, configuration files, and data files.
- Information about your product and system environment, which you can obtain as described in "System information" below.

System information

When you call IBM Unica technical support, you might be asked to provide information about your environment.

If your problem does not prevent you from logging in, much of this information is available on the About page, which provides information about your installed Unica applications.

You can access the About page by selecting **Help > About**. If the About page is not accessible, you can obtain the version number of any Unica application by viewing the **`version.txt`** file located under each application's installation directory.

Contact information for Unica technical support

For ways to contact IBM Unica technical support, see the IBM Unica Product Technical Support website: (**http://www.unica.com/about/product-technical-support.htm**).

1 IBM Unica Validation Plug-in Developer’s Kit (PDK)

The IBM Unica Validation Plug-in Developer’s Kit (PDK) allows you to develop custom validation logic for use in IBM Unica Campaign. The Validation PDK is a subclass of a more generic plug-in framework provided with IBM Unica Campaign.

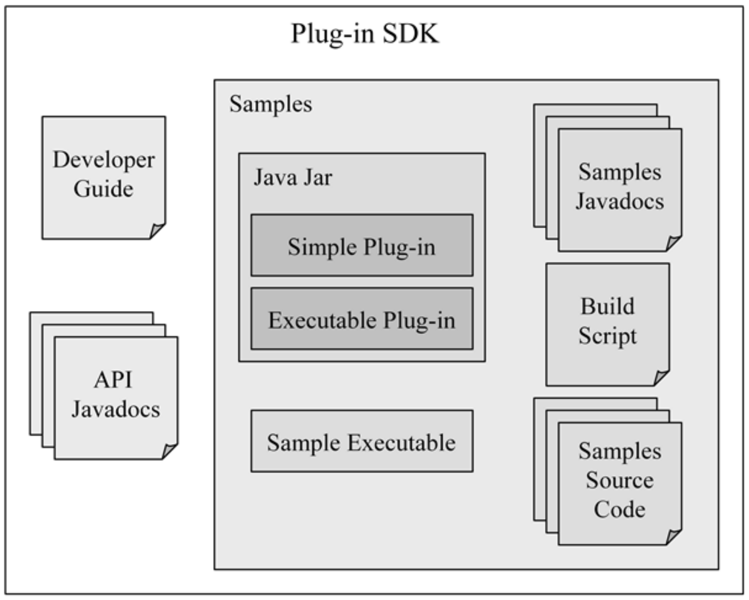
Key sections in this chapter include:

Additional Validation PDK help

If you require help using the Validation Plug-in Developer’s Kit (PDK), please contact Unica Technical Support with your questions. Refer to “Contacting IBM Unica technical support” on page v for details about contacting Unica Technical Support.

Contents of the Validation PDK

The Validation PDK contains everything that you need to develop Java plug-ins or command line executables to add validation to IBM Unica Campaign.



Components of the Validation PDK

The Validation Plug-in Developer’s Kit (PDK) has the following components.

Developer’s Guide	This document.
Samples	The samples are documented, buildable examples of how to use the PDK.
Java .jar file	A sample jar file containing the sample plug-ins. The jar contains: **Simple Plug-in**: an example of a self-contained validator class. **Executable Plug-in**: an example validator that will run a user-defined command line executable to perform validation.
Sample Executable	A command-line executable that can be used with the executable plug-in on UNIX.
Sample Javadocs	Javadocs describing the sample validator classes.
Build Script	An Ant script that will build the included source code into usable validator plug-ins.
Samples Source Code	The Java source code for the simple validator and the executable validator.

Finding reference material

The Validation PDK contains reference information for both the Plug-In API and the sample code. The reference information is in the form of Javadocs. To view the documentation, open the following file:

```
C:\Unica_Home\Campaign_Home\devkits\validation\javadoc\index.html
```

where *Unica_Home* is the path to your IBM Unica root install directory and *Campaign_Home* is the path to your Campaign install directory.

For example, a typical path to the documentation could be as follows:

```
C:\Unica\Campaign\devkits\validation\javadoc\index.html
```

You can view the Javadocs in any web browser.

Capabilities of the Validation PDK

A plug-in made with the Validation PDK can perform custom validation logic for campaigns and/or offers. Some possible uses of the validation logic are:

- Checking extended attributes (for example, valid versus optional, or related dependent fields)

- Providing authorization services that are outside of the scope of IBM Unica Marketing Platform (for example, validating which users are allowed to edit which extended attributes).

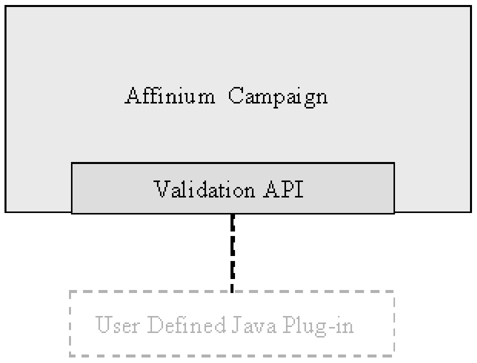
Load plug-in into IBM Unica Campaign

There are two ways to use the API:

- Use it to build a Java class plug-in that is loaded into the application
- Use one of the included plug-ins to call out to an executable application to handle the validation

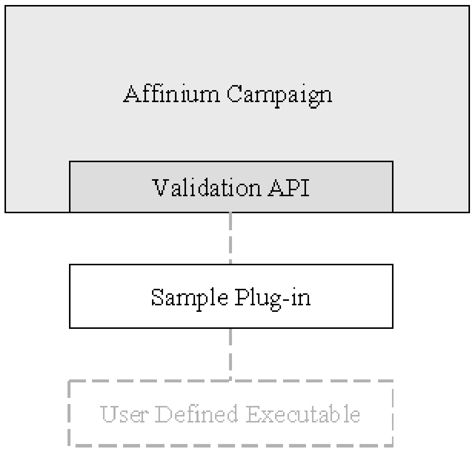
Build a Java class plug-in that is loaded into the application

The Validation PDK provides the interfaces, helper classes, and Developer's tools for developing these classes.



Call an application to handle validation

The second way to use the Validation PDK is to use one of the included plug-ins to call out to an executable application to handle the validation:



This executable may be written in any language, must reside on the IBM Unica Campaign server, and will be executed on the server. The plug-in that calls the executable sends in an XML file that contains the information to be validated (for example, the user editing the object and the before/after values for all standard and extended attributes of that object). IBM Unica Campaign expects results information in the form of an XML file in return. For more information, see “Developing a plug-in” on page 13.

Offer versus campaign validation

The Validation PDK can validate offers and campaigns. If a validation plug-in is defined, it is automatically called by IBM Unica Campaign each time an offer and/or campaign object is saved. IBM Unica Campaign sets a flag when it calls the plug-in’s validate method. IBM Unica Campaign passes the following flags:

- `ValidationInputData.CAMPAIGN_VALIDATION`, when adding or changing a campaign, or
- `ValidationInputData.OFFER_VALIDATION`, when adding or editing an offer.

You can then use these flags to construct validation rules applying to offers and campaigns.

Configuring the Validation PDK

The Validation PDK uses configuration parameters that tell Campaign how to find the plug-in class that should be used and offers a way to pass some configuration information into those plug-ins.

All of the settings are on the Configuration page at:

`Campaign > partitions > partition[n] > validation`

Validation works with multiple partitions; `partition[n]` can be changed to any partition name to provide validation routines for those partitions as well.

This section contains the following settings:

- validationClass
- validationClasspath
- validatorConfigString

validationClass

Description	This is the name of the class to use for validation. The value of the validationClasspath property indicates where this class resides.
Details	The class must be fully qualified with its package name. If this property is not set, Campaign will not perform any custom validation.
Example	`com.unica.campaign.core.validation.samples.SimpleCampaignValidator` This sets `validationClass` to the `SimpleCampaignValidator` class from the sample code.
Default	By default, no path is set: `<property name="validationClass" />`

validationClasspath

Description This is the path to the class used for custom validation. It can be either be a full path or a relative path.

Details If the path ends in a slash (forward slash / for UNIX or backslash \ for Windows), Campaign assumes it to be a path to a directory that contains the Java plug-in class that should be used.

If the path does not end in a slash, Campaign will assume it is the name of a .jar file that contains the Java class. For example, the value

```
/opt/Unica/Campaign/devkits/validation/lib/validator.jar?
```

is the path on a UNIX platform that would point to the jar file that comes out of the box with the plug-in developer's kit.

If the path is relative, the behavior depends on the application server that is running Campaign. WebLogic uses the path to the domain work directory, which by default, this is

```
c:\bea\user_projects\domains\mydomain
```

If the setting does not contain a string, Campaign will not attempt to load a plug-in.

Example

```
/opt/Unica/Campaign/devkits/validation/lib/validator.jar?
```

This is the path on a UNIX platform that points to the jar file that comes packaged with the plug-in developer's kit.

Default By default, no path is set: `<property name="validationClasspath" />`

See also See validationClass for information on designating the class to use.

validatorConfigString

Description	This is a string that is passed into the validator plug-in when it is loaded by Campaign.
Details	How the plug-in uses this string is up to the designer. For example, you could use it to send a configuration string into your plug-in when the system loads it. For example, the `ExecutableCampaignValidator` (from the sample executable plug-in included with the PDK) uses this property to indicate the executable to run.
Example	To run the sample Bourne shell script as the validation script, set `validatorConfigString` to `/opt/unica/campaign/devkits/validation/src/com/unica/campaign/core/validation/samples/validate.sh`
Default	By default, no path is set: `<property name="validatorConfigString" />`

Developing a plug-in

Overview

A plug-in is a Java class that is loaded at startup time and called whenever a campaign or offer is validated. This validation occurs whenever a user saves a campaign. You can create your own Java plug-ins using the tools provided by the Validation PDK. It contains source code for sample plug-ins and an Ant file (Apache Ant is a Java-based build tool) you use to compile plug-ins. The following sections show you how to set up your environment to develop a plug-in and then walks you through the creation of your own plug-in.

Sample Validators

Two sample validators are included with the Campaign standard installation.

- `SimpleCampaignValidator` is a self-contained plug-in that shows how to do such things as custom authorization and validating allowable campaign names. It can be found in the following path:

  `Unica\campaign\devkits\validation\src\com\unica\campaign\core\validation\samples\SimpleCampaignValidator.Java`

  We recommend you make a copy of the class when working with it, rather than editing it directly.

- `ExecutableCampaignValidator` is a Java plug-in that calls out to an executable application to perform the validation. The source code for the ExecutableCampaignValidator is included in the same directory as the SimpleCampaignValidator.

  However, the real purpose of this example is for use as a command-line executable for validation. This file is located in the following path:

  ```
  Unica/Campaign/devkits/validation/src/com/unica/campaign/core/validation
  /samples/validate.sh
  ```

  This file is a sample loopback executable, illustrating common types of validation work.

Test Harness

Extreme programming and other agile methodologies are useful and popular. One important aspect of these methodologies is that they are test centric. Customers who use those methodologies use unit testing extensively. The Validation PDK supports these methodologies by offering a test harness for running a plug-in outside of Campaign. Being able to test the code without putting it into IBM Unica Campaign speeds up the plug-in Developer’s process.

To use the test harness

1. Alter the unit test case to reflect the validation logic in the plug-in.
2. Run the build script:

- To create the plug-in without performing any unit tests, run the build scripts using the “`ant jar`” command.
- To create the plug-in and also perform unit testing, run the build scripts using the “`ant run-test`” command.

Build Scripts

The build scripts in the PDK compile all of the classes in a directory and put them in a jar suitable for use in IBM Unica Campaign. The directory used by the supplied build script is the following:

```
Unica/campaign/devkits/validation/src/com/unica/campaign/core
/validation/samples/
```

Major steps to create plug-ins

These are the major steps to create plug-ins:

1. Setup
2. Build the validators
3. Configure IBM Unica Campaign
4. Test the validator configuration

5. Create a validator

The following sections will show you how to build the source to create the .jar file.

Setup

The Validation PDK can be installed on any machine, but the plug-ins you create with it must be placed on the machine running IBM Unica Campaign. We recommend installing the PDK on the machine on which you will be testing your plug-ins.

The PDK requires you to have Jakarta Ant and a Sun Java Developer Kit on your machine to create Java plug-ins. We recommend using the Ant and JDK packages that come with your application server to ensure compatibility.

You can use another JDK. However, if you use the JDK that comes with your application server, the plug-ins you create will pass compile-time compatibility checks. For example, if you are using WebLogic 8.1 which uses the 1.4 JDK to run, any classes which are specific to the 1.5 JDK would cause “class not found” errors when Campaign attempts to use the plug-in. Using a 1.4 JDK to compile will prevent the use of 1.5 JDK specific classes and ensure compatibility of the plug-in with Campaign.

To set up your environment for using the Validation PDK

1. Add the folder containing the Ant executable to your path. For example, for WebLogic 8.1 installed in the default directory on Windows, add the following to your path:

   ```
   c:\bea\weblogic81\server\bin
   ```

2. Set the `JAVA_HOME` environment variable to the directory containing the `bin` and `lib` directories of the JDK. For example, for WebLogic 8.1 on Windows, set `JAVA_HOME` to:

   ```
   c:\bea\jdk141_03
   ```

Build the validators

The PDK supplies an Ant script that can build all of the code in the sample files. The default behavior for the script is to create a jar that contains the validation classes. Optionally, it can also create Javadocs and run tests against the validators to ensure that they will work in Campaign before trying to use the plug-in in production.

To build the validator

1. Change directory to the PDK directory,

   Unica`\Campaign\devkits\validation\build`

You will see the Ant script, `build.xml`, in this directory.

2. Run the Ant jar at the command line.

Ant will run the script and produce a jar file called `validator.jar` in the directory:

`Unica``\campaign\devkits\validation\build\lib`

You now have a custom validator that can be used in IBM Unica Campaign. The next section will explain how to configure Campaign to use this validator.

Configure IBM Unica Campaign

Once you have created a validator plug-in, you must tell IBM Unica Campaign where it is so that you can use it.

To use the `SimpleCampaignValidator`, set the properties described in "Configuring the Validation PDK" on page 11 as follows:

- `validationClasspath`: *`Unica`*`\campaign\devkits\validation\lib\validator.jar`
- `validationClass`: `com.unica.campaign.core.validation.samples.SimpleCampaignValidator`
- The `validatorConfigString` does not have to be set to use the SimpleCampaignValidator because it does not use a configuration string.

Test the validator configuration

After building the `validator.jar` file that contains the `SimpleCampaignValidator` class and making the necessary configuration changes, you are ready to test and use the plug-in. This plug-in prevents users from saving a Campaign named "badCampaign."

To test your configuration

1. Redeploy your application server to have the changes take effect. For details on redeploying your application server, see your server documentation.
2. Log in to IBM Unica Campaign and navigate to the campaign creation page.
3. Create a new campaign with the name **badCampaign** and attempt to save it.

If everything is properly configured, you will not be able to save the new campaign. You should receive an error message from the validator.

Create a validator

In this section, you will create a validation plug-in that is much like the `SimpleCampaignValidator`, but prevents the creation of campaigns that are called "badCampaign2."

To create a validator

1. Make a copy of the sample validator, `SimpleCampaignValidator.java` located in `Unica\campaign\devkits\validation\src\com\unica\campaign\core\validation\samples`
2. Name the copy `MyCampaignValidator.java` and leave it in the same directory as the source.
3. Open `MyCampaignValidator.java` in an editor. Find the word "badCampaign" in the document and replace it with the word "badCampaign2."
4. Save the file and close the editor.
5. Build the validators again. For details, see "Build the validators" on page 15.

If your application server locks the `validate.jar` file while in use, you will need to stop the server before building the validators.

6. Reconfigure `campaign_config.xml` to use your new class:

```
<property name="validationClass"
value="com.unica.campaign.core.validation.samples.MyCampaignValidator">
```

7. Test the validator. For details, see "Test the validator configuration" on page 16.

You should not be able to save campaigns named "badCampaign2."

Executable Sample

The sample validators also include a validator, `ExecutableCampaignValidator`, that can run an executable from the command line.

This section:

- Shows how to set up IBM Unica Campaign to run the sample executable plug-in, and
- Describes how to create your own executable plug-ins that conform to using the executable usage interface.

Configure Campaign for the sample executable plug-in

To use the ExecutableCampaignValidator, set the properties described in "Configuring the Validation PDK" on page 11 as follows:

- validationClasspath:

  `Unica\campaign\devkits\validation\lib\validator.jar`
- validationClass:

  `com.unica.campaign.core.validation.samples.ExecutableCampaignValidator`
- validatorConfigString:

  `Unica\Campaign\pdk\bin\validate.sh`

The sample script that ships with the PDK is a Bourne shell script for UNIX. It denies campaign creation to anyone that has the username "badUser." You can view the code for that executable in this directory:

```
Unica\campaign\devkits\validation\src\com\unica\campaign\core
\validation\samples\validate.sh
```

You will need to develop your own script that performs relevant validation for your implementation. Scripting languages like PERL and Python are good candidates for text processing scripts like this, but any language that can be run from the command line is acceptable.

Expected executable usage interface

`ExecutableCampaignValidator` calls an executable with a command line that contains the following arguments:

- *`executable_name`*: this is the string set in the `validatorConfigString` in IBM Unica Marketing Platform.
- *`data_filename`*: this is the name of the file that the executable will read as input. The input data must be formatted in XML.
- *`expected_result_filename`*: this is the name of the file that the executable should send as output. The expected result are of the form `data`*`XXX`*.xml where XXX is a number.
  - Here is an example of how successful data should be sent:

  ```
  <ValidationResult result="0" generalFailureMessage="" />
  ```
  - Here is an example of how failed data should be sent:

  ```
  <ValidationResult result="1" generalFailureMessage="">
     <AttributeError attributeName="someAttribute"
  errorMessage="something" />
     <AttributeError attributeName="someAttribute2"
  ```

```
errorMessage="something2" />
</ValidationResult>
```

- Text in the XML file should be encoded in regular ASCII characters or UTF-8.

It is highly recommended that you provide easy-to-comprehend error messages to users so they can correct the problem before re-attempting another save operation.

Example

This section contains an example of a validation scenario.

Prevent campaign edits

If you are trying to prevent someone editing a campaign from changing the campaign code, you can use a custom campaign validation routine. The routine would ensure that when saving the campaign, the following check is done:

```
new_campaign_code == old_campaign_code
```

To handle the case when the campaign is first being created, pass to the routine a flag indicating whether the campaign being validated is new (creation) or existing (edit). If this flag indicates **edit**, then do the comparison of campaign codes.

The Campaign application sets this flag in the `InputValidationData` object that it then passes to the plug-in. The plug-in reads the flag when it is determining whether the validation is for a new or changed campaign.