# Unica Campaign Validation PDK Guide V12.1.5

# Contents

# Chapter 1. Validation Plug-in Developer's Kit (PDK) overview

Use the  Validation Plug-in Developer's Kit (PDK) to develop custom validation logic for use in  Unica Campaign.

You can create plug-ins to perform custom validation logic for campaigns, offers, or both.

Some possible uses of validation logic are:

- To check extended (custom) attributes
- To provide authorization services that are outside of the scope of  Unica Platform (for example, validating which users are allowed to edit which extended attributes).

The Validation PDK is a subclass of a more generic plug-in framework that is provided with Unica Campaign.

The Validation PDK contains Javadoc™ reference information for both the Plug-In API and the sample code. To view the documentation, open the following file in your web browser:

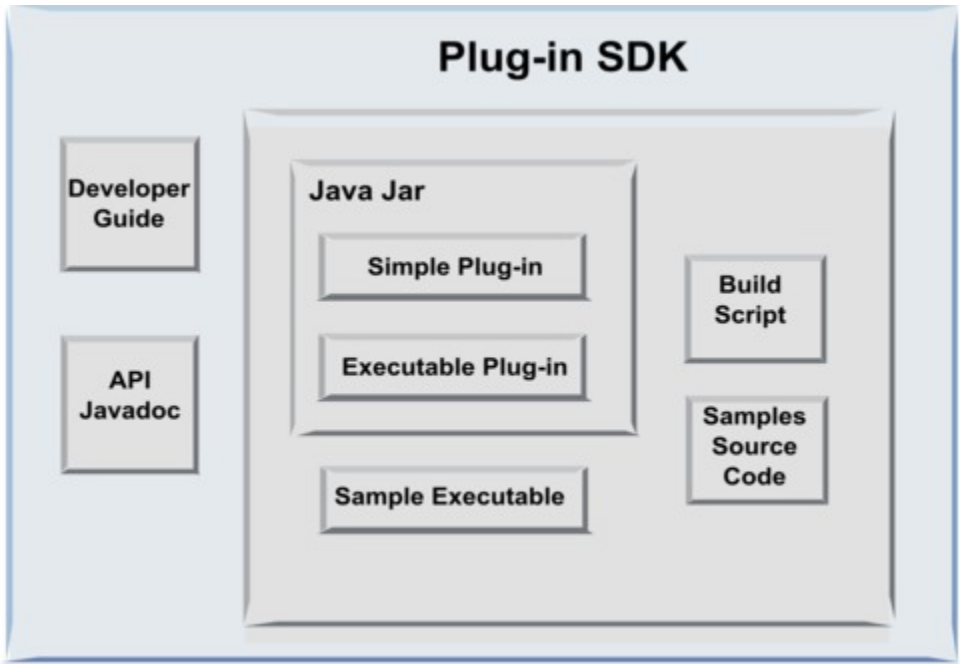
`C:\HCL\Unica\Campaign_Home\devkits\validation\javadoc\index.html`

For example:

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

## Contents of the Validation PDK

The Validation PDK contains components to develop Java™ plug-ins or command-line executables to add custom validation to  Unica Campaign. The PDK contains documented, buildable examples of how to use the PDK.




The following table describes each component.

**Table 1. Components of the Validation PDK**

| Component | Description |
|---|---|
| Developer guide | A PDF document titled *Unica Campaign Validation PDK Guide*. |
| API Javadoc™ | Reference information for the plug-in API. |
| Java™ .jar file | A sample JAR file that contains the sample plug-ins. The JAR file contains:<br>• Simple plug-in: an example of a self-contained validator class.<br>• Executable plug-in: an example validator that runs 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™. |

**Table 1. Components of the Validation PDK (continued)**

| Component | Description |
|---|---|
| Build Script | An Ant script that builds the included source code into usable validator plug-ins. |
| Samples Source Code | The Java™ source code for the simple validator and the executable validator. |

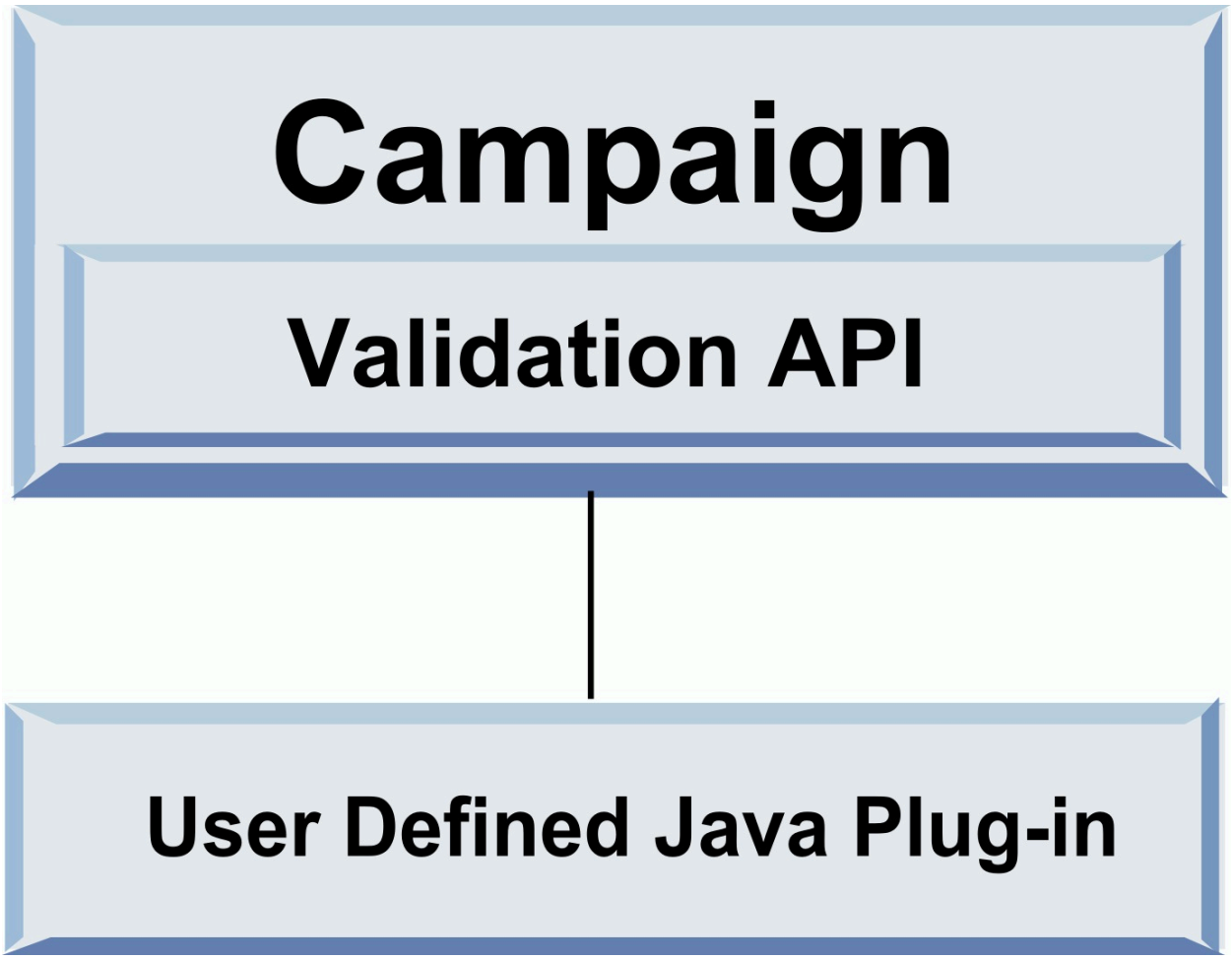
# Two ways to use the validation API

There are two ways to use the Validation 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.

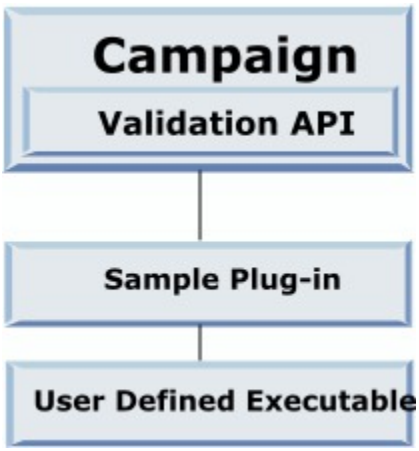
## Build a Java™ class plug-in that is loaded into the application

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




## Call an application to handle validation

You can use one of the included Validation PDK plug-ins to call out to an executable application to handle the validation.




The executable may be written in any language, but must reside on the Unica Campaign server and 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 and after values for all standard and extended attributes of that object. Unica Campaign expects results information in the form of an XML file in return.

# Offer versus campaign validation

A plug-in that is made with the Unica Campaign Validation PDK can perform custom validation logic for campaigns, offers, or both.

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

# Sample validators included in the Validation PDK

Two sample validators are included in the Unica Campaign Validation PDK: `SimpleCampaignValidator` and `ExecutableCampaignValidator`.

- `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:

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

  We recommend that you make a copy of the class before you edit it, so you can retain the original version if needed.
- `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:

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

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

  ```
  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 for the Validation PDK

Being able to test validation code without putting it into  Unica Campaign speeds up the plug-in Developer's process.

Customers who use extreme programming and other agile methodologies use unit testing extensively. The Validation PDK supports these methodologies by offering a test harness for running a plug-in outside of Unica Campaign.

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 doing any unit tests, run the build scripts using the "`ant jar`" command.
    - To create the plug-in and also do unit testing, run the build scripts using the "`ant run-test`" command.

# Build scripts for the Validation PDK

The build scripts in the Validation PDK compile all of the classes in a directory and put them in a JAR file that is suitable for use in Unica Campaign.

The supplied build script uses the following directory:

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

# Chapter 2. Developing validation plug-ins for Unica Campaign

A plug-in is a Java™ class that is loaded at startup time and called whenever a campaign or offer is validated.

The validation occurs whenever a user saves a campaign. You can create your own Java™ plug-ins with the tools that are provided in the Validation PDK. The PDK contains source code for sample plug-ins and an Ant file (Apache Ant is a Java™ based build tool) that you use to compile plug-ins.

The following steps explain how to set up your environment to develop a plug-in and then walk you through the creation of your own plug-in.

1. Setting up your environment to use the Validation PDK *(on page 8)*
2. Building the validator *(on page 9)*
3. Configuring Unica Campaign to use a validation plug-in *(on page 10)*
4. Testing the validator configuration *(on page 13)*
5. Creating a validator *(on page 14)*

## Setting up your environment to use the Validation PDK

To use the Validation PDK with Unica Campaign, you must modify your path and set the `JAVA_HOME` environment variable.

The Validation PDK can be installed on any machine, but the plug-ins that you create with it must be on the machine where Unica Campaign is running. We recommend that you install the PDK on the machine where you are testing your plug-ins.

The PDK requires you to have Apache Ant and a Sun Java™ developer kit on your machine to create Java™ plug-ins. To ensure compatibility, use the Ant and JDK packages that come with your application server.

To set up your environment to use the Validation PDK:

1. Add the folder containing the Ant executable to your path. Two examples are provided.
    - For WebLogic 11gR1 installed in the default directory on Windows™, add the following to your path: `C:\Oracle\Middleware\wlserver_10.3\common\bin`
    - For WebSphere® 7.0 installed in the default directory on Windows™, add the following to your path: `C:\HCL\WebSphere\AppServer1\bin`
2. Set the `JAVA_HOME` environment variable to the directory containing the `bin` and `lib` directories of the JDK. Two examples are provided.
    - For WebLogic 11gR1 on Windows™, set `JAVA_HOME` to `C:\Oracle\Middleware\jdk160_18`
    - For WebSphere® 7.0 on Windows™, set `JAVA_HOME` to `C:\HCL\WebSphere\AppServer1\java\jre`

# Building the validator

The  Unica Campaign Validation 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 Javadoc™ and run tests against the validators to ensure that they work in Unica Campaign before trying to use the plug-in in production.

To build the validator:

1. Change to the PDK directory `<HCL_Unica_Home\Unica Campaign_Home>\devkits\validation\build`

   For example: `C:\HCL\Unica\Campaign\devkits\validation\build`

   This directory contains the Ant script, `build.xml`.
2. Run the Ant jar at the command line.
    - To create the plug-in without doing any unit tests, use the `ant jar` command.
    - To create the plug-in and also do unit testing, use the `ant run-test` command.

Ant runs the script and produces a JAR file called `validator.jar` in the `lib` subdirectory. For example:

`C:\HCL\Unica\Campaign\devkits\validation\build\lib`

You now have a custom validator that can be used in Unica Campaign. Your next step is to configure Unica Campaign to use this validator.

# Configuring Unica Campaign to use a validation plug-in

To configure Unica Campaign to use a validation plug-in, use the configuration settings at `Unica Campaign > partitions > partition[n] > validation`.

The configuration properties tell Unica Campaign how to find the plug-in class and they provide a way to pass configuration information to the plug-ins.

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

You can adjust the following validation configuration settings:

- validationClass *(on page 11)*
- validationClasspath *(on page 11)*
- validatorConfigString *(on page 12)*

To use the `SimpleCampaignValidator`, set the properties 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.

To use the `ExecutableCampaignValidator`, set the properties as follows:

- `validationClasspath`: `<Campaign_home>\devkits\validation\lib\validator.jar`
- `validationClass`: `com.unica.campaign.core.validation.samples.ExecutableCampaignValidator`
- The `validatorConfigString`: `<Campaign_home>\pdk\bin\validate.sh`

## validationClass

The `validationClass` tells Unica Campaign the name of the class to use for validation with a Validation PDK plug-in.

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

## validationClasspath

The `validationClasspath` tells Unica Campaign the location of the class to use for validation with a Validation PDK plug-in.

| Property | Description |
|---|---|
| Description | The path to the class that is used for custom validation. |

| Property | Description |
|---|---|
| Details | Use either a full path or a relative path. If the path is relative, the behavior depends on the application server that is running Unica Campaign. WebLogic uses the path to the domain work directory, which by default is<br>`c:\bea\user_projects\domains\mydomain`.<br>If the path ends in a slash (/ for UNIX™ or \ for Windows™), Unica Campaign assumes that it points to the location of the Java™ plug-in class to be used.<br>If the path does not end in a slash, Unica Campaign assumes that it is the name of a .jar file that contains the Java™ class, as shown in the following example.<br>If the path is not set, Unica Campaign does not attempt to load a plug-in. |
| Example | `/<CAMPAIGN_HOME>/devkits/validation/lib/validator.jar`<br>This is the path on a UNIX™ platform that points to the JAR file that is packaged with the plug-in developer's kit. |
| Default | By default, no path is set:<br>`<property name="validationClasspath" />` |
| See also | See validationClass *(on page 11)* for information about designating the class to use. |

## validatorConfigString

The `validatorConfigString` is passed into the validator plug-in when it is loaded by Unica Campaign.

| Property | Description |
|---|---|
| Description | A string that is passed into the validator plug-in when it is loaded by Unica Campaign. |
| Details | How the plug-in uses this string is up to the designer. You can use it to send a configuration string into your plug-in when the system loads it.<br>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<br>`validatorConfigString`<br>to<br>`/opt/unica/campaign/devkits/validation/src/com/unica/campaign /core/validation/samples/validate.sh` |
| Default | By default, no path is set:<br>`<property name="validatorConfigString" />` |

# Testing the validator configuration

After building the `validator.jar` file that contains the `SimpleCampaignValidator` class and making the necessary configuration changes, you can test and use the plug-in.

The following plug-in example prevents Unica Campaign users from saving a campaign that is named "badCampaign."

To test your configuration:

1. Redeploy your application server so the changes take effect. For instructions, see your server documentation.
2. Log in to  Unica Campaign and go to the campaign creation page.
3. Create a campaign with the name **badCampaign** and try to save it.

If everything is properly configured, you are not able to save the new campaign. If you receive an error message from the validator, you know that it is working correctly.

# Creating a validator

Follow these instructions to create a validation plug-in that is much like the `SimpleCampaignValidator`, but prevents the creation of campaigns that are called "badCampaign2."

1. Make a copy of the sample validator `SimpleCampaignValidator.java` in `<HCL_Unica_Home\Campaign_Home>\devkits\validation\src\com \unica\campaign\core\validation\samples`. Name the copy `MyCampaignValidator.java` and leave it in the same directory as the source. For example:

   ```
   C:\HCL\Unica\Campaign\devkits\validation\src\com \unica\campaign\core\validation\samples\MyCampaignValidator.java
   ```
2. Open `MyCampaignValidator.java` in an editor. Find the word "badCampaign" in the document and replace it with the word "badCampaign2."
3. Save the file and close the editor.
4. Build the validators again. For details, see Building the validator *(on page 9)*. If your application server locks the `validate.jar` file while in use, stop the server before you build the validators.
5. Reconfigure `campaign_config.xml` to use your new class: `<property name="validationClass" value="com.unica.campaign.core.validation.samples.MyCampaignValidator">`
6. Test the validator. For details, see Testing the validator configuration *(on page 13)*.

Confirm that the validator works: You should not be able to save campaigns named "badCampaign2."

# Example validation scenario: prevent campaign edits

This example explains how to use validation to prevent specific edits to a campaign.

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

```
new_campaign_code == old_campaign_code
```

To handle the case when the campaign is first being created, pass to the routine a flag that indicates whether the campaign being validated is new (creation) or existing (edit). If this flag indicates **edit**, then compare the 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 determines whether the validation is for a new or changed campaign.

# Chapter 3. Calling an application to handle validation

The Validation PDK includes a sample validator, `ExecutableCampaignValidator`, which runs an executable, `validate.sh`, from the command line, to perform validation.

The following sections explain how to:

- Configure Unica Campaign to run the sample executable plug-in, and
- Create your own executable plug-in that conforms to using the executable usage interface.

## Configuring Unica Campaign to use the sample executable plug-in

To use the `ExecutableCampaignValidator`, adjust the configuration settings at `Unica Campaign > partitions > partition[n] > validation`.

Set the properties as follows:

- validationClasspath:

  ```
  <Unica Campaign_home>\devkits\validation\lib\validator.jar
  ```

- validationClass:

  ```
  com.unica.campaign.core.validation.samples.ExecutableCampaignValidator
  ```

- validatorConfigString:

  ```
  <Unica Campaign_home>\pdk\bin\validate.sh
  ```

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

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

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

# Expected executable usage interface

The `ExecutableCampaignValidator` plug-in calls an executable file with a command line that contains the following arguments.

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

      ```
      <ValidationResult result="0" generalFailureDeliver="" />
      ```

    - Here is an example of how failed data is sent:

      ```
      <ValidationResult result="1" generalFailureDeliver="">
       <AttributeError attributeName="someAttribute"
       errorMessage="something" />
       <AttributeError attributeName="someAttribute2"
       errorMessage="something2" />
       </ValidationResult>
      ```

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

**Note:** It is highly recommended that you provide easy-to-understand error messages to users so they can correct the problem before reattempting another save operation.