*FOR:*

**Headstrong**

*CONTENT:*

Teevra – Server-side Component Developer Guide

*April 18, 2013*

L



**DISCLAIMER**

**This document contains confidential and proprietary information belonging to Headstrong Corporation and the same is distributed only in pursuance of and subject to the terms of the license agreement entered into with Headstrong Corporation. No part of this document may be copied, reproduced, distributed or disseminated in any form (electronic or otherwise) without the prior written permission of Headstrong Corporation. No intellectual property rights, copyright on the provided documents are conferred to the recipient. The disclosure of the information contained herein does not lead to the legal conferring or granting of rights, unless provision is expressly made for this.**

**Headstrong reserves all rights to make changes to the information described in this document at any time without notice and obligation to notify any person of such changes.**

**© 2009 Headstrong Corporation. All rights reserved.**

**Document History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Release Date | Author | Description of Change |
| 0.1 | 11-Nov-2009 | Som Krishan Soni | Initial Draft |
|  |  |  |  |

**Reference Documents**

|  |  |
| --- | --- |
| **Document Name** | **Document Version** |
| Teevra\_Specification\_ProcessConfigurationModel.docx | 2.0 |
|  |  |
|  |  |
|  |  |

**TABLE OF CONTENTS**

[1 Introduction 5](#_Toc246852251)

[2 Scope of the document 5](#_Toc246852252)

[3 Assumptions 5](#_Toc246852253)

[4 Abbreviations 5](#_Toc246852254)

[5 Dependencies 5](#_Toc246852255)

[5.1 Inheriting POM from Teevra’s Parent POM 5](#_Toc246852256)

[5.1.1 Artifact 6](#_Toc246852257)

[5.2 Accessing required interface for a component 6](#_Toc246852258)

[5.2.1 List of interfaces 6](#_Toc246852259)

[5.2.2 Artifact 7](#_Toc246852260)

[6 OSGi-Fying 8](#_Toc246852261)

[7 Writing Mediation Service 8](#_Toc246852262)

[7.1 Registering Service 8](#_Toc246852263)

[7.2 Complex Configuration 9](#_Toc246852264)

[8 Writing Connector 9](#_Toc246852265)

[8.1 Writing a FusionComponent 9](#_Toc246852266)

[8.2 Writing a FusionEndpoint 9](#_Toc246852267)

[8.3 Writing a FusionProducer 10](#_Toc246852268)

[8.4 Writing a FusionConsumer 10](#_Toc246852269)

[9 Error Handling 10](#_Toc246852270)

[10 Examples 11](#_Toc246852271)

[10.1 Mediation Service 11](#_Toc246852272)

[10.2 Binding Component 11](#_Toc246852273)

[11 JavaDoc for dependent classes 11](#_Toc246852274)

# Introduction

Teevra is a message oriented framework built on top of OSGi (Open Source Gateway Initiative) based runtime. Teevra comes with multiple pre-built components which can directly be used in the process orchestration. It also provides API for creation of new component which can be deployed to extend the functionality of Teevra server.

# Scope of the document

This document is a developer guide to create new component which can be plugged in into Teevra container.

# Assumptions

* Developer has access to Teevra project repository (Maven Repository).
* Apache Maven (2.0.9) is being used as the build tool.

# Abbreviations

|  |  |
| --- | --- |
| PCM | Process Configuration Model |
| POM | (Maven’s) Project Object Model |

# Dependencies

Mediation service and components should conform to the interfaces defined in Teevra . Please follow steps below to include the Maven dependencies in the component’s POM file.

## Inheriting POM from Teevra’s Parent POM

Though it’s not mandatory, it is recommended to extend the new service/component from the *Teevra platform parent* module, which would give the new service/component access to common properties and dependencies.

* Properties
  + camel-version
  + spring-version
  + platform.version
* Dependencies
  + Junit:junit:4.3
  + org.slf4j:slf4j-api:1.5.2
  + org.apache.felix: org.osgi.core:1.0.1
  + org.easymock: easymock:2.0

### Artifact

* **Group Id**: com.hs.fusion
* **Artifact Id**: teevra.platform.parent
* **Version**: teevra version

## Accessing required interface for a component

Commons module contains all the interfaces required for development of new service/component. It also contains some common utility required by all other modules.

### List of interfaces

* MediationService
  + Mediation service will extend from this service. Each mediation service will have to register itself as an OSGi service. The key (fully qualified class/interface name) used to register the service will be used in the PCM to refer to this service.
  + To avoid the usage of fully qualified in the PCM service alias can also be registered with *ServiceAliasManager.*
  + There will always be a single instance of service registered with the container for a key.
  + On receipt of the message process method is invoked with MessageExchange and ServiceConfig as parameter.
* MessageExchange
  + MessageExchange contains detail of incoming message ( #getIn() ) , header information.
  + Mediation service can modify the MessageExchange, which will be passed to next service/component in the PCM.
* ServiceConfig
  + Contains service configuration required for processing the message.
* Message
  + Represent a message payload with two important parts.
    - Message header
    - Message Body.
* ErrorHandler
  + If Error handler is configured for the process/service same can be accessed from the ServiceConfig ( #getErrorHandler )
  + Service/Component level configuration overrides the configuration defined at process level.
* FusionComponentFactory
  + Each component has to be registered with FusionComponentFactory before it can be used in the PCM.
  + Key used to register the component will be used in PCM to refer the component.
* FusionComponent
  + Represent a binding (Connector) component in Teevra.
  + It’s a factory of FusionEndpoint.
* FusionEndpoint
  + FusionEndpoint has two important methods.
  + createProducer will create a FusionProducer to sending message exchanges to the endpoint.
  + createConsumer will create consumer for consuming message exchanges from endpoint via a Fusionprocessor.
* FusionProducer
  + Provides a channel on which clients can create and invoke message exchanges on a FusionEndpoint.
* FusionConsumer
  + FusionConsumer consumes messages from the endpoint using the given FusionProcessor
* FusionProcessor
  + Processor will process the message as per the PCM.

### Artifact

* **Group Id**: com.hs.fusion
* **Artifact Id**: fusion-commons
* **Version**: ${platform.version}

***If service/component extends from Teevra platform parent module the variable platform.version can be used to specify the version number otherwise the version number has to be specified manually.***

# OSGi-Fying

Each service module should be converted into an [OSGi](http://www.osgi.org/Main/HomePage) Bundle to be deployed in the Teevra container. Teevra is using [Apache Felix](http://felix.apache.org/site/index.html) as the OSGi container. Teevra uses BND maven plug-in to create OSGi bundles, but any plug-in/process can be used to create the bundle. Teevra just requires the modules to be converted in the OSGi bundle before deployment. Please follow the instructions mentioned in the below mentioned link to create an OSGi bundle using maven bundle plugin.

<http://felix.apache.org/site/apache-felix-maven-bundle-plugin-bnd.html>

# Writing Mediation Service

Mediation Service is a component that does any kind of processing with the message in the exchange. MediationService interface is used to create a new Mediation Service. Alternatively service can extend GenericMediationService which in turn implements MediationService. Each mediation service defines a process method which is invoked whenever a new message is received and required to be processed by the service.

## Registering Service

Each OSGi bundle may also define an Activator which would receive bundle start/stop events. On bundle startup the mediation service has be registered with OSGI Bundle Context.

Below is a sample code snippet to do the same.

*MediationService service = new SampleService(); // Declare service.*

*Dictionary props = new HashTable();*

*// additional properties.*

*bundleContext.registerService(SampleService.class.getName(), service, props );*

Above snippet would register a service with **BundleContext** against the fully qualified SampleService class name.

Same key has to be used in the PCM to refer to the service. Alternatively an alias can also be provided after the registration. Below is a snippet to register a service alias for this same service.

*ServiceAliasManager.getInstance().registerServiceAlias(SampleService.class*

*.getSimpleName(), SampleService.class.getName());*

After alias registration simple name **SampleService** can also be used to refer to the service in PCM.

## Complex Configuration

TODO :

# Writing Connector

Connector is a component through which Teevra can interact with the external system handling its protocol and session. Connector can be used either as the source or destination of Message Exchange. Each new Connectorhas to implement following interface in order to support both sending and receiving the message from the endpoint.

## Writing a FusionComponent

**Component** implements the FusionComponent interface, is a factory of FusionEndpoint with only one method createEndpoint. This class will create the Endpoint using the Configuration (BindingConfig) provided. Same component instance will be used in all the processes to create the FusionEndpoint. Below is a code snippet for a sample Component class.

public class SampleComponent implements *FusionComponent* {

public FusionEndpoint createEndpoint(

BindingConfig bindingConfig) throws Exception {

// create endpoint and return.

}

}

## Writing a FusionEndpoint

**Endpoint** implements FusionEndpoint interface, It creates producer and consumer based on their usage in the PCM, if the Endpoint is being used as the source *createConsumer* will be called to create a new Consumer, which will be used to receive message from the endpoint. And in case the Endpoint is being used as destination createProducer will be called to create the Producer, which will be used to send the messages to the endpoint.

public class SampleEndPoint implements FusionEndpoint {

public SampleEndPoint (BindingConfig bindingConfig) {

// constructor.

}

public FusionConsumer createConsumer(

FusionProcessor fusionProcessor) throws Exception {

// Create consumer and return.

}

public FusionProducer createProducer() throws Exception {

// Create consumer and return.

}

}

## Writing a FusionProducer

**Producer** implements FusionProducer interface, It will send message to the Endpoint. It’ll receive MessageExchange it’ll use the Binding Configuraiton ( This is different from Mediation service where with each method invocation the configuration is also being passed, as there is only one instance of service registered, which is not the case with Endpoint), already provided to the Endpoint at the time of creation, to format and send the message to the Endpoint.

## Writing a FusionConsumer

**Consumer** implements FusionConsumer interface, It receives the message from the Endpoint ( polling/event driven ) and creates a Message Exchange ( Default implementation FusionMessageExchange) and invokes process method on FusionProcessor.

# Error Handling

ErrorHandling API allows service/component to handle/report error as per the configuration provided in the PCM. Error handling if configured can be accessed from the ServiceConfig/BindingConfig. To report error create one ErrorContext object and invoke reportError method.

# Examples

Sample Service and Component are maven projects. With eclipse if maven plug-in is installed these can directly be imported into eclipse.

## Mediation Service



## Binding Component



# JavaDoc for dependent classes

