

Oracle® Fusion Middleware

Developing Custom Jars and Custom Stages in Oracle Stream Analytics



E93125-03
April 2019



Oracle Fusion Middleware Developing Custom Jars and Custom Stages in Oracle Stream Analytics,

E93125-03

Copyright © 2018, 2019, Oracle and/or its affiliates. All rights reserved.

Primary Author: Oracle Corporation

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Audience	iv
Documentation Accessibility	iv
Conventions	iv
Related Documents	v

1 Developing Custom Stages and Custom Functions

Creating a Custom Jar	1-1
Custom Stage Type	1-2
Adding a Custom Stage	1-2
Implementing a Custom Stage	1-3
Custom Functions	1-3
Implementing Custom Functions	1-3
Limitations	1-3
Mapping of Data Types	1-4

A Samples

Sample Custom Stage Type	A-1
Sample Custom Function	A-2

Preface

Developing Custom Stages and Functions describes the supported data types, limitations, and the procedure for adding custom stages and functions in Oracle Stream Analytics.

Topics:

- [Audience](#)
- [Documentation Accessibility](#)
- [Conventions](#)
- [Related Documents](#)

Audience

This document is intended for developers who are looking to add custom stages and custom functions in Oracle Stream Analytics.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Accessible Access to Oracle Support

Oracle customers who have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Related Documents

Documentation for Oracle Stream Analytics is available on [Oracle Help Center](#).

Also see the following documents for reference:

- *Understanding Oracle Stream Analytics*
- *Quick Installer for Oracle Stream Analytics*
- *Known Issues in Oracle Stream Analytics*
- *Spark Extensibility for CQL in Oracle Stream Analytics*
- *Using Oracle Stream Analytics*

1

Developing Custom Stages and Custom Functions

Custom stage types or functions allow you to develop functionality that is not available in common stages and functions. For example, uncommon calculations, conversions, or algorithms.

As an example, you might want to calculate a message digest using the MD5 algorithm. This algorithm is not part of the in-built function library and it is not practical to implement it as an expression.

Custom stage types and functions are implemented in Java programming language using interfaces, classes, and annotations provided in the `osa.spark-cql.extensibility.api.jar` library. You can download this jar file from the installation folder: `osa-base/extensibility-api/osa.spark-cql.extensibility.api.jar`. For more information, see [Spark Extensibility for CQL in Oracle Stream Analytics](#).

For a custom stage type, you need to implement the `EventProcessor` interface and apply the `@OsaStage` annotation to your class declaration. You must implement the `processEvent()` method that takes an input `Event` and returns an `Output Event`, both of which must be defined using the input and output spec respectively.

Topics:

- [Creating a Custom Jar](#)
- [Custom Stage Type](#)
- [Custom Functions](#)
- [Limitations](#)
- [Mapping of Data Types](#)

Creating a Custom Jar

A custom jar is a user-supplied Jar archive containing Java classes for custom stage types or custom functions that will be used within a pipeline.

To create a custom jar:

1. In the **Create New Item** menu, select **Custom Jar**.
The Import a jar for custom stages and functions wizard appears.
2. On the **Type Properties** page, enter/select suitable values and click Next:
 - a. In the **Name** field, enter a meaningful name for the custom jar you are trying to import into the application.
 - b. In the **Description** field, provide a suitable description.
 - c. In the **Tags** field, select one or more of existing tags, or enter your own tags.

- d. In the **Custom Jar Type** drop-down list, select **Custom Jar**.

3. On the **Custom Jar Details** page, click **Upload file**, select the jar file that you want to import into the application, and then click **Save**.

Make sure that the jar file you select for uploading is a valid jar file and includes all the required dependencies.

Custom Stage Type

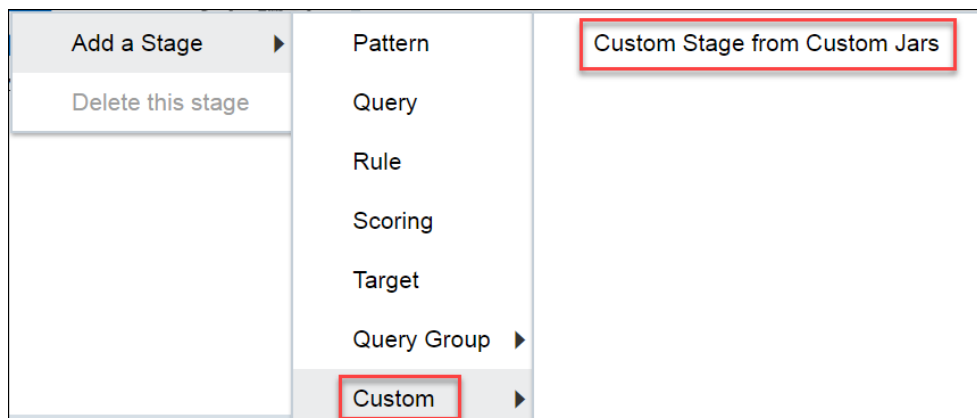
Custom Stage is a type of stage where you can apply your custom stage type to your streaming data in your pipeline. It behaves like any other type of stage with data flowing into and out of it. It is close to a pattern stage in the way that you are asked to configure a few parameters before its logic applies to the stream.

Adding a Custom Stage

You can add filters in a pipeline to obtain more accurate streaming data.

To add a custom stage:

1. Open the required pipeline in Pipeline Editor.
2. Right-click the stage after which you want to add a custom stage. Click **Add a Stage**, and **Custom** and then select **Custom Stage from Custom Jars**.



3. Enter a meaningful name and suitable description for the scoring stage and click **Save**.

4. In the stage editor, select appropriate values for the following:
 - a. **Custom Stage Type** — the custom stage that was previously installed through a custom jar
 - b. **Input Mapping** — the corresponding column from the previous stage for every input parameter

The screenshot shows a web interface for configuring a stage. It has two tabs: 'Parameters' (selected) and 'Visualizations'. Under 'Parameters', there are two sections:

- 'Custom Stage Type': A dropdown menu with the text 'Set Custom Stage Type' and a downward arrow.
- 'Input Mapping': A table with two columns. The first column is labeled 'Input' and the second is labeled 'Previous Stage'.

You can add multiple custom stages based on your use case.

Implementing a Custom Stage

For a custom stage type, you need to implement the `EventProcessor` interface and apply the `@OsaStage` annotation to your class declaration. You must implement the `processEvent()` method that takes an input `Event` and returns an `Output Event`, both of which must be defined using the input and output spec respectively.

Custom Functions

The functions that get installed when you add a custom jar are known as custom functions.

The custom functions will be available in the Expression Builder after they get installed. The custom functions will be listed under the *Custom* category. These functions are accessible like any other out of the box function within Oracle Stream Analytics.

Implementing Custom Functions

For a custom function, apply the `@OsaFunction` annotation to a method in any class, including a class implementing a custom stage type. For more information, see the [Javadoc](#) and the [Samples](#).

Note:

Functions with same name within same package/class/method in same/different jar are not supported.

Limitations

The limitations and restrictions of the custom stages and custom functions are listed in this section.

Custom stage type and custom functions must:

- only be used for stateless transformations. Access to state from previous calls to stage type or function methods cannot be guaranteed and might change based on optimizations.
- not use any blocking invocations.
- not start a new thread.
- not use any thread synchronization primitives, including the wait() method, which could potentially introduce deadlocks.
- have/be in a fully-qualified class name.

When you use the custom stages or custom functions, be careful about the heap space usage.

**Note:**

The resulting jar must include all the required dependencies and third-party classes and the size of the jar file must be less than 160 MB.

Mapping of Data Types

The following table lists the data types that can be used by custom stage types and custom functions.

Oracle Stream Analytics Data Type	Java Data Type	Comment
BOOLEAN	boolean	
INT	int	
BIGINT	long	
FLOAT	float	
DOUBLE	double	
STRING	String	
BIGDECIMAL	BigDecimal	
TIMESTAMP	long (in nanoseconds)	Can only be used in Custom Stage Types
INTERVAL	long	Can only be used in Custom Stage Types

A

Samples

A sample custom stage type and sample function that you can use are provided.

Topics:

- [Sample Custom Stage Type](#)
- [Sample Custom Function](#)

Sample Custom Stage Type

This sample class defines a custom stage that takes one textual field and produces an MD5 hash for it.

```
package com.oracle.osacs;

import com.oracle.cep.api.event.*;
import com.oracle.cep.api.annotations.OsaStage;
import com.oracle.cep.api.stage.EventProcessor;
import com.oracle.cep.api.stage.ProcessorContext;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.HashMap;
import java.util.Map;

@SuppressWarnings("serial")
@OsaStage(name = "md5", description = "Create an md5 hex from a string",
inputSpec = "input, message:string", outputSpec = "output, message:string,
md5:string")
public class CustomMD5Stage implements EventProcessor {

    EventFactory eventFactory;
    EventSpec outputSpec;

    @Override
    public void init(ProcessorContext ctx, Map<String, String> config) {
        eventFactory = ctx.getEventFactory();
        OsaStage meta = CustomMD5Stage.class.getAnnotation(OsaStage.class);
        String spec = meta.outputSpec();
        outputSpec = TupleEventSpec.fromAnnotation(spec);
    }

    @Override
    public void close() {
    }

    @Override
```

```
public Event processEvent(Event event) {
    Attr attr = event.getAttr("message");
    Map<String, Object> values = new HashMap<String, Object>();
    if (!attr.isNull()) {
        String val = (String) attr.getObjectValue();
        String md5 = null;
        try {
            MessageDigest md = MessageDigest.getInstance("MD5");
            md.update(val.getBytes());
            byte[] digest = md.digest();
            md5 =
javax.xml.bind.DatatypeConverter.printHexBinary(digest);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        values.put("message", val);
        values.put("md5", md5);
    } else {
        values.put("message", "empty");
        values.put("md5", "empty");
    }
    Event outputEvent = eventFactory.createEvent(outputSpec, values,
event.getTime());
    return outputEvent;
}
}
```

Sample Custom Function

This sample class defines a custom function that takes one textual field and produces an MD5 hash for it.

```
package com.oracle.osacs;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import com.oracle.cep.api.annotations.OsaFunction;

public class CustomMD5Function {

    @OsaFunction(name = "md5", description = "Create an md5 hex from a
string")
    public static String md5(String message) {
        String result = null;

        try {
            MessageDigest md = MessageDigest.getInstance("MD5");
            md.update(message.getBytes());
            byte[] digest = md.digest();
            result =
javax.xml.bind.DatatypeConverter.printHexBinary(digest);
        } catch (NoSuchAlgorithmException e) {
```

```
        e.printStackTrace();  
    }  
    return result;  
}  
}
```