**Abstract**

The EPG Custom Rule SDK is responsible for implementing a programmable rule with additional custom functions. Custom Rules could be packaged into a jar file and be loaded automatically when the app server starts after it is deployed to the application server.

Custom Rule is one class designed to execute SQL to update data in DB or memory DB to do some quick update.

This guide introduces how to create and customize a Custom Rule demo project programmatically.

**Contents**

[1 Revision Information 3](#_Toc437619533)

[2 Background 3](#_Toc437619534)

[3 Overview 3](#_Toc437619535)

[4 Quick Start 3](#_Toc437619536)

[5 SDK Guide 6](#_Toc437619537)

[5.1 Interface 6](#_Toc437619538)

[5.2 Rule Types 6](#_Toc437619539)

[5.3 Parameter Types 6](#_Toc437619540)

[5.4 Custom Rule Metadata 7](#_Toc437619541)

[5.5 Exceptions 8](#_Toc437619542)

[5.6 Hint 9](#_Toc437619543)

[5.7 Package Structure 9](#_Toc437619544)

[6 FAQ 9](#_Toc437619545)

# Revision Information

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Prepared** | **Information** |
| PA1 | 12/02/15 | Anderson Lu | Initial Version |

# Background

The Custom Rules are running on Ericsson CMS EPG since version 5.0. And it is a kind of EPG Rule with additional custom functions.

# Overview

Custom Rule SDK supports to create a custom rule by implementing a specified interface in the SDK. And it supports to add additional custom functions in a custom rule.

Custom Rules are written in Java and packaged as a jar file and deployed the CMS Application server in a folder “/opt/tandbergtv/cms/plugins/epgmgmt/rules”.

# Quick Start

**Prerequisite**: Maven 3.0+ and JDK 7 must be installed firstly on the system.

cms\_epgmgmt\_custom\_rule\_interface-5.1.000.jar

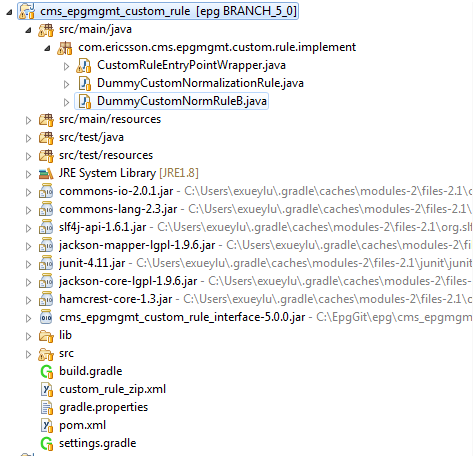
Execute the following steps to build and deploy your custom rule:

1. Extract cms\_epgmgmt\_custom\_rule\_sdk.zip.

In Windows, unzip it. In \*nix OS, run shell command unzip optionally.

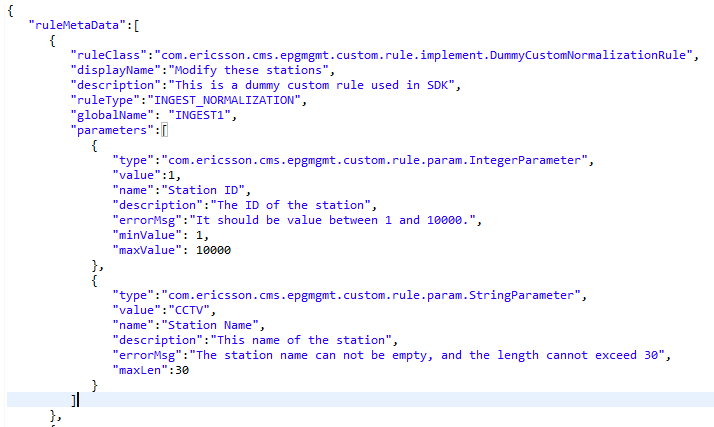
Run “**mvn clean eclipse: eclipse**” to generate the Eclipse configuration files.

1. Import the demo project to your IDE. The structure should be like below:



1. Define custom rule metadata

All custom rule information is defined in a Json file named “custom-rule-definitions.js” under path “src/main/resources”. The format of the file is like below. For more details see section 5.4.



1. Customize your own rule logic by create a special class.

A Custom Rule is class which extends base class BaseCustomNormalizationRule. And override its two methods, one is innerExecute () and the another is validate (). Method innerExecute () is designed to customize your own rule logic and method validate () is designed to validate the rule metadata. For more detail information, see section 5.

1. Implement interface CustomRuleEntryPoint. We give the demo code for you and you can modify it as required.
2. Package the demo project to a jar or a zip.

For this step, we suggest that you use maven to build your package. Please run “**mvn clean package**”.

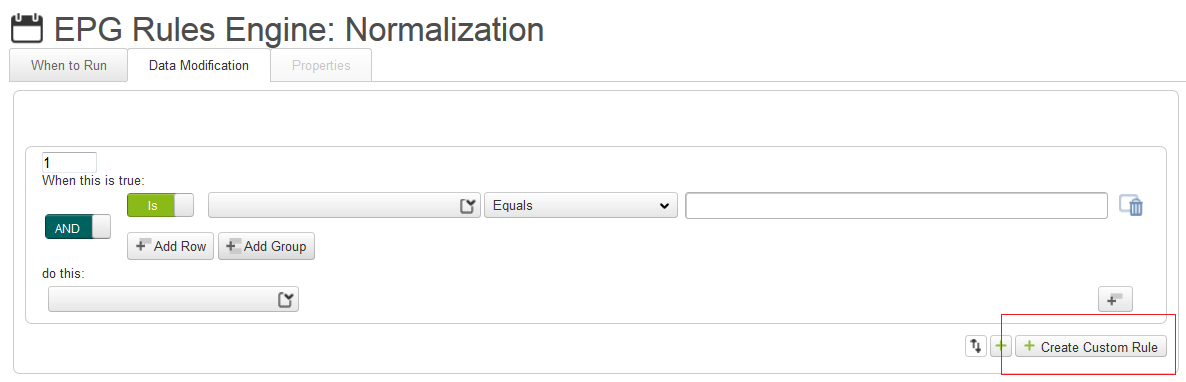
1. Deploy to your application server.

Copy the jar to your application server, and the path to deploy is “/opt/tandbergtv/cms/plugins/epgmgmt/rules”.

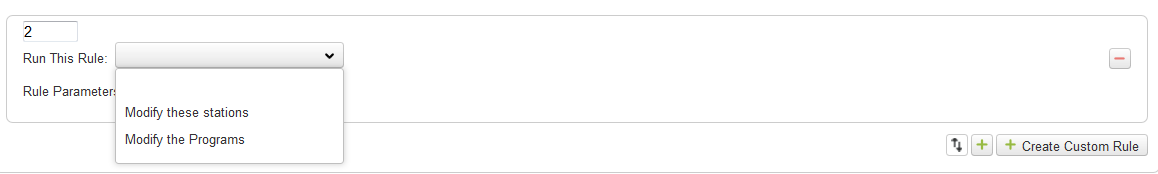
1. Restart application server to load your custom rules.

**# service cms restart**

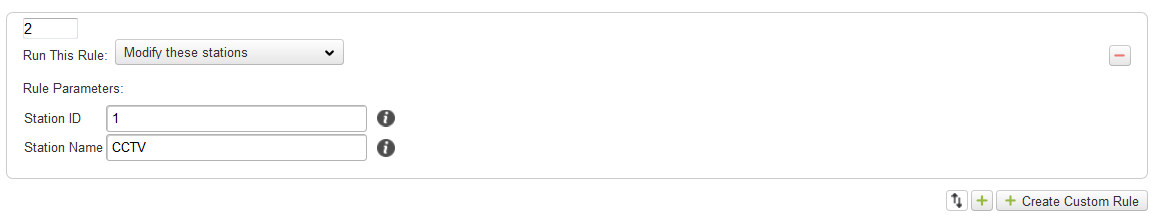
1. Now your custom rule has been loaded automatically. And you can use it in CMS.
2. Log on to the system and select **EPG>Rule>Normalization**, click **Create Custom Rule** button.



1. Select the desired custom rule in the **Run This Rule** drop-down list.



1. Enter the **Rule Parameter** information, as desired.

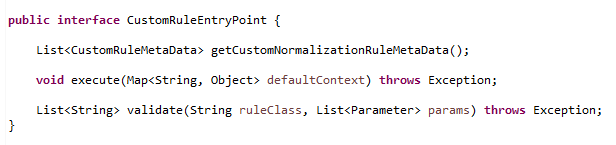


# SDK Guide

## Interface

Interface CustomRuleEntryPoint is the entry point interface that handles your custom rule logic. Therefore you have to add an entry point class (also called “CustomRuleEntryPointWrapper” in the demo project) to inherit your custom rule logic as required. And a custom rule class extends this entry point class.

The following is the source code of the interface:



## Rule Types

Four rule types are supported. All rule types are defined in enum CustomRuleType.

|  |  |
| --- | --- |
| **Name** | **Description** |
| VALIDATION | The custom rule is a rule for validation. |
| INGEST\_NORMALIZAITON | The custom rule is a normalization rule when ingesting. |
| EXPORT\_NORMALIZATION | The custom rule is a normalization rule when exporting. |
| ALL\_NORMALIZATION | The custom rule is a normalization rule when ingesting and exporting. |

## Parameter Types

Three kinds of parameter types are supported. And we will extend in the future version.

| **NAME** | **Description** |
| --- | --- |
| IntegerParameter | Type for integer parameter. |
| StringParameter | Type for string. |
| DateTimeParameter | Type for date time format. |

## Custom Rule Metadata

1. All custom rule metadata is defined in a Json file “custom-rule-definitions.js” under path “src/main/resources”.

All the data is under scope below:

{

“ruleMetaData”: [

//Rule Metadata is defined here

]

}

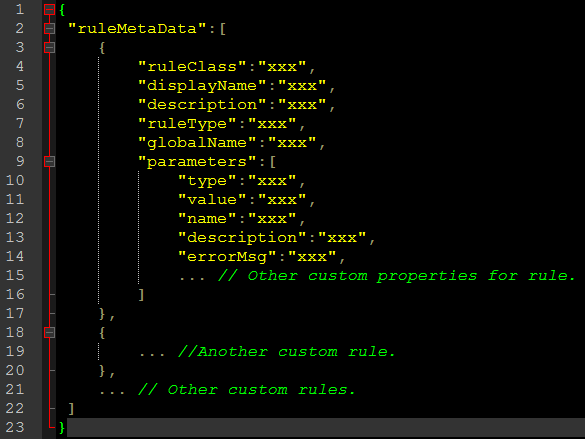
1. Necessary parameter for rule metadata:

|  |  |
| --- | --- |
| **NAME** | **Description** |
| ruleClass | Defines the class to handle your custom rule logic. |
| displayName | Defines the name that displays in the CMS UI. |
| description | Defines description for custom rule. |
| ruleType | Defines the type of the rule. See section 5.2. |
| globalName | Defines the global name. See section 4 below. |
| parameters | Defines parameters. |

1. Necessary arguments for parameters:

|  |  |
| --- | --- |
| **NAME** | **Description** |
| type | Defines the parameter type, see section 5.3. |
| value | Defines the default value of the parameter. User can change it in UI |
| name | Defines the name of the parameter. |
| description | Defines the description. |
| errMsg | Defines the error message when validated failed. It will be shown in UI when UI validation Fails. |
| [other optional properties] | Defines your custom properties. |

1. A prototype for rule metadata:



Note：In order to make sure that the global name is unique in EPG, the value of global name consists of two parts, one is official prefix that we provide and the other is your custom value. For example, we provide a prefix “G01”, and then you can define the global name like “G01xxx”. So please ask us and we will give you a prefix string value.

Important: Please ask us for getting a prefix string for custom rule property “globalName”.

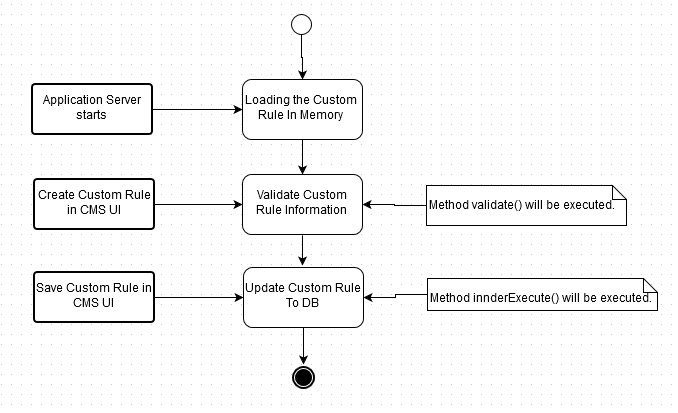
## Exceptions

Using Custom Rule SDK, you may catch the following two exceptions.

1. CustomRuleExecutionException
2. CustomRuleValidationException

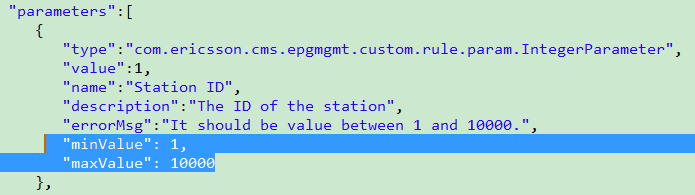
When the method innderExecute () fails, it throws CustomRuleExecutionException. And when the method validate () fails, it throws CustomRuleValidationException.

## Hint



When saving the rule, the validation logic is :

1. Validate on browser site.  The logic is defined in the parameters. If this steps fails,  ***errorMsg*** will show in UI.



1. If step 1 is OK, call the validate () method of each rule in backend (app site). If it fails, the error messages (the return value of validate ()) will display on UI.

## Package Structure

|  |  |  |
| --- | --- | --- |
| **Package** | | **Description** |
| com.ericsson.cms.epgmgmt.custom.rule | | Defines interface for custom rule Entry Point. |
| com.ericsson.cms.epgmgmt.custom.rule.constants | | Defines some rule constants. |
| com.ericsson.cms.epgmgmt.custom.rule.context | | Defines context for custom rule including rule metadata, rule types and context. |
| com.ericsson.cms.epgmgmt.custom.rule.db.executor | | Defines Executor for handling sql. |
| com.ericsson.cms.epgmgmt.custom.rule.exception | Defines some exceptions. | |
| com.ericsson.cms.epgmgmt.custom.rule.implement | Defines a abstract rule implemented class. | |
| com.ericsson.cms.epgmgmt.custom.rule.param | Defines some parameter types. | |
| com.ericsson.cms.epgmgmt.custom.rule.util | Defines some utility class for custom rule. | |
| For more detail, see the file Custom\_Rule\_Sdk\_for\_EPG\_5.1.000\_api.zip. | | |

# FAQ

**Q: When are the custom rules loaded?**

**A:** A custom rule has been built and deploy to application server. And the next time when the application server starts, custom rules are loaded automatically.

**Q: Is it compulsive to implement the validate logic when I create a custom rule?**

**A:** No.You can choose whether to add validation logic code or not.

**Q: Can I implement other parameter types for custom rule meta-data?**

**A:** No. But we will support more parameter types in the future version.

**Q: Can I pack multiple custom rules into one jar file?**

**A:** Yes. Steps are as below:

1. Create sigle or multiple classes that inherit BaseCustomNormalizationRule.
2. Override method “innerExecute(…)” and “validate(…)” according to your custom logic for each custom rule class.
3. Modify the custom rule meta-data in json file as required.
4. Build and deploy your custom rules.

**Q: Can I modify the logic of the custom rule in CMS UI?**

**A:** No. This version only supports to modify some properties value, such as program id, program name, or other properties that you have defined in custom-rule-definations.js. So the logic of a custom rule can not be changed after it has been built.

**Q: Can I use custom rule in other EPG version lower than 5.0?**

**A:** No. Only EPG 5.0+ supports custom rule.

**Q: Can EPG 5.0 support to recognize and load multiple custom rule packages with different names?**

**A:** Yes. Both multiple custom rules in one package and in multiple packages are supported.