Unica Campaign – Validierungs-PDK-Handbuch Version 12.1

Inhalt

Kapitel 1. Übersicht Developer's Kit für das Validierungs-Plug-in (PDK)

Mit dem Developer's Kit für das Validierungs-Plug-in (PDK) können Sie eine benutzerdefinierte Validierungslogik für die Verwendung in Unica Campaign entwickeln.

Sie können Plug-ins erstellen, um eine benutzerdefinierte Validierungslogik für Kampagnen und/oder Angebote auszuführen.

Mögliche Einsatzbereiche der Validierungslogik sind zum Beispiel:

- Überprüfung von erweiterten (benutzerdefinierten) Attributen
- Bereitstellung von Autorisierungsservices, die nicht von Unica Platform angeboten werden (zum Beispiel die Validierung, welche Benutzer welche erweiterten Attribute bearbeiten dürfen).

Das Validierungs-PDK ist eine Unterklasse der von Unica Campaign bereitgestellten allgemeinen Plug-ins.

Das Validierungs-PDK enthält Javadoc™-Referenzinformationen sowohl für die Plug-in-API als auch für den Beispielcode. Öffnen Sie die folgende Datei in Ihrem Web-Browser, um die Dokumentation anzuzeigen:

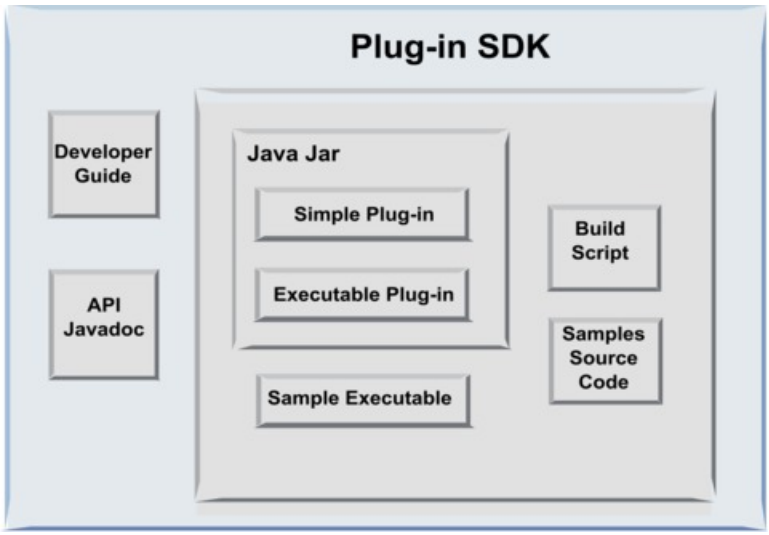
`C:\HCL\Unica\`*`Campaign_Home`*`\devkits\validation\javadoc\index.html`

Beispiel:

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

Inhalt des Validierungs-PDK

Das Validierungs-PDK beinhaltet Komponenten für die Entwicklung von Java™-Plug-ins oder über die Befehlszeile ausführbaren Funktionen, um eine benutzerdefinierte Validierung in Unica Campaign hinzuzufügen. Das PDK enthält dokumentierte Beispiele für die Verwendung des PDK, auf denen Sie aufbauen können.



Die einzelnen Komponenten werden in der folgenden Tabelle beschrieben.

Tabelle 1. Komponenten des Validierungs-PDK

Komponente	Beschreibung
Handbuch für Entwickler	PDF-Dokument mit dem Titel *Unica Campaign - Handbuch für das Validierungs-PDK*.
API Javadoc	Referenzinformationen für die Plug-in-API.
Java .jar-Datei	JAR-Beispieldatei mit Beispiel-Plug-ins. Die JAR-Datei enthält: • Einfaches Plug-in: Beispiel einer eigenständigen Validatorklasse. • Ausführbares Plug-in: Beispielvalidator, der für die Validierung eine benutzerdefinierte ausführbare Funktion ausführt, die über die Befehlszeile aufgerufen werden kann.
Beispiel einer ausführbaren Funktion	Über die Befehlszeile ausführbare Funktion, die zusammen mit dem ausführbaren Plug-in unter UNIX™ verwendet werden kann.
Erstellungsscript	Ant-Script, das aus dem enthaltenen Quellcode verwendbare Validator-Plug-ins erstellt.
Beispielquellcode	Java-Quellcode für den einfachen Validator und den ausführbaren Validator.

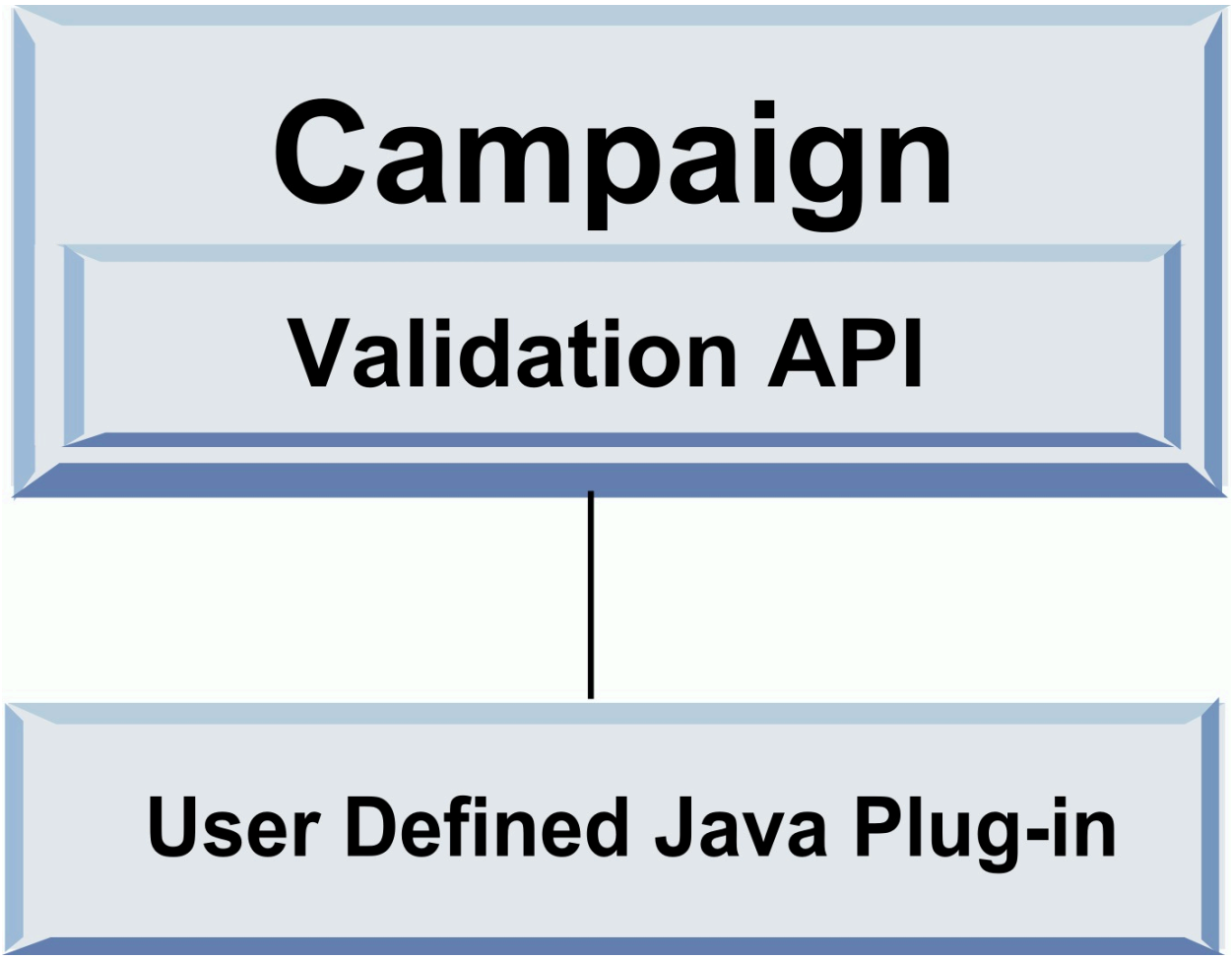
Zwei Möglichkeiten für die Verwendung der Validierungs-API

Es gibt zwei Möglichkeiten für die Verwendung der Validierungs-API.

- Verwendung der API für die Erstellung eines Java-Klasse-Plug-ins, das in die Anwendung geladen wird.
- Verwendung eines der enthaltenen Plug-ins für den Aufruf einer ausführbaren Anwendung zur Durchführung der Validierung.

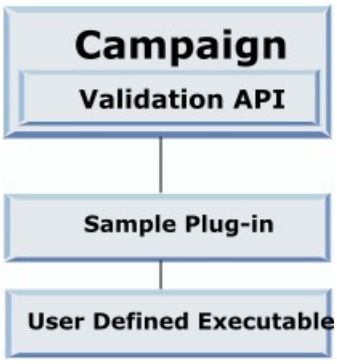
Erstellen eines Java-Klasse-Plug-ins, das in die Anwendung geladen wird

Das Validierungs-PDK bietet die Benutzeroberflächen, Helper-Klassen und Entwicklertools für die Entwicklung dieser Klassen.



Aufrufen einer Anwendung zur Ausführung der Validierung

Sie können eines der enthaltenen Plug-ins des Validierungs-PDK verwenden, um eine ausführbare Anwendung aufzurufen, mit der die Validierung durchgeführt werden soll.



Die ausführbare Funktion kann in einer beliebigen Sprache geschrieben sein, muss sich aber auf dem Unica Campaign-Server befinden und dort ausgeführt werden. Das Plug-in, mit dem die ausführbare Funktion aufgerufen wird, sendet eine XML-Datei mit den zu validierenden Informationen. Das sind zum Beispiel der Bearbeiter des Objekts oder die Vorher-/Nachher-Werte für alle Standardattribute und erweiterten Attribute des betreffenden Objekts. Unica Campaign erwartet im Gegenzug Ergebnisinformationen in Form einer XML-Datei.

Validierung von Angeboten/Kampagnen

Ein mit dem Validierungs-PDK von Unica Campaign erstelltes Plug-in kann eine benutzerdefinierte Validierungslogik für Kampagnen und/oder Angebote ausführen.

Das Validierungs-PDK wird zur Validierung von Angeboten und Kampagnen verwendet. Wenn ein Validierungs-Plug-in definiert wird, wird es bei Unica Campaign jedes Mal automatisch aufgerufen, wenn ein Angebot oder ein Kampagnenobjekt gespeichert wird. Unica Campaign setzt ein Flag, wenn es die Validierungsmethode des Plug-ins aufruft. Unica Campaign übergibt die folgenden Flags:

- `ValidationInputData.CAMPAIGN_VALIDATION`, beim Hinzufügen oder Ändern einer Kampagne

  oder
- `ValidationInputData.OFFER_VALIDATION`, beim Hinzufügen oder Bearbeiten eines Angebots.

Mit diesen Flags können Validierungsregeln für Angebote und Kampagnen aufgestellt werden.

Beispielvalidatoren im Validierungs-PDK

Zwei Beispielvalidatoren sind im Unica CampaignValidierungs-PDK enthalten: `SimpleCampaignValidator` und `ExecutableCampaignValidator`.

- `SimpleCampaignValidator` ist ein eigenständiges Plug-in beispielsweise für die benutzerdefinierte Autorisierung oder für die Validierung zulässiger Kampagnennamen. Es befindet sich im folgenden Pfad:

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

  Es wird empfohlen, vor der Bearbeitung der Klasse eine Kopie anzulegen, damit die ursprüngliche Version bei Bedarf zur Verfügung steht.
- `ExecutableCampaignValidator` ist ein Java-Plug-in, das eine ausführbare Anwendung aufruft, um die Validierung auszuführen. Der Quellcode für ExecutableCampaignValidator befindet sich im selben Verzeichnis wie der Quellcode von SimpleCampaignValidator:

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

  Der eigentliche Zweck dieses Beispiels ist jedoch die Verwendung als in der Befehlszeile aufrufbare Validierungsfunktion. Diese Datei befindet sich in folgendem Pfad:

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

  Bei dieser Datei handelt es sich um ein Beispiel für eine ausführbare Loopbackdatei, mit der häufige Validierungsaktionen demonstriert werden.

Testharness für das Validierungs-PDK

Durch die Möglichkeit, den Gültigkeitscode testen zu können, ohne dass dieser in Unica Campaign geladen werden muss, lässt sich der Entwicklungsprozess für das Plug-in beschleunigen.

Kunden, die Extreme Programming und andere agile Methodiken nutzen, führen in der Regel sehr viele Komponententests durch. Das Validierungs-PDK unterstützt diese Methodiken mit

dem bereitgestellten Testharness für die Ausführung eines Plug-ins außerhalb von Unica Campaign.

Vorgehensweise für die Anwendung des Testharness:

1. Ändern Sie den Komponententest so ab, dass die Validierungslogik im Plug-in berücksichtigt wird.
2. Führen Sie das Erstellungsscript aus:
    - Wenn Sie das Plug-in ohne Komponententests erstellen möchten, führen Sie die Erstellungsscripts mit dem Befehl "`ant jar`" aus.
    - Wenn Sie das Plug-in erstellen und auch Komponententests durchführen möchten, führen Sie die Erstellungsscripts mit dem Befehl "`ant run-test`" aus.

Erstellungsscripts für das Validierungs-PDK

Mit den Erstellungsscripts im Validierungs-PDK werden alle Klassen in einem Verzeichnis kompiliert und in einer JAR-Datei abgelegt, die sich zur Verwendung in Unica Campaign eignet.

Das bereitgestellte Erstellungsscript befindet sich in folgendem Verzeichnis:

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

Kapitel 2. Entwickeln von Validierungs-Plug-ins für Unica Campaign

Ein Plug-in ist eine Java-Klasse, die beim Start geladen und bei jeder Validierung einer Kampagne oder eines Angebots aufgerufen wird.

Die Validierung erfolgt jedes Mal, wenn ein Benutzer eine Kampagne speichert. Sie können Ihre eigenen Java-Plug-ins mit den Tools erstellen, die mit dem Validierungs-PDK bereitgestellt werden. Das PDK enthält Quellcode für Beispiel-Plug-ins und eine Ant-Datei (Apache Ant ist ein Erstellungstool auf Java-Basis) für die Kompilierung von Plug-ins.

In den folgenden Schritten wird erläutert, wie Sie Ihre Umgebung für die Entwicklung eines Plug-ins einrichten können. Anschließend werden Sie durch die Erstellung eines eigenen Plug-ins geführt.

Konfiguration der Umgebung für die Verwendung des Validierungs-PDK

Damit Sie das Validierungs-PDK mit Unica Campaignverwenden können, müssen Sie Ihren Pfad ändern und die Umgebungsvariable `JAVA_HOME`festlegen.

Das Validierungs-PDK kann in jedem beliebigen System installiert werden, die damit erstellten Plug-ins müssen sich jedoch in dem System befinden, in dem Unica Campaign ausgeführt wird. Es wird empfohlen, das PDK in dem System zu installieren, in dem Sie Ihre Plug-ins testen.

Damit das PDK Java-Plug-ins erstellen kann, müssen in dem System Apache Ant und ein Sun JavaDeveloper Kit installiert sein. Um die Kompatibilität sicherzustellen, sollten Sie die Ant- und JDK-Pakete verwenden, die mit Ihrem Anwendungsserver ausgeliefert wurden.

Konfiguration Ihrer Umgebung für das Validierungs-PDK:

1. Stellen Sie den Ordner mit der ausführbaren Ant-Datei in Ihren Pfad. Beispiele:
   - Bei einer Installation von WebLogic 11gR1 im Standardverzeichnis unter Windows™fügen Sie Ihrem Pfad Folgendes hinzu: `C:\Oracle\Middleware\wlserver_10.3\common\bin`
   - Bei einer Installation von WebSphere®7.0 im Standardverzeichnis unter Windowsfügen Sie Ihrem Pfad Folgendes hinzu: `C:\HCL\WebSphere\AppServer1\bin`
2. Geben Sie für die Umgebungsvariable `JAVA_HOME`das Verzeichnis mit den `bin`und `lib`-Verzeichnissen Ihres JDK an. Beispiele:
   - Bei WebLogic 11gR1 unter Windows, setzen Sie `JAVA_HOME` auf `C:\Oracle\Middleware\jdk160_18`
   - Bei WebSphere 7.0 unter Windows, setzen Sie `JAVA_HOME` auf `C:\HCL\WebSphere\AppServer1\java\jre`

Erstellen des Validators

Das Unica Campaign-Validierungs-PDK stellt ein Ant-Script bereit, mit dem sich der gesamte Code in den Beispieldateien erstellen lässt.

Standardverhalten des Scripts ist die Erstellung einer JAR-Datei mit den Validierungsklassen. Optional kann Javadocerstellt und Tests der Validatoren ausgeführt werden, um vor der Verwendung des Plug-ins in der Produktionsumgebung sicherzustellen, dass die Validatoren in Unica Campaignfunktionsfähig sind.

Vorgehensweise zum Erstellen des Validators:

1. Wechseln Sie in das Verzeichnis PDK. `<HCL_Unica_Home\Unica Campaign_Home>\devkits\validation\build`

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

Dieses Verzeichnis enthält das Ant-Script `build.xml`.

2. Führen Sie die Ant-JAR-Datei in der Befehlszeile aus.
   - Zur Erstellung des Plug-ins ohne Komponententests verwenden Sie den Befehl `ant jar`.
   - Zur Erstellung des Plug-ins mit Komponententests verwenden Sie den Befehl `ant run-test`.

   Das Ant-Script wird ausgeführt und erstellt eine JAR-Datei mit dem Namen `validator.jar`im Unterverzeichnis `lib`. Beispiel:

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

Sie haben nun einen benutzerdefinierten Validator für die Verwendung in Unica Campaign erstellt. Im nächsten Schritt müssen Sie Unica Campaign für die Verwendung dieses Validators konfigurieren.

Konfigurieren von Unica Campaign für die Verwendung eines Validierungs-Plug-ins

Für die Konfiguration von Unica Campaignzur Verwendung eines Validierungs-Plug-ins müssen Sie die Konfigurationseinstellungen unter `Unica Campaign > partitions > partition[n] > validation`verwenden.

Mit den Konfigurationseigenschaften wird Unica Campaign informiert, wie die Plug-in-Klasse gefunden werden kann, und sie bieten eine Möglichkeit zur Übergabe der Konfigurationsinformationen an die Plug-ins.

Anmerkung: Die Validierung kann mit verschiedenen Partitionen arbeiten; `partition[n]`kann auf einen beliebigen Partitionsnamen geändert werden, um auch für diese Partitionen Validierungsroutinen bereitzustellen.

Sie können die folgenden Konfigurationseinstellungen für die Validierung anpassen:

- validationClass *(auf Seite 11)*

- validationClasspath *(auf Seite 12)*
- validatorConfigString *(auf Seite 13)*

Zur Verwendung von `SimpleCampaignValidator`, stellen Sie die Eigenschaften wie folgt ein:

- `validationClasspath`: *`Unica`*`\campaign\devkits\validation\lib\validator.jar`
- `validationClass`: `com.unica.campaign.core.validation.samples.SimpleCampaignValidator`
- `validatorConfigString` muss für die Verwendung von SimpleCampaignValidator nicht festgelegt werden, da dieser keine Konfigurationszeichenfolge verwendet.

Zur Verwendung von `ExecutableCampaignValidator`, stellen Sie die Eigenschaften wie folgt ein:

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

validationClass

Mit der Eigenschaft `validationClass` wird Unica Campaign der Name der Klasse übermittelt, die für die Validierung mit einem Plug-in des Validierungs-PDK verwendet werden muss.

Eigenschaft	Beschreibung
Beschreibung	Name der Klasse, die für die Validierung verwendet werden soll. Der Wert der Eigenschaft `validationClasspath` gibt die Position dieser Klasse an.
Details	Die Klasse muss mit dem Paketnamen vollständig qualifiziert sein. Wenn diese Eigenschaft nicht festgelegt ist, führt Unica Campaign keine benutzerdefinierte Validierung durch.
Beispiel	`com.unica.campaign.core.validation.samples.SimpleCampaignValidator`

Eigenschaft	Beschreibung
	In diesem Beispiel wird `validationClass` auf die Klasse `SimpleCampaignValidator` aus dem Beispielcode gesetzt.
Standard	Standardmäßig wird kein Pfad festgelegt: `<property name="validationClass" />`

validationClasspath

Mit der Eigenschaft `validationClasspath` wird Unica Campaign der Name der Klasse übermittelt, die für die Validierung mit einem Plug-in des Validierungs-PDK verwendet werden muss.

Eigenschaft	Beschreibung
Beschreibung	Pfad der Klasse, die für die benutzerdefinierte Validierung verwendet wird.
Details	Verwenden Sie entweder einen vollständigen Pfad oder einen relativen Pfad. Bei einem relativen Pfad ist das Verhalten von dem Anwendungsserver abhängig, auf dem Unica Campaign ausgeführt wird. WebLogic verwendet den Pfad zum Domänenarbeitsverzeichnis, das Standardverzeichnis ist: `c:\bea\user_projects\domains\mydomain`. Wenn der Pfad mit einem Schrägstrich endet (normaler Schrägstrich (/) bei UNIXbzw. umgekehrter Schrägstrich (\) bei Windows), geht Unica Campaign davon aus, dass er auf die Position der zu verwendenden Java-Plug-in-Klasse zeigt. Wenn der Pfad nicht mit einem Schrägstrich endet, geht Unica Campaign davon aus, dass dies der Name einer JAR-Datei ist, welche die Java-Klasse enthält (siehe folgendes Beispiel). Wenn kein Pfad festgelegt ist, versucht Unica Campaign nicht, ein Plug-in zu laden.
Beispiel	`/<CAMPAIGN_HOME>/devkits/validation/lib/validator.jar`

Eigenschaft	Beschreibung
	Dies ist auf einer UNIX-Plattform der Pfad zu der JAR-Datei, die im Paket des Developer's Kit für Plug-ins enthalten ist.
Standard	Standardmäßig wird kein Pfad festgelegt: `<property name="validationClasspath" />`
Siehe auch	Siehe validationClass *(auf Seite 11)*; hier finden Sie Informationen zur Benennung der zu verwendenden Klasse.

validatorConfigString

Die Eigenschaft `validatorConfigString` wird an das Validator-Plug-in übergeben, wenn es von Unica Campaign geladen wird.

Eigenschaft	Beschreibung
Beschreibung	Zeichenfolge, die an das Validator-Plug-in übergeben wird, wenn es von Unica Campaign geladen wird.
Details	Wie das Plug-in diese Zeichenfolge verwendet, wird durch den Entwickler festgelegt. Sie können zum Beispiel festlegen, dass eine Zeichenfolge an Ihr Plug-in übergeben wird, wenn es vom System geladen wird. Zum Beispiel verwendet der `ExecutableCampaignValidator` (im PDK enthaltenes Beispiel-Plug-in für eine ausführbare Funktion) diese Eigenschaft, um die ausführbare Funktion anzugeben, die aufgerufen werden soll.

Beispiel	Um das Beispiel-Bourne-Shell-Script als Validierungsscript auszuführen, legen Sie
	`validatorConfigString`
	auf
	`/opt/unica/campaign/devkits/validation/src/com/unica/campaign /core/validation/samples/validate.sh`

Standard	Standardmäßig wird kein Pfad festgelegt: `<property name="validatorConfigString" />`

Testen der Validatorkonfiguration

Nach Erstellen der Datei `validator.jar`, welche die Klasse `SimpleCampaignValidator` enthält, und nach den erforderlichen Konfigurationsänderungen können Sie das Plug-in testen und verwenden.

Mit dem folgenden Beispiel-Plug-in wird verhindert, dass Benutzer von Unica Campaign eine Kampagne namens "badCampaign" speichern.

Vorgehensweise zum Testen Ihrer Konfiguration:

1. Implementieren Sie Ihren Anwendungsserver erneut, damit die Änderungen wirksam werden. Sie finden entsprechende Anweisungen in der Dokumentation Ihres Servers.
2. Melden Sie sich bei Unica Campaign an und rufen Sie die Seite für die Kampagnenerstellung auf.
3. Erstellen Sie eine Kampagne mit dem Namen **badCampaign** und versuchen Sie, diese zu speichern.

Wenn alles richtig konfiguriert ist, können Sie die neue Kampagne nicht speichern. Die Konfiguration funktioniert ordnungsgemäß, wenn Sie eine Fehlernachricht vom Validator erhalten.

Erstellen eines Validators

Befolgen Sie diese Anweisungen, um ein Validierungs-Plug-in ähnlich `SimpleCampaignValidator`zu erstellen, mit dem die Erstellung von Kampagnen namens "badCampaign2." verhindert wird.

1. Machen Sie eine Kopie des Mustervalidators `SimpleCampaignValidator.java` in `<HCL_Unica_Home\Campaign_Home>\devkits\validation\src\com\unica\campaign\core\validation\samples`. Geben Sie der Kopie den Namen `MyCampaignValidator.java`und belassen Sie sie im selben Verzeichnis wie die Quelle. Beispiel:

```
C:\HCL\Unica\Campaign\devkits\validation\src\com \unica\campaign
\core\validation\samples\MyCampaignValidator.java
```

2. `MyCampaignValidator.java` im Editor öffnen Suchen Sie das Wort "badCampaign" im Dokument und ersetzen Sie es durch das Wort "badCampaign2".
3. Speichern Sie die Datei und schließen Sie den Editor.
4. Erstellen Sie die Validatoren erneut. Details hierzu finden Sie in Erstellen des Validators *(auf Seite 9)*. Wenn Ihr Anwendungsserver die Datei `validate.jar`sperrt, während sie verwendet wird, müssen Sie den Server vor der Erstellung der Validatoren stoppen.
5. Konfigurieren Sie `campaign_config.xml`für die Verwendung Ihrer neuen Klasse neu: `<property name="validationClass" value="com.unica.campaign.core.validation.samples.MyCampaignValidator">`
6. Testen Sie den Validator. Details hierzu finden Sie in Testen der Validatorkonfiguration *(auf Seite 13)*.

Überprüfen Sie, ob der Validierer funktioniert: Es darf nicht möglich sein, Kampagnen mit dem Namen "badCampaign2" zu speichern.

Beispiel für ein Validierungsszenario: Kampagnenbearbeitungen verhindern

In diesem Beispiel wird erläutert, wie eine bestimmte Bearbeitung einer Kampagne mit der Validierung verhindert werden kann.

Wenn Sie verhindern möchten, dass ein Bearbeiter den Kampagnencode ändert, können Sie eine benutzerdefinierte Routine für die Kampagnenvalidierung verwenden. Durch die Routine wird beim Speichern der Kampagne die folgende Prüfung vorgenommen:

```
new_campaign_code == old_campaign_code
```

Damit auch neu erstellte Kampagnen berücksichtigt werden, müssen Sie ein Flag an die Routine übergeben, das anzeigt, ob die zu validierende Kampagne neu ist (creation) oder ob sie bereits existiert (edit). Zeigt dieses Flag **edit** an, werden die Kampagnencodes verglichen.

Die Campaign-Anwendung setzt dieses Flag im Objekt `InputValidationData`, von dem es dann an das Plug-in übergeben wird. Das Plug-in liest das Flag bei der Ermittlung, ob die Validierung für eine neue oder für eine geänderte Kampagne vorgenommen werden soll.

Kapitel 3. Aufruf einer Anwendung zur Durchführung der Validierung

Das Validierungs-PDK enthält den Beispielvalidator `ExecutableCampaignValidator`. Dieser führt für die Validierung über die Befehlszeile die ausführbare Datei `validate.sh`aus.

In den folgenden Abschnitten werden diese Vorgänge erläutert:

- Konfiguration von Unica Campaign für die Ausführung des ausführbaren Beispiel-Plug-ins und
- Erstellung Ihres eigenen ausführbaren Plug-ins, das sich für die Verwendung der Benutzeroberfläche für ausführbare Funktionen eignet.

Konfigurieren von Unica Campaign für die Verwendung des ausführbaren Beispiel-Plug-ins

Zur Verwendung der `ExecutableCampaignValidator` , passen Sie die Konfigurationseinstellungen bei `Unica Campaign > partitions > partition[n] > validation` an.

Stellen Sie die Eigenschaften wie folgt ein:

- 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
  ```

Das mit dem Validierungs-PDK bereitgestellte Beispielscript ist ein Bourne-Shell-Script für UNIX. Damit wird allen Benutzern mit dem Benutzernamen "badUser" die Erstellung von

Kampagnen verweigert. Sie können den Code für diese ausführbare Funktion im folgenden Verzeichnis anzeigen:

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

Sie müssen Ihr eigenes Script entwickeln, das die Ihrer Implementierung entsprechende Validierung durchführt. Für Textverarbeitungsscripts wie dieses eignen sich scriptbasierte Sprachen wie PERL und Python gut, wobei aber alle Sprachen akzeptabel sind, die über die Befehlszeile ausgeführt werden können.

Erwartete Benutzeroberfläche für ausführbare Funktionen

Das Plug-in `ExecutableCampaignValidator`ruft über eine Befehlszeile mit den unten genannten Argumenten eine ausführbare Datei auf.

- `executable_name`: Die Zeichenfolge, die in `validatorConfigString` in Unica Platform festgelegt ist.
- `data_filename`: Name der Datei, die von der ausführbaren Funktion als Eingabe gelesen wird. Die Eingabedaten müssen in XML formatiert sein.
- `expected_result_filename`: Name der Datei, die von der ausführbaren Funktion als Ausgabe gesendet werden soll. Die erwarteten Ergebnisse haben die Form `dataXXX`.xml, wobei XXX eine Zahl ist.
    - Beispiel für erfolgreich gesendete Daten:

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

    - Beispiel für nicht erfolgreich gesedete Daten:

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

```
</ValidationResult>
```

- Der Text in der XML-Datei muss mit normalen ASCII-Zeichen oder mit UTF-8 codiert sein.

Anmerkung: Es wird dringend empfohlen, leicht verständliche Fehlernachrichten zu definieren, damit Benutzer das Problem beheben können, bevor sie erneut versuchen, eine Sicherungsoperation durchzuführen.