# Speedment Live Data Agent™

Free your IBM OMoC production data to your Business Users

Since the release of the IBM Order Management on Cloud (OMoC) customers have been challenged to access their production data in near-real time and have faced the frustration of having to use the only solution made available by IBM that moved table-level entity data over FTP to CSV flat files on an hourly basis at best. Consuming data in this form is not only error-prone, but cumbersome and makes it extremely difficult to expose that data to their business users who need to glean important insights from that data trapped in their IBM OMoC production system.

Figure 1 - IBM OMoC Data Extract Agent vs. Speedment Live Data Agent
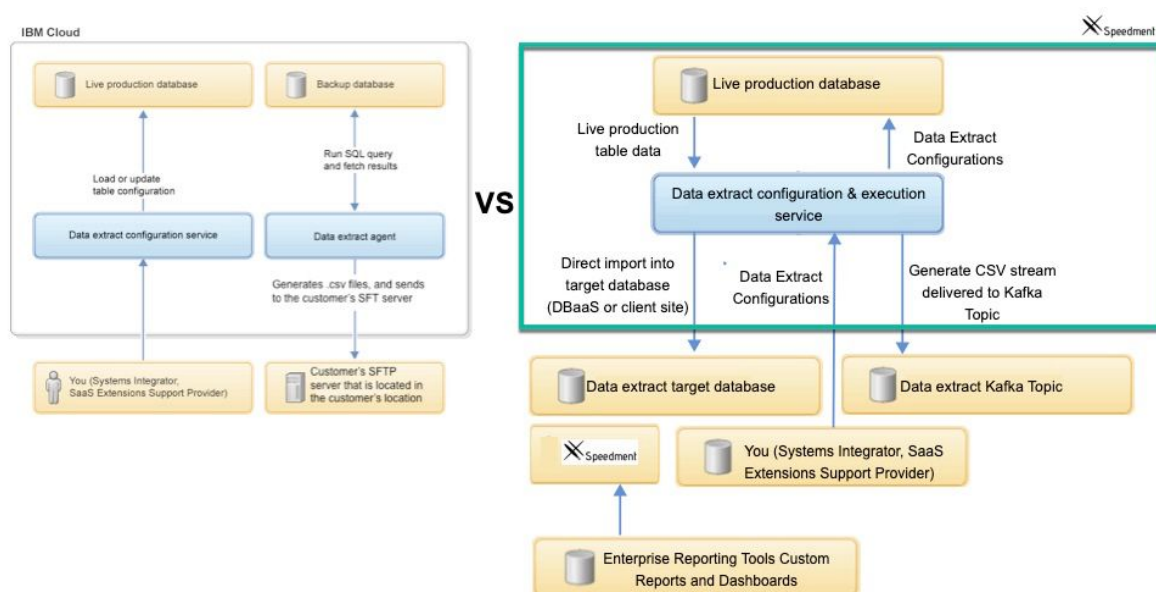
1. The Obstacles

In the world of IBM Order Management on Cloud (OMoC), customers are *not* granted direct access to their production database via JDBC or any other open standard. This creates a major obstacle for customers looking to use their enterprise reporting tools to connect to their order repository to run reports from in near real time.   Although IBM provides an agent for customers that will extract data from select tables, the agent does so by creating FTP files and transferring them to the customer site leaving it up to the customer to convert those files into something their reporting tools can generally use.   Furthermore, there is no way to schedule these FTP extracts any more frequently than one hour at a time, which means, at best, the data is more than an hour old, while further, significant efforts are required to to adjust the routines processing the FTP files to adapt to any changes to the data extract configurations. This is just plain difficult and customers often wonder why they can't have their own data extracted directly into their own shadow copy for use in analytics and reporting and on a near-real time basis. **Well now that's possible with Speedment Live Data Agent for IBM OMoC!**

1.1 A Better solution to Access Customer Data

It's apparent that a better solution is needed for IBM OMoC customers to access their production data in near-real time and the ideal solution would have the following benefits and capabilities.

1. The solution would be a turn-key replacement of the IBM Data Extract Agent that uses cumbersome FTP to transfer table row/column data from the **Backup** IBM OMS system to a remote FTP file system with an agent that moves the data from the **Production** instance over a *Kafka* Topic Queue, directly into a shadow database or both. From a Kafka Topic, it can be easily loaded into a database of the customer's choosing (DBaaS or On-Site) using a Client-Side Kafka Consumer application (provided with the solution) that can automatically move the data on the Kafka Topic directly into a client-side shadow database. That database, in-turn will be directly accessible to your enterprise reporting tools and by Speedment's enhanced streaming tools via JDBC and *all* it's capabilities.

2. The solution should utilize all the same configuration tables that are already available to set up the IBM Data Extract Agent. This means customers can use existing IBM API's to configure what is fed to the Speedment Live Data Agent. For the most part, the documentation for that IBM Data Extract agent should still apply, disregarding the FTP-centricity aspects since FTP is not used.. Those docs are here:https://www.ibm.com/support/knowledgecenter/en/SSGTJF/com.ibm.help.omcloud.administer.doc/tools/c_omc_dataextract.html

3. The solution should utilize Transactional Kafka technology (Kafka is the latest in Scalable Messaging Technology and is used by IBM internally and by many others) to move the data from the IBM OMoC Production DB to a Kafka Topic. The data is delivered to the queue in Comma Delimited Format and the delivery is very fast, very reliable, and very scalable. Using Kafka Transactions, the clients can request only the committed records from the topic and can communicate to the Kafka Partition which records have been consumed. This virtually eliminates any issues with data loss and ensures only what was intended to be consumed to be consumed.

4. The solution should allow customers to "**Override**" the part of the *Speedment Live Data Agent* that moves the data to **Kafka** in CSV format, so the customer can instead, decide to move it over their own ESB technology   The default implementation of this override moves it over Kafka in CSV format and/or directly into a target database you can configure via **customer_override.properties**.

5. Since the data is typically going to be targeted for a Database vs FTP files, the solution should allow the customer to configure table groups and sequence the extracted tables so they can, for example, send the YFS_ORDER_HEADER records before YFS_ORDER_LINE records which have a one-to-many relationship and leverage foreign keys to tie the headers to the lines. This grouping should not require any change to the existing YFS_DATA_EXTR_CFG table to accomplish this.

6. The solution should come with a Client Side Server application that can move the data from the Kafka Topic to a **target** to a database (DBaaS or Local) via JDBC. That client

application should only read the committed records from the Speedment Live Data Agent topic and should be stoppable, and restartable, and should allow multiple consumers to be active to support one-to-many target db instances. It should also have the ability to create the corresponding table schema on the destination database using the YFS_DATA_EXTR_CFG configurations stored and to re-create any of these tables as new columns are added or others dropped.

7. The solution should extend IBM's DataExtract's "First Run" capabilities that is used to synch back any number of days up to the current date/time. This solution should allow customers to trigger a **Reset** to force any Pending or Running tasks refreshed to a given start date dictated by the table's **FirsRunExtractInDays** configuration setting. It should also facilitate a way to tell the downstream client what tables, if any, should be deleted, dropped, created, or dropped and created, or left as is in preparation for getting a new full data synch.

1.2 Conclusion

The Speedment Live Data Agent lays the groundwork for IBM OMoC customers to realize the full potential of their order management data that is today, *trapped* inside the four walls of the IBM Cloud data centers.   Using this revolutionary Data Extract tool that is purpose built for IBM OMoC using all the standard IBM OM Agent framework, customers can finally get at their critical transaction data, in near real-time, quickly, efficiently, and with little to no impact on their production system. Once the data is extracted into the customer-owned databases, they're free to leverage the Speedment tools, or any of their other enterprise reporting frameworks to build the real-time reports..

Finally, if you're an IBM OMoC customer leveraging the IBM Cognos on Cloud solution, you now have a much more flexible alternative that will allow you to use your own internal enterprise reporting tools. Your reports will be more real-time and you can reduce the resources needed to care and feed this less than optimal reporting solution.

2 – The Setup of the Speedment Live Data Agent

2.1 Step 1 – Create the Sample Data Extract Configuration Records

To create the Sample Data Extract Configurations copy and paste the XML below and paste it into the API Tester set up to call the multiAPI. Don't worry about any existing Configuration Records as these will not be disturbed or used going forward. This XML can also be found in the file **manageDataExtractConfigInput.xml** inside the SpeedmentLiveDataAgent.jar file under install.

Name
- com
- docs
- **install**
- META-INF

2.2 Step 2 – Disable Triggering the Original COC_DATA_EXTRACT Agent provided by IBM

IMPORTANT NOTE: You *MUST* disable the Schedule Trigger Message option for the original **Data Extract Agent** provided by IBM as you don't want to run this agent any longer.

Figure 2 - COC_DATA_EXTRACT Transaction

## 2.3 Step 3 - Create a Custom Agent in the General Process Type

Below is an image of what the agent setup looks like in the IBM OMS Applications Manager UI. Effectively we're setting up one Triggerable Task for each table or table group we want to extract. Doing this we can ensure master data is synched separate from transaction data and on schedules that make sense for each entity.

You will start by creating a new transaction in the **General** Repository with **SPEEDMENT_LIVE_DATA** as the **Transaction ID**. Note that you don't need to enter the '**.0002.ex**' portion as that's done for you. Make sure to set it up as a new transaction vs. an extended transaction in the **General Process Type** Repository of **Transactions.** In the **Time Triggered** tab you'll enter the class name for the Speedment Live Data Agent which is:

Java Class: **com.speedment.livedata.agent.SpeedmentLiveDataAgentImpl.**

2.4 Step 4 - Create Agent Criteria Definitions for each Task

Figure - 3.1 - Agent Configuration - 3 Sample Tasks plus the RESET Task
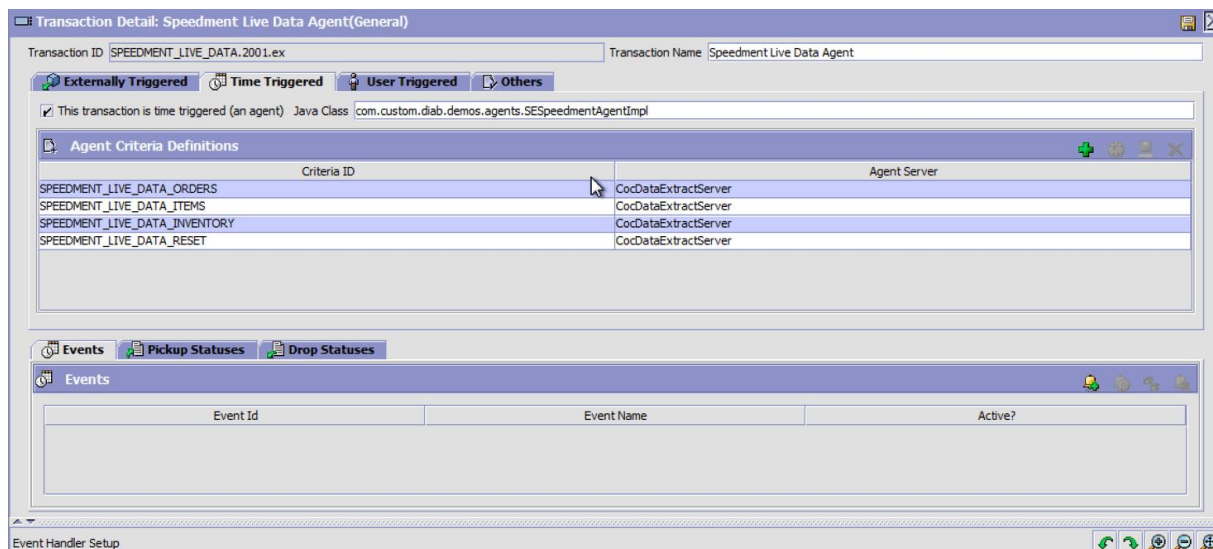
Runtime Properties Setup

The Runtime configuration should resemble all your other agents and should use the already provided **CocDataExtratServer** as the Agent Server. The number of threads should always be set to **1**. This is because of the nature of this agent which uses a db semaphore to prevent the same task from running concurrently, therefore, each task is single threaded. However since we have one CriteriaID for each task (e.g. Orders, Items, Inventory), each task runs on it's own dedicated thread so the agent is, in this way, multi-threaded.

Figure - 3.2 - Example Agent Runtime Configuration
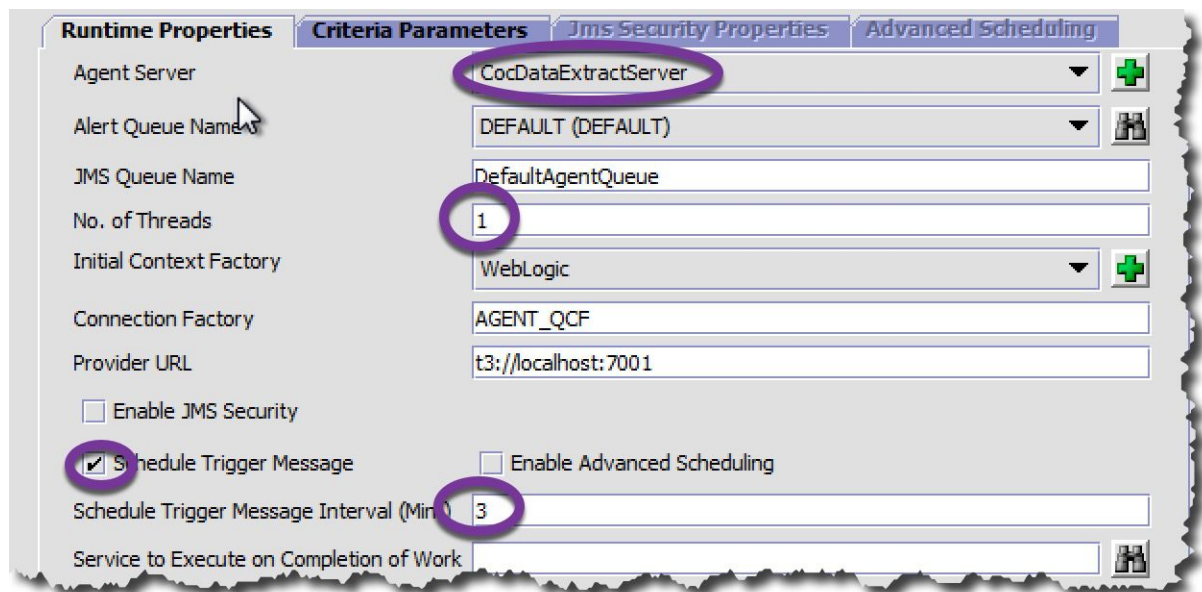
Once you have tested each task and verified they're working manually, then you can choose to Schedule a Trigger Message and set the Interval to reflect the minimum frequency of the all the data extract config records who share the same **TaskId**. So, for example, both the **YFS_ORDER_HEADER** and **YFS_ORDER_LINE** tables have a default **FrequencyInMins=2,** so it would make sense to trigger this agent every 3 minutes. By going one minute above it ensures you that the agent scheduler will not trigger the agent before it's next trigger interval. In general, it won't hurt to trigger the agent more or less frequently as the agent will ignore triggers that occur inside the configured 2 minute interval, and will roll-up all the changes in the configured 2 minute increments (never to exceed the current time) if triggers occur outside the configured 2 minute interval. **IMPORTANT NOTE:** Initially you will not want to schedule any triggers until you have tested each task manually (see section 4.0 Running Live Data Agent).

Beginning in step 2.5, you will want to create 4 distinct **Agent Criteria Definitions** all with the same Runtime Properties as follows:

1) **SPEEDMENT_LIVE_DATA_ORDERS**
2) **SPEEDMENT_LIVE_DATA_ITEMS**
3) **SPEEDMENT_LIVE_DATA_INVENTORY (optional)**
4) **SPEEDMENT_LIVE_DATA_RESET**

IMPORTANT NOTE: that the **SPEEDMENT-RESET** Criteria shown above is not configured as an extract task, but instead is an Internal task that the Speedment Live Data Agent uses to facilitate resetting extract jobs that are not in a valid terminal status (SUCCESS/FAILED), or to force the "First Run" of a job (i.e. extracts **FirstRunExtractDays** worth of records). The Client-side Consumer application will process these RESET requests and performs a specific DB Action prior to executing the "First Run RESET"

Figure 4 - SPEEDMENT_LIVE_DATA_RESET Agent Criteria Setup
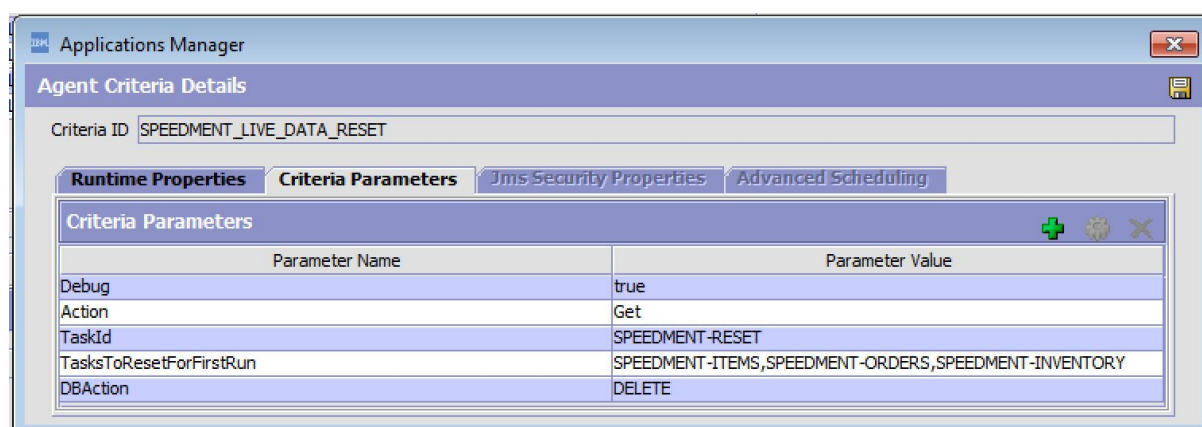
Initially you would want to setup the **SPEEDMENT-RESET** agent to trigger manually and you should always check to make sure the **TasksToResetForFirstRun** are correct before manually or scheduling trigger messages. It's worth pointing out that each client consumer can override the **DBAction** using **-DDBAction=<DBAction>** on the JVM arguments when starting the client-side consumer.

The Client-Side consumer application is provided with the Speedment Live Data Agent and runs on both LINUX and Windows servers. It's primary function is to take the Live Data streams from the Kafka Topic that the Live Data Server is streaming to, and moves the data into your shadow database via JDBC. The Client also processes the **SPEEDMENT-RESET** task to take a corresponding action on the client-side shadow database for each of the tasks/tables defined in the **TaskToResetForFirstRun** parameter.

You can have as many Client-Side consumers as you wish as long as each is configured with it's own distinct Kafka Group Id so the data from the server can be replicated to more than one shadow copy. For example, if two consumers of the Live Data Stream are required to move the data to distinct and different shadow databases, you would simply configure the **speedment.consumer.kafka.group.id** property differently for each consumer.

Consumer 1 - speedment.consumer.kafka.group.id=YOUR_COMPANY-MKTING
Consumer 2 - speedment.consumer.kafka.group.id=YOUR_COMPANY-ECOM

2.5 Step 5 - Complete Agent Criteria Definitions for each Task

Criteria Parameters Setup

The **Criteria Parameter** configurations should consist of one agent criteria record for each **TaskId** configured and created in **Step 1** using the **CocDataExtractConfig** service as defined in the Service Definition Framework (SDF). The Initial sample extract records includes 3 Tasks Id's (excluding the **SPEEDMENT-RESET** task).   They include the SPEEDMENT-ITEMS, **SPEEDMENT-INVENTORY** and **SPEEDMENT-ORDERS** Tasks.   A sample of the SPEEDMENT-ORDERS task is depicted below:

Figure 5.1 - Sample SPEEDMENT-ORDERS Criteria Parameters

Assuming you've created the Data Extract Configuration Records in **Step 1** to capture the **YFS_ORDER_HEADER** and **YFS_ORDER_LINE** tables, the data extract config records would look as follows:

Figure 5.2 - Sample SPEEDMENT-ORDERS Task Configuration

```xml
<?xml version="1.0" encoding="UTF-8"?>
<DataExtractConfigList TotalNumberOfRecords="2">
  <DataExtractConfig
Columns="ORDER_HEADER_KEY,DOCUMENT_TYPE,ENTERPRISE_KEY,BUYER_ORGANIZATION_CODE,SELLER_ORGANIZATION_CODE,BILL_TO_ID,SHIP_TO_ID,ORDER_
DATE,ORDER_NAME,ORDER_NO,ORDER_TYPE,ENTRY_TYPE,AUTHORIZED_CLIENT,REQ_SHIP_DATE,REQ_DELIVERY_DATE,CUSTOMER_FIRST_NAME,CUSTOMER_LAST_N
AME,CUSTOMER_EMAILID,CUSTOMER_PHONE_NO,CUSTOMER_ZIP_CODE,CURRENCY,TAX,PAYMENT_STATUS,TOTAL_AMOUNT"
DataExtractConfigKey="SPEEDMENT-ORDERS-10" FirstRunExtractInDays="30" FrequencyInHours="0" TableName="YFS_ORDER_HEADER"
TaskId="SPEEDMENT-ORDERS"/>
  <DataExtractConfig
Columns="ORDER_HEADER_KEY,ORDER_LINE_KEY,PRIME_LINE_NO,SUB_LINE_NO,ITEM_ID,UOM,PRODUCT_CLASS,DELIVERY_METHOD,ITEM_DESCRIPTION,ITEM_S
HORT_DESCRIPTION,SHIPNODE_KEY,SCAC,CARRIER_SERVICE_CODE,RECEIVING_NODE,ORDERED_QTY,OTHER_CHARGES,LINE_TOTAL"
DataExtractConfigKey="SPEEDMENT-ORDERS-20" FirstRunExtractInDays="30" FrequencyInHours="0" TableName="YFS_ORDER_LINE"
TaskId="SPEEDMENT-ORDERS"/>
</DataExtractConfigList>
```

Note that the **TaskId** for each of the Data Extract Configuration records is identical (**SPEEDMENT-ORDERS**) which makes them part of a table group. The **DataExtractConfigKey** attribute is explicitly set to include the **TASKID-SEQNO**, where **TASKID**=the **TaskId** and **SEQNO**=Sequence of the table in the Extract Job. So for example, in this case the **SPEEDMENT-ORDERS-10** (**YFS_ORDER_HEADER**) job is always extracted before **SPEEDMENT-ORDERS-20** (**YFS_ORDER_LINE**) job to ensure the header records will exist on the target database before the order lines are extracted and inserted.

SPEEDMENT Criteria Parameter Data Used by Live Data Agent Tasks

EXTRACT TASK CRITERIA	All tasks configured in YFS_DATA_EXTR_CFG
Action	Get (Always use this value)
Debug	false (WIll output helpful INFO Log messages if true)
TaskId	TaskId from Data Extract Config Record above
SimulatedRunTime	0 (Always use this value - speedment use only)
FetchLimit	Limit Fetch of Records to this Value (Default=500)
SPEEDMENT-RESET ONLY	Additional Criteria Below is Optional
DBAction	DELETE | DROP | CREATE | DROPANDCREATE | NONE
TasksToResetForFirstRun	TaskId-1,TaskId-2,TaskId-3

Note the **FetchLimit** limits the extract query result sets for a given task to the **FetchLimit** number of records (e.g. 500). If not specified or set, a default limit of 500 records is used. It is recommended that you set this limit to a value that approximates the number of records during non-peak periods you expect to process in the frequency period defined for the given extract task. For example, if you're trying to set the optimal **FetchLimit** for the **SPEEDMENT-ORDERS** task, you might use the following calculation:

*FetchLimit = Non-Peak Order Lines/HR / 60 * Frequency In Mins*

Example:
Non-Peak Order Lines/HR: **30000**
Frequency of Task: **2** (In Minutes)
FetchLimit = (**30,000** / 60 * **2**) **= 1000**

This will ensure the extract jobs are keeping up with your order volumes yet, if a spike occurs that exceeds the expected non-peak orders, then the extract will catch up when the volumes fall back to your non-peak volumes. In general, it's better to run the extract tasks more frequently than less frequently to reduce the amount of data processed by any one task thread. Generally speaking, if your **FetchLimit** is above **5000**, then it is recommended that you add an index on the **MODIFYTS** field on all the tables you're extracting above this limit. Not doing so *could* have an impact on your production system's *performance*.

Note the DBAction is only used by the SPEEDMENT-RESET transaction. This action will be taken on the client side database for the tasks listed in the **TasksToResetForFirstRun** unless that action is overridden by the Client-Side consumer.
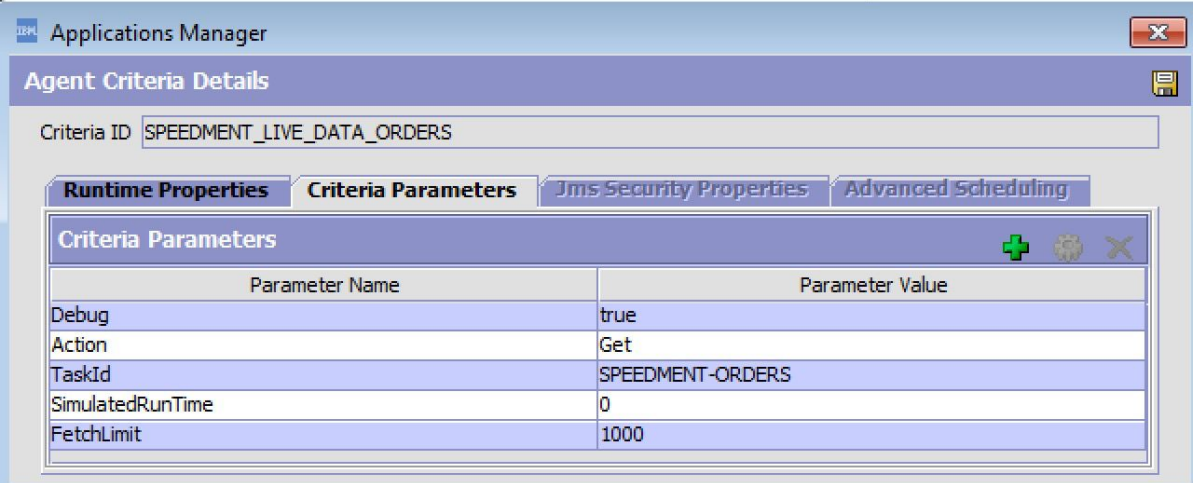
Figure 4 - SPEEDMENT-RESET Criteria
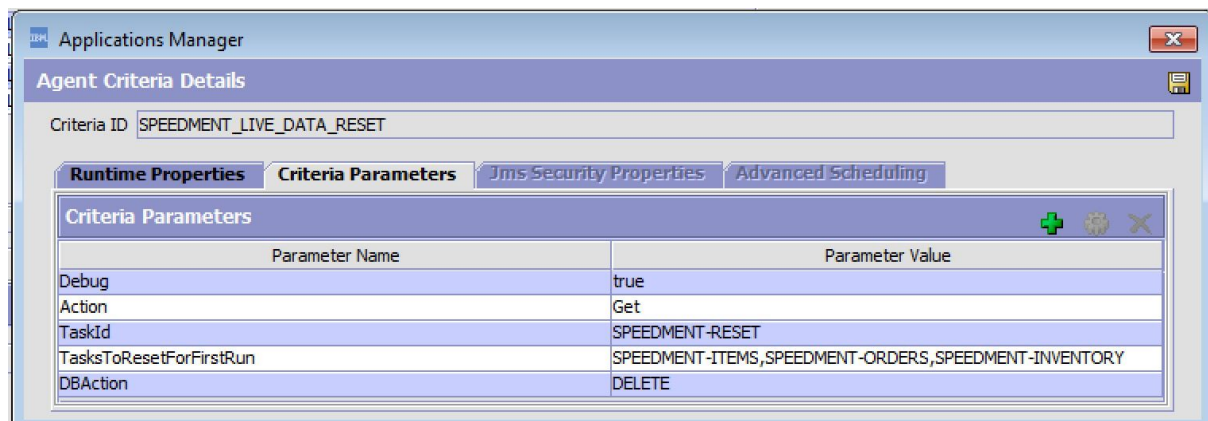
Note the TasksToResetForFirstRun is only used by the SPEEDMENT-RESET transaction. Include any task id that you want to take the corresponding **DBAction** on. All tables associated to that task id's listed will be affected by the corresponding **DBAction**.

The **CREATE** only occurs if the tables associated with the provided TaskId's don't exist and the **DROP** only occurs on tables that do exist. The **DROPANDCREATE** combines both a **DROP** and **CREATE** action in one **RESET** trigger.

IMPORTANT NOTE - The **CREATE** task on the client side application automatically generates the **SQL CREATE TABLE** schema based on the **Columns** chosen for the given Data Extract configuration so if you change the set of columns (add or remove) or add new tables to a given Task, you should always trigger the **SPEEDMENT-RESET** for those tasks that are affected using the **DROPANDCREATE DBAction**.

You can also configure more than one **RESET** criteria to be more granular (e.g. RESET-ORDERS) which might only reset the SPEEDMENT-ORDERS task. You only have to ensure the **CriteriaID** contains "**RESET**" in the CriteriaID itself to be considered a RESET task.

Figure 5.3 - SPEEDMENT-ORDERS Criteria
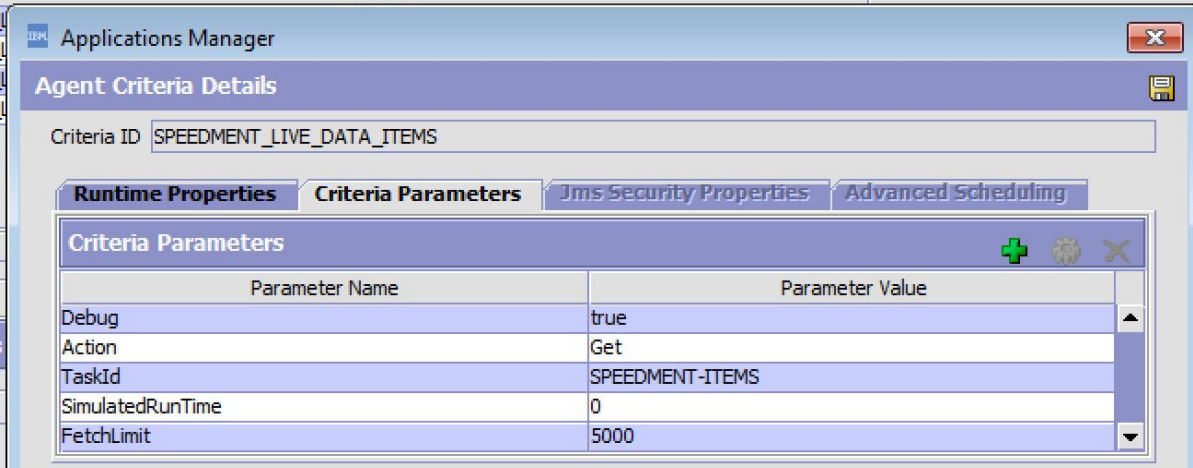
Figure - 5.4 - SPEEDMENT-ITEMS Criteria
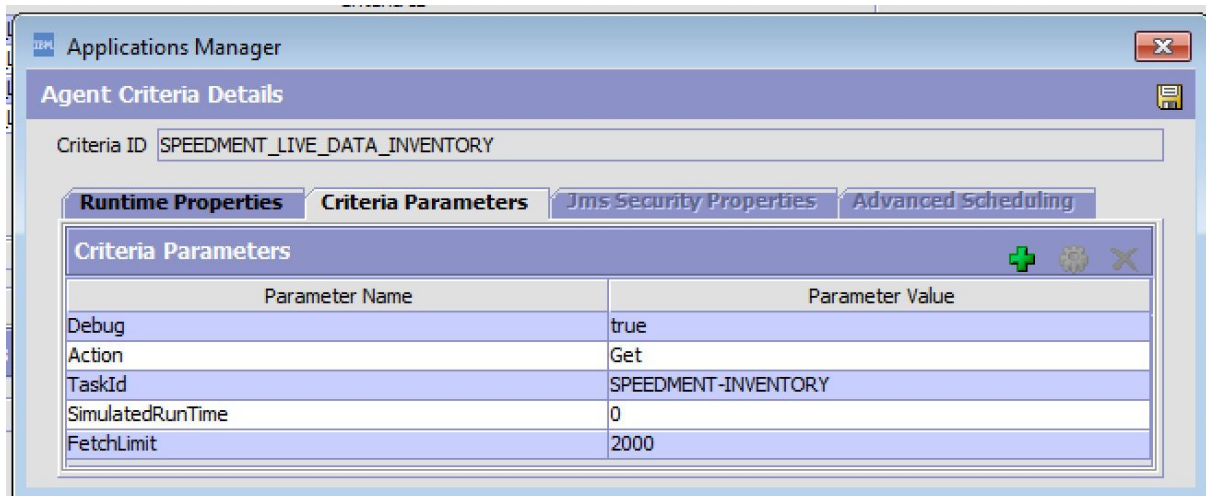
Figure - 5.5 - SPEEDMENT-INVENTORY Criteria
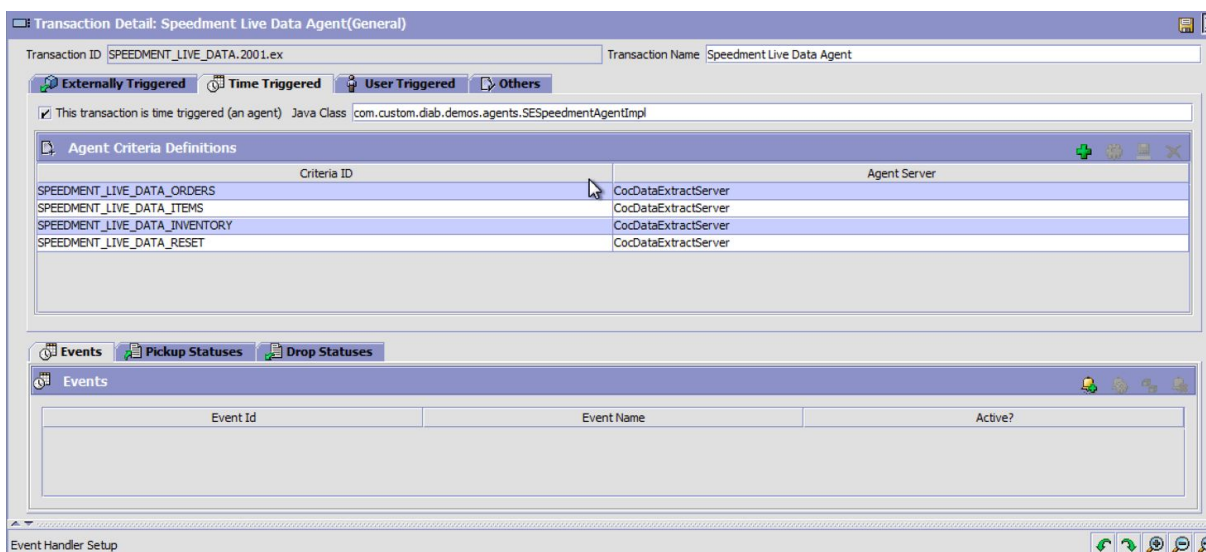
When completed your agent configuration should look as follows:

Figure - 5.6 - Finalized Agent Configuration

2.6 Step 4 - Install Properties for the Live Data Agent

A set of **cutomer_overrides.properties** is available for legacy customers that still use the customer_overrrides method to extend the System Properties. Note these properties are not manageable if added to the **customer_overrides.properties.**

To make these properties manageable via the Systems Management Console, there is a **managePropertiesInput.xml** file that you will find in the install folder that can be used to add these properties to the database as manageable properties. If you do this you *must still add* them to the **customer_overrides.properties**, however you can manage them via the Systems Management Console.

```
#############################################################################
# IMPORTANT: PRODUCER PROPERTIES MUST BE COPIED TO customer_overrides.properties
# TEST KAFKA INSTANCE - LOCAL
#############################################################################
yfs.speedment.producer.kafka.enabled=true
yfs.speedment.producer.kafka.topic=speedment-topic
#yfs.speedment.producer.kafka.bootstrap.servers=[KAFKA_BOOTSTRAP_SERVER]:[PORT]
yfs.speedment.producer.kafka.key.serializer=org.apache.kafka.common.serialization.StringSerializer
yfs.speedment.producer.kafka.value.serializer=org.apache.kafka.common.serialization.StringSerializer
#############################################################################
# IBM CLOUD - Connection Parameters - Need your bootstrap servers and api_key to use
#############################################################################
yfs.speedment.producer.kafka.bootstrap.servers=[KAFKA_BOOTSTRAP_SERVER]:[PORT]
yfs.speedment.producer.kafka.client.dns.lookup=use_all_dns_ips
yfs.speedment.producer.kafka.client.id=SPEEDMENT
yfs.speedment.producer.kafka.security.protocol=SASL_SSL
yfs.speedment.producer.kafka.sasl.mechanism=PLAIN
yfs.speedment.producer.kafka.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="token" password="[API-KEY]";
yfs.speedment.producer.kafka.ssl.protocol=TLSv1.2
yfs.speedment.producer.kafka.ssl.enabled.protocols=TLSv1.2
yfs.speedment.producer.kafka.ssl.endpoint.identification.algorithm=HTTPS
yfs.speedment.producer.kafka.acks=-1

yfs.speedment.producer.database.enabled=false
yfs.speedment.producer.database.DstDBType=DB2
yfs.speedment.producer.database.DstDBServer=[DB Server IP/DNS]
yfs.speedment.producer.database.DstDBPort=50000
yfs.speedment.producer.database.DSTDatabase=OMDB
yfs.speedment.producer.database.DstDBUsername=[DBUser]
#your database password can be been encrypted by the live data agent's encrypter class - see client server help page
yfs.speedment.producer.database.DstDBPassword=encrypted:[ENCRYPTED_PASSWORD]
yfs.speedment.producer.database.DstDBSchema=OMDB

# columns that need to be url decoded - typically description columns
yfs.speedment.urlencoded.columns=ITEM_DESCRIPTION,ITEM_SHORT_DESCRIPTION,SHORT_DESCRIPTION,DESCRIPTION,EXTENDED_DESCRIPTION,IMAGE_ID

# encryption interfaces
#yfs.security.encrypter.class=<value>
#yfs.security.propertyencrypter.class=com.speedment.livedata.encrypter


#############################################################################
# IMPORTANT: PRODUCER PROPERTIES MUST BE COPIED TO customer_overrides.properties
#############################################################################
```

Note the **yfs.speedment.producer.database.DstDBPassword** property can be encrypted by the Live Data Client application. Run the live data client with the **-help** option to see how this is done.

IMPORTANT: These two properties determine what mode(s) the live data agent runs in

yfs.speedment.producer.kafka.enabled=true|false
yfs.speedment.producer.database.enabled=true|false

If Kafka option is Enabled then all the data records will be written to a Kafka Topic determined by the speedment-livedata.producer.properties. If the Database option is Enabled, then in all records will be directly written from the Live Data Agent directly to a database, however, you should keep in mind that there is additional overhead to writing to a Database vs. writing to a Kafka topic.

IMPORTANT: Use of Kafka vs. Database Mode is Strongly Encouraged

Using the **Database mode,** writes data directly to the target database from the Live Data Agent and eliminates the need for using the client-side server. While this mode is allowed, this mode can be prone to more data extract exceptions/failures if the client database is not available, and it can degrade the performance of the Live Data Agent if there is significant latency in accessing the target database. Using the **Kafka mode** instead eliminates any issues that may result from the Client Side database not being available or accessible and any latency issues that might result depending on where the destination database is located relative to your OMS instance.

2.7 - Step 5 - Create a new Custom API to Call the Extract Worker API

The Live Data Agent allows you to configure your own "**executeJob Worker**" API which will get called every time the Live Data Agent has a record to write to the target. The worker provided out-of-the-box will write all the data records over the configured **Kafka** topic, or write directly to a shadow database based on the server's producer properties.

Open up the SDF services in the General Repository and look for the CocDataExtract Group of services.
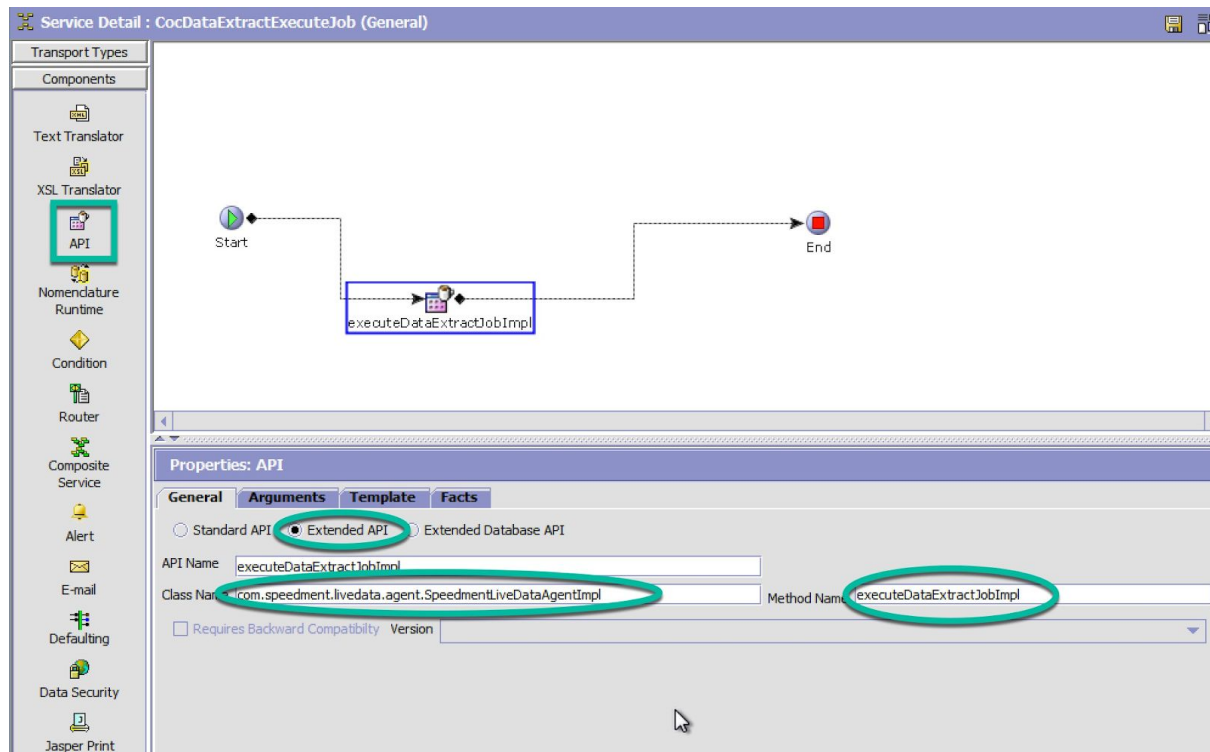

Create a new Synchronous Service in the **Coc_Data_Extract** Group of Services as shown here:

This service will, by default, be setup to call the built-in Custom API provided as part of the **Speedment Live Data Extract Agent** and so you'll configure the service to call an API as shown here:

Figure 7 - CocDataExtractExecuteJob Custom API

Drag API component into SDF, and select the Extended API and then enter in:
Api Name: *executeDataExtractJobImpl*
**Class Name:** *com.speedment.livedata.agent.SpeedmentLiveDataAgentImpl*
Method Name: *executeDataExtractJobImpl*

Save the Service and you should see it added to the group.

3.0 - Provisioning IBM Cloud Events (Kafka) Instance

1) Create an IBM Cloud account and provision an IBM Event Streams resource in your cloud instance. There are three options which include a Free test instance, or pay as you go instanances for your production environment.
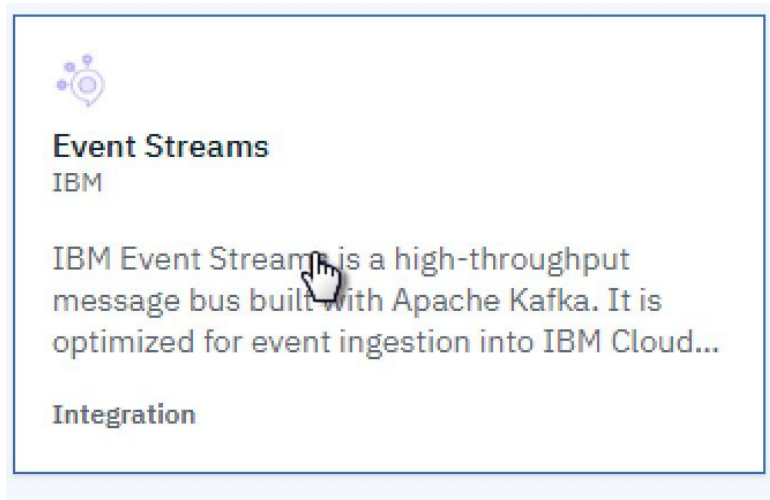
2) Once provisioned click into the Credentials for you new service and copy the kafka.boostrap.servers property and the api-key and paste them into the customer_overrides.properties where indicated and redeploy the properties file.
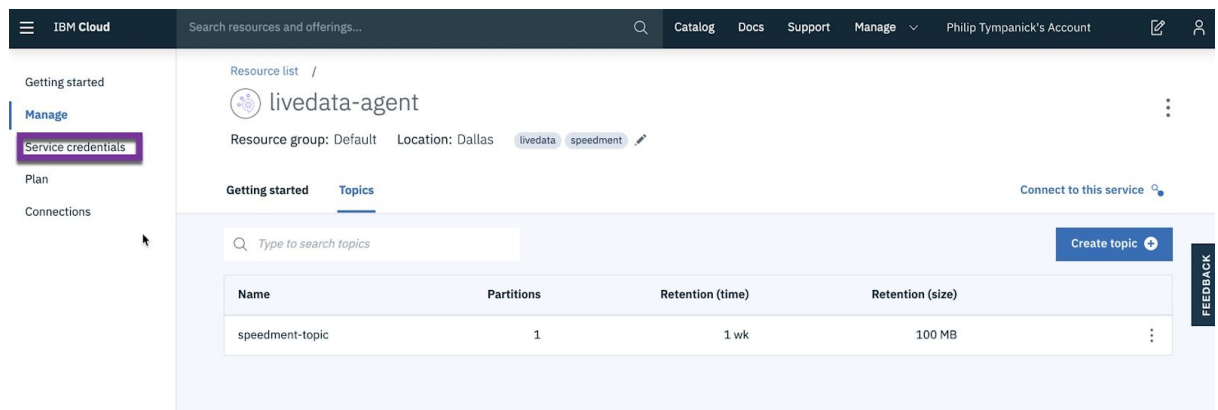
3) Copy and Paste the credentials into your Client-Side properties file where indicated and try starting the Client Server to make sure it can connect to your IBM Cloud instance.

4.0 - Installing Kafka if an instance not already Available

4) Please refer to the following:
   https://www.javainuse.com/misc/apache-kafka-hello-world
5) Once you have the hello-world example working then you're ready to update your speedment properties (server & client) to reflect the server details you've set up.
6) Update speedment properties to reflect the agent server DNS etc. and redeploy the customer_overrides.properties and test the client application.

5.0 - Running the Live Data Agent Server

Start an agent server process using the **CoDataExtractServer**

Launch a LINUX SHELL and run the following:

runtime: $STERLING_HOME/bin/agentserver.sh CocDataExtractServer

Important Note: Before you run the live data agent server for the **FIRST** time, disable all the Schedule Trigger messages for each task set up. You'll want to test each task individually by manually triggering each distinct task when you're ready to test run the task.

6.0 - Running the Live Data Client Server

The Live Data Client jar should be installed on a local client-side server along with a shadow database. Try SQLite if you don't have a database of your own to use. https://www.sqlite.org/index.html. The client supports DB2, Oracle and SQLite today.

Update client properties with the DB connection parameters. The sample properties file works with SQLite database but you can use any other DB you prefer as long as it's JDBC compliant.

The Speedment Client Application is a Java Program and the SpeedmentLiveDataClient.jar file is a runnable JAR.

There is a Linux SHELL or MS-DOS batch file you can use to start the client-side application. You will need to tweak these files to match your deployment environment.
startSpeedment.sh, speedmentServer.sh
or
startSpeedment.bat

Client Application Usage:
    SpeedmentLiveDataClient [-props:<properties_file>] [-save:<save_message_file>]
        [-generatePassword] [-verbose] [-help|-usage|-?]

    -props:<properties_file>
        Allows to specify the properties file name that contains client
        information about the Kafka consumer and other important properties.
        If <properties_file> is not specified, this program
        attempts to find and use 'speedment-livedata-client.properties' on the root
        of classpath.

    -generatePassword
        Encrypts the plain text property speedment.consumer.database.DstDBPassword
        and prints out its encrypted value to replace in the properties files

-save:<save_message_file>
   Indicates that incoming messages should be just saved to a file
   without being processed by the consumer. <save_message_file>
   allows you to specify the name of the file where all the incoming
   messages are to be saved.
 -flush
   Flush Kafka Topic of all outstanding messages

-verbose
   Runs in verbose mode (prints extra info on execution).

-help|-usage|-?
   Outputs this usage information.

Example:
   SpeedmentLiveDataClient -props:speedment-livedata-client.properties
   SpeedmentLiveDataClient -props:speedment-livedata-client.properties -save:message.txt
-verbose
   SpeedmentLiveDataClient -props:speedment-livedata-client.properties -generatePassword

VMArgs Available:
   -DBAction=DELETE|DROP|CREATE|DROPANDCREATE -DSQLDebug=true|false
   Where:
   DBAction overrides value coming from Agent Server.
   SQLDebug will display SQL being executed on the Destination database while running.

Creating your Shadow Database Schema

Once your client application is running it will try to connect to the database. At this point you
should manually trigger the SPEEDMENT-RESET transaction and ensure that the tables for all the
tasks are created on the client database. Make sure you start the Client with the VMArg
-DDBAction=DROPANDCREATE. If you don't see the shadow data tables created in your shadow
database look at the log files on the client to determine what happened.

It is recommended you use manual triggering of each task until you're satisfied that you have
everything working as you expect.