|  |  |
| --- | --- |
| www.jBilling.com | newlogo  The Open Source Enterprise Billing System |
| Telecom Guide |
| Mediation 3.0 |

**Copyright**

This document is Copyright © 2004-2013 Enterprise jBilling Software Ltd. All Rights Reserved. No part of this document may be reproduced, transmitted in any form or by any means -electronic, mechanical, photocopying, printing or otherwise- without the prior written permission of Enterprise jBilling Software Ltd.

**jBilling** is a registered trademark of Enterprise jBilling Software Ltd. All other brands and product names are trademarks of their respective owners.

**Author**Vikas Bodani

**Revision number and software version**Document Revision **1.0**, based on **jBilling** **3.3 Telco Release**

***Mediation 3.0***

jBilling 3.3 Telco version release packs within itself a new version of Mediation, formally referred by the name **Mediation 3.0**. *The previous mediation process has been rechristened as Mediation 2.0.*

To begin with, **Mediation 3.0** had two important goals:

1. Provide unlimited scalability to jBilling Mediation
2. Higher throughput for the jBilling Mediation System (Benchmarks of 1000 records per second)

To this end, **Mediation 3.0** uses Apache Hadoop as the engine for distributed processing of Call Data Records (CDRs).

Other important goals of the newer version of Mediation were:

1. Simple orchestration of Mediation batch jobs/steps
2. Backwards compatibility
3. Consistent reporting

In the subsequent sections, we will examine architectural decisions and choices made for **Mediation 3.0** to achieve the important goals.

**Distributed Processing**

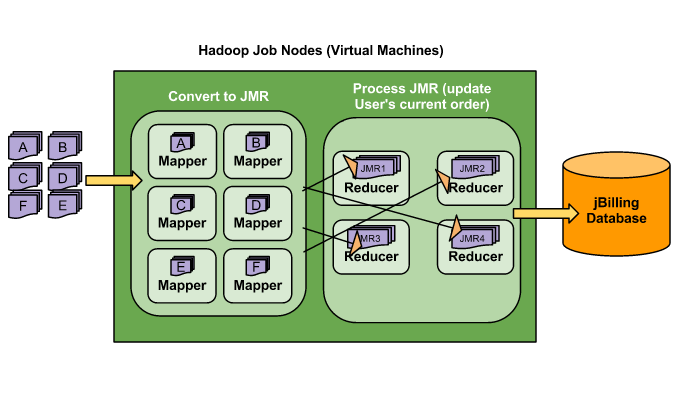
Distributed processing may also be termed as Parallel processing. The obvious goal being higher through put per unit of time. However, distributed computing set-up and architecture cannot be trivial. Secondly, a system that can grow and shrink based on need is a tough goal in itself.

Today, all that has been mitigated with the open source Apache Hadoop framework. Apache Hadoop is a framework that allows distributed processing of large data sets across a cluster of commodity computer hardware. It is built for fault-tolerance and scalability.

**Distributed Processing - Architecture**

JBilling leverages this feature to run its Mediation module as a Hadoop **Map-Reduce** jobs. Each Hadoop Mapper and Reducer is a component of jBilling Mediation and the way Hadoop is designed, each Mapper or a Reducer component runs on a separate node, known as a Task Node. This task node can be a physical computer node in a network or a Virtual node that exists anywhere on the network.

In other words, by leveraging Hadoop Map Reduce as jBilling Mediation components, jBilling Mediation achieves un-limited scalability that can grow and shrink based on the requirement.

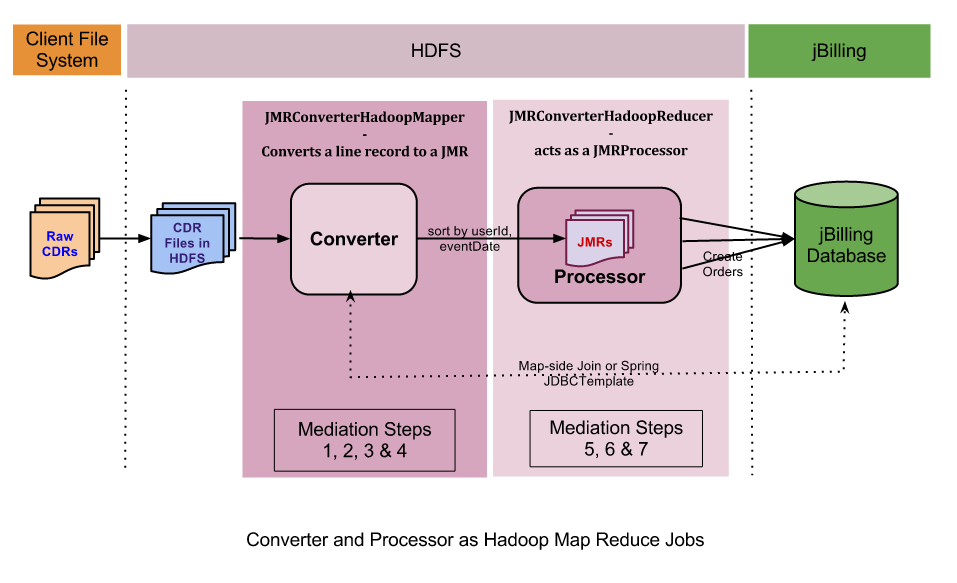


Distributed processing of events/CDRs in jBilling **Mediation 3.0** is divided into two steps as depicted in the diagram above. These two steps are:

1. **Converter** (convert proprietary event/cdr data to **JMR** or **JbillingMediationRecord**)
2. **Processor** (consume JMRs to update user's orders)

|  |
| --- |
| **JMR or JbillingMediationRecord**  JbillingMediationRecord or JMR in short, is a new entity/class in jBilling which represents the bare minimum data that jBilling needs to mediate or charge an event. A valid JMR must have following fields resolved:   1. jBilling User ID 2. jBilling User's Currency ID 3. Event Date or Mediation Event Date 4. jBilling Item or Product ID (Required to create the Order) 5. Quantity to be charged 6. Event Description 7. Pricing Fields Data (for Pricing Rules) 8. Record Key   Pricing Field Data could be the destination country, purchase date, time etc..  More about JMR in this document later. |

Let us review these Distributed processing of CDRs in more detail.



**JMR Converter - *Mapper task***

**Convert Proprietary CDR/Events into jBilling's JMR Format**

JMR Converter is the first step of Mediation Event processing in jBilling **Mediation 3.0.** The JMR Converter step is a Hadoop Mapper phase of a Hadoop Map-Reduce job.

This is a *translation* phase in which a raw CDR data in any format is parsed, the fields split and are passed through the configured Mediation Steps to resolve it into JMRs.

In simple terms, in this phase a raw CDR or Event is broken down into two parts: pricing fields & JMR record. The details of how this is done is wrapped in the default implementation of this phase in the Hadoop Mapper class **JMRConverterHadoopMapper.** The process involves going through the sub-set of 7 Mediation Steps from the traditional jBilling Mediation, now called as Mediation 2.0. Let us review the sample steps in the example below:

**JMRConverterHadoopMapper**

The way JMRConverterHadoopMapper is designed, it receives each line from the Input record file as one record. The Mapper joins this record with jBilling User/Customer database and resolves the fields required for JMR Creation. Each good record from the input file ends up as a good JMR object and is transferred to the JMR Processor.

The converter phase involves following separate activities:

1. **Convert Line to Mediation Record: MediationRecordLineConverter**

The MediationRecordLineConverter populates a Record object for use by the MediationCDRResolver to resolve the Record into a JMR as a first step of the Converter phase. It does this by splitting the line using format tokenizer (default separator being ',' or comma) and populating each field as a Pricing Field into the Record 'fields' list.

The MediationRecordLineConverter interface defines a single method convertLineToRecord(line, format). The default implementation, **BasicMediationRecordLineConverter** is all that is required to split an input line into a **Record**.

1. **Resolve CDR: IMediationCDRResolver**

The IMediationCDRResolver implements a single method called resolveCdr(entityId, result, record).

Its basic job is to invoke every step configured in the *steps* list field property. Each steps in the configuration is an instance of AbstractMediationStep class. Prior to steps execution, it adds a few validation steps to validate the input record.

Each validation step is an instance of IMediationStepValidation and can be customized by overriding the default implementation.

Both MediationSteps as well as IMediationStepValidation may be customized by providing a new implementation and swapping them from the default ones in the configuration. Therefore, making Mediation Steps more like plug-in, with add/remove capability through configuration.

Both, Mediation Steps for resolving data/fields as well as Validation steps may be added via Spring configuration as well.

e.g. Configuring MediationCDRResolver

|  |
| --- |
| <bean id=*"mediationCdrResolver"* class=*"com.sapienter.jbilling.server.mediation.step.JMRMediationCdrResolver"*>  <property name=*"steps"* ref=*"mediationCdrResolverSteps"* />  </bean> |

1. **Sort by User ID and Event Date**

JMRConverterHadoopMapper uses ***secondary-sort*** of Map-Reduce framework to sort the output JMR records by the resolved jBilling User ID and Event Date. Without secondary-sort, which is the default behaviour, the records will only be sorted by User ID, which acts as the Key for the record.

**JMR Processor - Process JMRs into Orders**

JMR Processor is equivalent to the Hadoop Reducer in the Hadoop Map-Reduce framework.

Each JMRProcessor receives JMRs or translated events as input for a given jBilling user. These events are ordered by the Event Date. This allows for sequential processing of events within jBilling with a default step of Pricing included in between.

**JMRProcessor**

JMRProcessor is an implementation of IJMRProcessor. JMRProcessor is responsible for

validating a JMR (JbillingMediationRecord) record and if good, processing that record into an order for that customer or updating an existing order.

The JMRProcessor implementation can be Spring configured and a right implementation can be swapped via configuration.

1. **Validate JMR Record**

Any errors discovered at this stage will close the record for further processing. The errors are saved with the JMR itself for further action. The record is then closed and no change is affected on jBilling customer or orders.

1. **Get/Create Current Order**

This is the standard mediation step and one of the final actions when an order must either be updated or created for the customer. It is important to resolve if a current order exists that can be updated.

1. **Update current order**

Add the item to the current order or the new order. This may internally trigger Pricing Tasks and Item Management tasks as well. This is different from rules based processing as the rating/pricing step is now combined or almost the final step of Mediation step Process JMR.

***Mediation Steps***

Mediation 3.0 carries the same simple architecture presented by the jBilling Mediation 2.0 i.e. CDR processing in Steps. While the steps for jBilling Mediation still remain the same for generic, simple and efficient processing of user events (CDRs), the architecture and implementation has undergone some significant towards something more modular and easier to manage the new components.

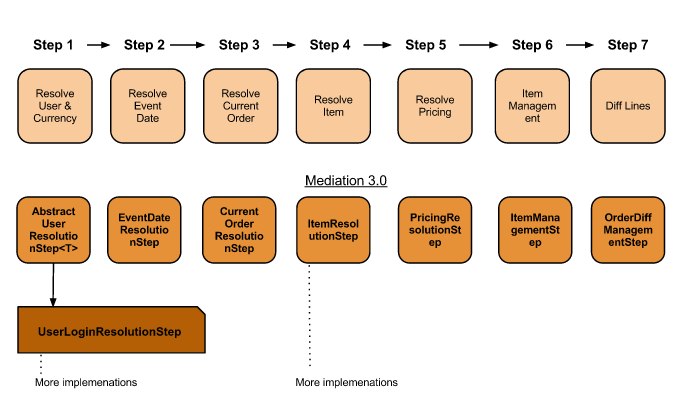
**Mediation Steps**

1. Resolve user and currency
2. Resolve event date
3. Resolve Current Order (Event Date required for this)
4. Resolve Item or Product
5. Resolve Price (Rating or Pricing)
6. Line Creation
7. Item Management

With Mediation 3.0, each step is a compilable unit with its own custom logic something similar to a 'Single responsibility principal'. For example, A jBilling user may be resolved in many ways like: User ID based resolution, User Name based resolution, Email ID of the user etc. One such example shown below is **UserLoginResolutionStep.java** component class, which tries to resolve the user based on the user's login name.

As a result, each erstwhile Mediation 2.0 Step has been transformed into a component of itself. And that is why each such step component is represented with its own class or Abstract class.

It will easier to visualize this with the component class for each step.



Each of the Mediation Step is now a Spring Bean and the sequence of steps are required to be added as an ordered collection to the steps to the configuration of CDR Resolver (IMediationCDRResolver)

**Why Steps**

Mediation was and is never a one-size-fits-all component of a Telco system. It is indeed customized for each event format.

In earlier versions of jBilling, the entire mediation was either a big rules file or a big java file with all the business logic encapsulated in methods with headers for appropriate steps. Each method implementation was customized depending on what rule to use and the format of the incoming record. Thus a fine-grained control was achieved over the processing.

In Mediation 3.0, the biggest separation is the separation of framework of Mediation or the Mediation component itself from the customization. Therefore, the only thing that needs to be customized is a relevant step that does not match the standard implementations vis-a-vis the input format. No other part of Mediation 3.0 is required to change including the Mapper, Reducers and the Job Runner.

Example Standard implementation for some steps.

**UserLoginResolutionStep**

This class implements the function to resolve a User based on the Username. The userName is made available to the class as an input by the User Name Field name via spring configuration. See below:

|  |
| --- |
| <bean id=*"userLoginResolverStep"* class= *"com.sapienter.jbilling.server.mediation.step.user.UserLoginResolutionStep"*>  <property name=*"usernameField"* value=*"userfield"*/>  </bean> |

**How to create Steps**

Mediation 3.0 steps are simple Java Classes which follow a Single-Responsibility principle. They focus on just enough logic to resolve JMR fields specific to that step.

For example, let us create a custom step class to resolve jBilling user based on the email field. This is under the assumption that each user can be uniquely identified by an email.

1. **Parent Class**

The root class of all steps is AbstractMediationStep class.

1. **Step class extends from AbstractMediationStep**

A Step class has to implement or provide behaviour to the method executeStep. In the example below, the Parent class for UserResolutionByEmailStep is AbstractUserResolutionStep which in turn extends from AbstractMediationStep.

A simple Step example:

|  |
| --- |
| **public** **class** UserResolutionByEmailStep **extends** AbstractUserResolutionStep {  @Override  **public** **boolean** executeStep(Integer entityId, MediationResult result, List fields) {  //implement logic to find user by email id  }  } |

1. **Configure it as a Spring Bean**

A Step can be injected/configured as Steps-list by defining it as a Spring bean and later using it as a reference. This helps in providing other fields/values as configuration itself.

Step as a bean:

|  |
| --- |
| <bean id=*"emailUserResolverStep"* class= *"com.sapienter.jbilling.server.mediation.step.user.UserResolutionByEmailStep"*>  <property name=*"emailField"* value=*"email"*/>  </bean> |

This Step can be used in multiple configurations.

1. **Add the step to list of steps configuration**

A configuration has many steps. Steps are added as required into the Steps configuration.

|  |
| --- |
| <util:map id=*"mediationCdrResolverSteps"*  map-class=*"java.util.LinkedHashMap"*>  <entry key=*"USER\_CURRENCY"* value-ref=*"emailUserResolverStep"* />  <entry key=*"EVENT\_DATE"* value-ref=*"JMREventDateResolverStep"*/>  ....  </util:map> |

***jBilling Mediation Record - JMR Format***

JbillingMediationRecord or JMR in short, is a new entity/class in jBilling which represents the bare minimum data that jBilling needs to mediate or charge an event. A valid JMR must hold the following data:

1. User ID
2. User's currency
3. Event Date
4. Item or Product ID resolved, to create the Order
5. Event Description
6. Quantity to be charged
7. Pricing - Any specific description to be added to the mediated event

Besides the above event related data, the each JbillingMediationRecord is also given a unique key called as the Record Key. This key, if not available in the input, is automatically provided by the jBilling Mediation Process.

**JMR Fields in detail**

**User ID**

This must be the jBilling User id for the respective customer. The user id is a mandatory field for a JMR and must be available for the mediation process to succeed

**Currency**

The Currency is generally resolved automatically by jBilling however, when the Mediation process is processing a JMR, this field must be resolved to the jBilling Currency ID that is set in the User's jBilling profile

**Event Date**

The date/time of the event for Mediation. This is an important value that is not just mandatory, but it may also have an impact on the pricing of the event in case the pricing is time sensitive.

*For e.g.* A GSM subscription plan with night-calling feature that comes with a discounted per minute usage charge etc.

**Product ID**

The jBilling Product/Item ID that is used to record the purchase/event. The one-time Order is created using this Product ID

**Description**

The event description that shows up on the Invoice. This is actually saved as the Order description.

**Quantity**

The amount/magnitude of purchase. This value can be a decimal value. It may represents number of items of the Product ID purchased or the number of minutes etc. purchased/consumed of a plan

**Pricing**

These are comma separated strings of PricingField objects encoded as a string using the jBilling method PricingField.encode(PricingField). Pricing field is not a mandatory field for mediation. Some Pricing Rules or Pricing Plug-ins may use pricing fields to resolve a more accurate price.

In order to resolve the prices of an event correctly, Pricing Rules or Models may be used. These Pricing rules may use/require additional information related to an event/record in order to determine the price correctly.

*For e.g.* Foran international call, an important pricing data is the *destination* country because the call is most likely priced differently for different country. Similarly, there could be various information that comes with the Event/CDR that is required for price calculation for that unique event.

It is for this reason JbillingMediationRecord also stores pricing fields as an encoded string called *pricing*.

|  |
| --- |
| **Sample JMR 'pricing' field value**  :src::string:310-1010,:dst::string:1-800-123-4567,:userid:integer:1234 |

Besides the above expected records, **JbillingMediationRecord** also happens to store any errors that may exist with a particular event/CDR data. These are not required as an input but instead are populated automatically by jBilling during JMR validation step.

The objective behind the **JbillingMediationRecord** is to have a proprietary data representation to allow mediation of records from multiple sources and represent them as the simplest format required for a successful processing.

Another intuitive objective to have/define JbillingMediationRecord is to allow clients to send JMR formatted records to jBilling which may be processed directly without any conversion/translation/normalization steps. This allows clients to optimize data transfer for mediation and filter out what is proprietary event data and will be discarded by jBilling.

***Mediation 3.0 - Configuration***

Mediation 3.0 requires the events input to the JMR Converter batch job as a file data. Each file consists of Pricing Fields or fields in general. In order to correctly parse each line record and convert it into a JMR, jBilling needs some information related to the file. These are:

1. Fields data type
2. Key value amongst the fields if any
3. Field separator if any

With **Mediation 3.0**, the first one is the most important. **Mediation 3.0** uses the same Format class that exists since Mediation 1.0 to define the file format. **Mediation 3.0** only extends it to make it configurable as a Spring Bean definition.

**Format Configuration - *Field definition***

|  |
| --- |
| <util:list id=*"mediationFields"* list-class=*"java.util.LinkedList"*>  <bean class=*"com.sapienter.jbilling.server.mediation.FormatField"*>  <constructor-arg value=*"username"* />  <constructor-arg value=*"string"*/>  <constructor-arg value=*"true"*/>  </bean>  <bean class=*"com.sapienter.jbilling.server.mediation.FormatField"*>  <constructor-arg value=*"source"* />  <constructor-arg value=*"string"*/>  </bean>  ...  <bean class=*"com.sapienter.jbilling.server.mediation.FormatField"*>  <constructor-arg value=*"duration"* />  <constructor-arg value=*"string"*/>  </bean>  </util:list> |

**Format Configuration - *Mediation Format***

A format requires format fields, data-types, and separator. Format used to be an internal jBilling class in Mediation 2.0, but it is now exposed as a Spring Bean. Therefore, its properties can now be configured conveniently. This also provides flexibility to achieve multiple combinations.

|  |
| --- |
| <bean id=*"mediationFormat"*  class=*"com.sapienter.jbilling.server.mediation.Format"*>  <property name=*"tokenizer"* ref=*"separatorMediationTokenizer"* />  <property name=*"fields"* ref=*"mediationFields"* />  </bean> |

**Configuration Mediation Steps via Spring Configuration**

|  |
| --- |
| <util:map id=*"exampleMediationStepConfig"*  map-class=*"java.util.LinkedHashMap"*>  <entry key=*"USER\_CURRENCY"* value-ref=*"userLoginResolverStep"* />  <entry key=*"EVENT\_DATE"* value-ref=*"eventDateResolverStep"* />  <entry key=*"CURRENT\_ORDER"* value-ref=*"currentOrderResolverStep"* />  <entry key=*"ORDER\_LINE\_ITEM"* value-ref=*"itemResolverStep"* />  <entry key=*"PRICING"* value-ref=*"pricingResolverStep"* />  <entry key=*"ITEM\_MANAGEMENT"* value-ref=*"itemManagementStep"* />  <entry key=*"DIFF\_MANAGEMENT"* value-ref=*"oderDiffManagementStep"* />  </util:map> |

**Configuration of Hadoop**

**Mediation Job**

A Mediation Job is defined as a Spring Batch job with all the orchestration steps configured as Spring Batch steps.

|  |
| --- |
| <job id=*"mediationJob"* restartable=*"true"* xmlns=*"http://www.springframework.org/schema/batch"*>  <step id=*"cleanHDFS"* next=*"copyToHDFS"*>  <tasklet ref=*"cleanHDFSTasklet"* >  <transaction-attributes isolation=*"DEFAULT"* propagation=*"REQUIRED"* />  </tasklet>  </step>  <!-- replace this step for other mediation configurations -->  <step id=*"copyToHDFS"*>  <tasklet ref=*"copyFileToHDFSTasklet"* >  <transaction-attributes isolation=*"DEFAULT"* propagation=*"REQUIRED"* />  </tasklet>  <end on=*"STOPPED"*/>  <next on=*"\*"* to=*"convertToJMR"*/>  </step>  <step id=*"convertToJMR"*>  <tasklet ref=*"converterTasklet"*>  <transaction-attributes isolation=*"DEFAULT"* propagation=*"REQUIRED"* />  </tasklet>  </step>  </job> |

**Configuration of Batch Jobs**

**Mediation Job Launcher**

The jBilling 3.0 Mediation process is now a Spring Batch job and is always launched using a Spring Job Launcher i.e. an instance of class SimpleJobLauncher.

A Spring Batch job requires a **jobLauncher** and a **job** definition.

|  |
| --- |
| <bean id="*jobLauncher*" class="*org.springframework.batch.core.launch.support.SimpleJobLauncher*">  <property name="*jobRepository*" ref="*jobRepository*"></property>  </bean>  <bean id=*"mediationJobLauncher"* class=*"com.sapienter.jbilling.server.mediation.batch.BatchMediationJobLauncher"*>  <property name=*"jobLauncher"* ref=*"jobLauncher"* />  <property name=*"job"* ref=*"mediationJob"* />  </bean> |

A Spring batch job is a combination of executable steps, which could be any action required for the job to complete successfully, including copying of file at a staging location, any pre-launch setup steps, including the launching of the Batch job. It may also include steps post successfully completion of the batch job. Generally, this step configuration may not change. What is likely to change is the Job Parameters, which are conveniently supplied/changed via a properties file such as hadoop.properties.

jBilling Mediation Batch Job definition:

|  |
| --- |
|  |

***JDBC Mediation***

Besides mediation from CDR files, Mediation 3.0 supports JDBC Mediation as well.

***Mediation 3.0 by Example***

**Scenario**

V Telecom provides is an Information/Communication/Entertainment service provider. It provides following services to its customers: **Internet/Data**, **IP TV**, **Fixed Land line** services. In order to provide its post-paid customers with a single Billing experience, V Telecom uses jBilling to drive Mediation & Billing for all the services. As a result, the Customers receive a single unified Invoice.

V Telecom has different systems which record the consumption of its post-paid customer’s which are consolidated as CDR records in a unified format.

**Input Format**

V Telecom’s CDRs are a comma separated list of records. The format is as described below:

|  |
| --- |
| record\_id,user\_name,start\_time,duration,length,client,destination  1,john\_wakefield,20130103-1:33:222,,10,76-2A-68-DC-E3-24,google.com  2,mark\_twain,20130215-11:45:111,446,,08988288888,919899699434  3,vikas\_sharma,20130312-21:37:333,,100,78-2A-68-DC-E3-24, |

Where:

* **Client**: Specified by a MAC Address, presumably certain MAC address series will always denote an IP TV Box or Set-top box
* **Duration**: Meaningful for a phone call as against an internet browsing usage or an IP TV Usage
* **Destination**: Meaningful for a phone call as well. Different destinations may have different call rates or products to resolve
* **Length:** It can be assumed that the length is in bytes or kilo-bytes to address charging based on plan and allowed usage
* **User Name:** Should resolve to a jBilling user by username in jBilling
* **Start Time:** Time and Date of consumption/call

**Mediation Configuration Steps**

**Format Definition**

**Format Definition is the most basic step that one needs in Mediation** and one of the first step that you need to set up before you can even decide what Steps implementation are required**.**

**The CDR Format Description and definition is slightly different when it comes to Mediation 3.0.**

Looking at the scenario and input sample, the format is expected to be captured/defined as a Spring bean in jBilling’s Spring configuration file *resources.xml* as follows:

<util:list id="vTelecomFormatFields" list-class="java.util.LinkedList">

<bean class="com.sapienter.jbilling.server.mediation.FormatField">

<constructor-arg value="record\_id" />

<constructor-arg value="integer"/>

<constructor-arg value="true"/>

<!-- note: 3rd parameter value true indicates this value acts as a key. Otherwise, jBilling can provide each record a unique key -->

</bean>

<bean class="com.sapienter.jbilling.server.mediation.FormatField">

<constructor-arg value="user\_name" />

<constructor-arg value="string"/>

</bean>

<bean class="com.sapienter.jbilling.server.mediation.FormatField">

<constructor-arg value="start\_time" />

<constructor-arg value="date"/>

<!-- note: date format is a separate configuration, where you can specify, what format this date is -->

</bean>

<bean class="com.sapienter.jbilling.server.mediation.FormatField">

<constructor-arg value="duration" />

<constructor-arg value="decimal"/>

<!-- note: duration is marked ‘decimal’, in anticipation of fractional duration decimal values -->

</bean>

<bean class="com.sapienter.jbilling.server.mediation.FormatField">

<constructor-arg value="length" />

<constructor-arg value="integer"/>

</bean>

<bean class="com.sapienter.jbilling.server.mediation.FormatField">

<constructor-arg value="client" />

<constructor-arg value="string"/>

</bean>

<bean class="com.sapienter.jbilling.server.mediation.FormatField">

<constructor-arg value="destination" />

<constructor-arg value="string"/>

</bean>

</util:list>

Following data types are supported:

1. "string" - any value
2. "integer" - integral values
3. "float", "decimal", "double" - decimal values
4. "date" - Date values
5. "boolean" - True/False values

Once you have the format definition, you would know how jBilling is expected to resolve, say for e.g. the User/Customer who will be charged for the event. For e.g. the ‘user\_name’ field is expected to match the Username in jBilling for that user. Therefore, we need a UserLoginResolutionStep, which says that it resolves the User based on the username value in the user-field. However, if we had just the phone number of the User in the second format field, we would need a custom implementation to resolve user based on phone number in jBilling.

Let us proceed with Steps configuration and determine if we need any custom step implementation.

**Custom Steps Implementation**

Custom implementations are generally NOT required for the following steps:

1. Event Date resolution – a straightforward resolution of data of purchase/consumption based on a format field value and the out-of-box **EventDateResolutionStep** can be used
2. Current Order Resolution – internal to jBilling, format fields input normally do not affect this step and the out-of-box **CurrentOrderResolutionStep** can be used
3. Price Resolution – based on pricing fields available as the input format. For pricing that is dependent or product price in jBilling, such as in the scenario of V Telecom, this step need not be customized. Even for a complex step, where pricing fields are used by the jBilling 3.x Pricing Models, this step remains un-modified and the out-of-box **PricingResolutionStep** can be used/configured
4. Order Diff Management - This step is internal to jBilling and invariably, the out-of-box **OrderDiffManagementStep** component can be configured

Custom implementations are generally required for the following steps:

1. User Resolution – In all cases, CDRs are generated by a third party system, which are not related to jBilling. It is easiest for jBilling to resolve the jBilling Customer based on jBilling’s internal User ID field. Unfortunately, that’ll require more work on the customer’s end to translate/map CDRs to jBilling User ID. Often this mapping is done by a Custom implementation of **AbstractUserResolutionStep**. Thus, there is flexibility to resolve user based on a Meta Field or other static properties such as jBilling User Name, Phone Number of the Primary Contact, unique Email ID etc. In the case of V Telecom, the CDR is expected to contain jBilling User’s username as the second format field, and therefore, we can use the in-built **UserLoginResolutionStep** step component
2. Item Resolution – Quite often, the CDR record/event needs to be mapped to a Product defined in jBilling unless we are talking about only one kind of CDRs such as calls, which need not be mapped to many products in jBilling. But in this scenario, V Telecom is a multi-service provider, and therefore, each CDR should map to a different product and/or a different plan. In case of V Telecom, we need a Custom implementation of **AbstractItemResolutionStep**

**Custom Item Resolution Step**

Let us call it VTelecomItemResolutionStep. The rules may be such as:

1. If the *client* (MAC /Physical address) belongs to the set-top box series, the event should be skipped, because there is no additional charge for this service
2. Else, if the *client* denotes one of customer’s many land-line numbers subscribed, the call should be charged to a specific fixed-line product/item say jBilling Product ID 100
3. Else, if the *client* denotes a Computer MAC Address, it is assumed to be an internet usage and the event charged to a data usage Item, say jBilling Product ID 102
4. Else, log the call as error

**Sample Implementation**

|  |
| --- |
| public class VTelecomItemResolutionStep extends  AbstractItemResolutionStep<MediationResult> {  private Integer itemId;  private ItemDAS itemLoader;  @Override  public boolean executeStep(Integer entityId, MediationResult mediationResult, List<PricingField> fields) {  PricingField client = PricingField.find(fields, "client");  PricingField duration = PricingField.find(fields, "duration");  PricingField length = PricingField.find(fields, "length");  if (client.startsWith(“88-GG”) { //88-GG is the series for set-top boxes  LOG.debug(“IPTV Event, will be skipped.”);  result.setDone(**true**);  return **false**;  } else if (isNumber(client)) {  itemId= 100;  OrderLineDTO line = newLine(itemId, duration.getDecimalValue());  mediationResult.getLines().add(line);  } else {  itemId=102;  OrderLineDTO line = newLine(itemId, lenght.getIntValue());  mediationResult.getLines().add(line);  }  if (itemId != null ) {  mediationResult.setItemId(itemId);  LOG.debug("Number called = " + destination.getStrValue() + ", " + duration.getStrValue() + " minutes");  mediationResult.setDescription("Phone call to " + destination.getStrValue());  return true;  }  } |

**Mediation & Steps Configuration**

Steps Configuration is the last piece in the jigsaw for Mediation 3.0 configuration. At this point, it is critical to wire/configure everything defined above together as a single piece. In other words, the jBilling Mediation 3.0 job and its dependent components (Converters, Processor) should know through the step configuration about the input CDR format, the field separator, Event Date format if any (defaults format to 'yyyyMMdd-HHmmss'), which is used by the Line Converter & last but not the least, Steps configuration.

Through the example of V Telecom, let us see sample configuration for the above scenario:

**Input CDR Format**

Mediation Format requires two things:

1. Format Fields Definition (Task 1)
2. Field Separator. The Default separator is a Comma ','

<bean id="mediationFormat" class="com.sapienter.jbilling.server.mediation.Format">

<property name="tokenizer" ref="*separatorMediationTokenizer*"/>

<!— ‘*separatorMediationTokenizer’ available by default and defines comma as a separator -->*

<property name="fields" ref="vTelecomFormatFields" />

</bean>

**Event Date format, if different from default:**

<bean id="vTelecomDateFormat" class="java.text.SimpleDateFormat">

<constructor-arg value="yyyyMMdd-HH:mm:ss"/>

</bean>

<bean id="mediationRecordLineConverter" class="com.sapienter.jbilling.server.mediation.BasicMediationRecordLineConverter">

<property name="dateFormat" ref="vTelecomDateFormat"/>

</bean>

**Steps Configuration**

Each Step configuration exists by default in jBilling resources.xml file. However, when there are Custom Steps implemented, which is typically true, as it is in this case of V Telecom, The Steps configuration needs to be adjusted for the custom steps.

Out-of-box Steps that exist:

<bean id="userLoginResolverStep" class="com.sapienter.jbilling.server.mediation.step.user.UserLoginResolutionStep" >

<property name="usernameField" value="userfield"/>

</bean>

<bean id="eventDateResolverStep" class="com.sapienter.jbilling.server.mediation.step.eventDate.EventDateResolutionStep" />

<bean id="currentOrderResolverStep" class="com.sapienter.jbilling.server.mediation.step.CurrentOrderResolutionStep" />

<bean id="itemResolverStep" class="com.sapienter.jbilling.server.mediation.step.item.ItemResolutionStep" >

<property name="itemId" value="2800" />

</bean>

....

....

However, for each custom Step, a bean definition for the step is required in resources.xml as follows:

<bean id="vTelecomItemResolutionStep" class=

"com.sapienter.jbilling.server.mediation.step.user.VTelecomItemResolutionStep">

</bean>

**Step Configuration - Replace standard steps with custom steps:**

<util:map id="vTelecomCDRStepsConfig" map-class="java.util.LinkedHashMap">

<entry key="USER\_CURRENCY" value-ref="userLoginResolverStep" />

<entry key="EVENT\_DATE" value-ref="eventDateResolverStep" />

<entry key="CURRENT\_ORDER" value-ref="currentOrderResolverStep" />

<entry key="ORDER\_LINE\_ITEM" value-ref="vTelecomItemResolverStep" />

<!-- note: Your custom step replaced the default one -->

<entry key="PRICING" value-ref="pricingResolverStep" />

</util:map>

<!-- Note: You may define many such Step Configs for each CDR format, and their independent Format Field definitions as well -->

<!-- Step Configs are required for the Resolver to execute -->

<bean id="mediationCdrResolver" class="com.sapienter.jbilling.server.mediation.step.JMRMediationCdrResolver">

<property name="steps" ref="vTelecomCDRStepsConfig" />

</bean>

<!-- Note: 'mediationCdrResolver' above is a key word, only the steps property above needs to point to V Telecom’s custom steps config -->

With the above, we are ready to kick start V Telecom’s mediation using jBilling Mediation 3.0, an jBIlling 3.2 Enterprise Edition Mediation solution.