Voice WMS Troubleshooting - 2012.1 and later

VoiceConsole device side logs

Traces of the RF terminals through the MTF framework will often tell you exactly what keys were pressed by the operator and what data was scanned by the scanners. In contrast, with Voice terminals, from the WMS side, all we have is the data sent from the devices in the Integrator transactions. It doesn't give much insight into exactly what the voice user was doing during the error. The VoiceConsole device log will provide this information. Below is an except of one of those logs:

Add sample log file text here.

We see dialog spoken from the task to the user, any button presses or spoken text from the user back to the system, and also the raw data in the transactions sent to the WMS server.

User interaction should be checked to ensure dialog is spoken correctly, and any expected user data was spoken correctly.

You should also be checking to see that the device properly sent a BSD transaction to the WMS, and the WMS responded with data when appropriate. Often issues here will display an incorrect terminiting character, or connection manager ports not properly listening. The only insight we have on these types of issues are from the VoiceConsole Logs.

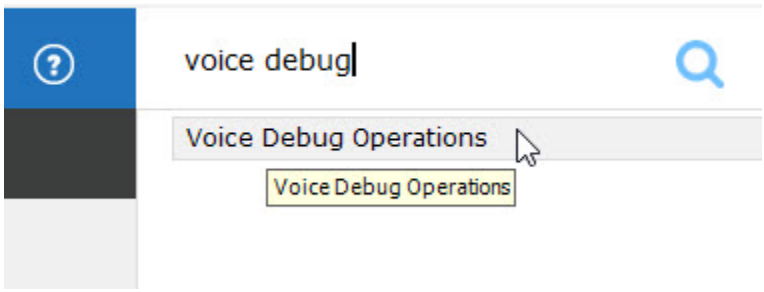
The logging Procedure differs depending on which version of VoiceConsole you are using but the below links should reference the most recent versions:
VoiceConsole 5.1
VoiceConsole 4.2
VoiceConsole 3.1
VoiceConsole 3.0

Tracing a Moca Connection Manager

When the Voice tasks were moved to Java socket code, it became impossible to enable tracing without bouncing the Voice Tasks. All logging is enabled in Task Maintenance, by locating the VOC_LUT and VOC_ODR tasks in the list, and entering a file name and trace level for the task. It is recommended that you stop the tasks before making these changes, and ensure the old tasks have shut down. Then make the changes for logging and start the Voice tasks.

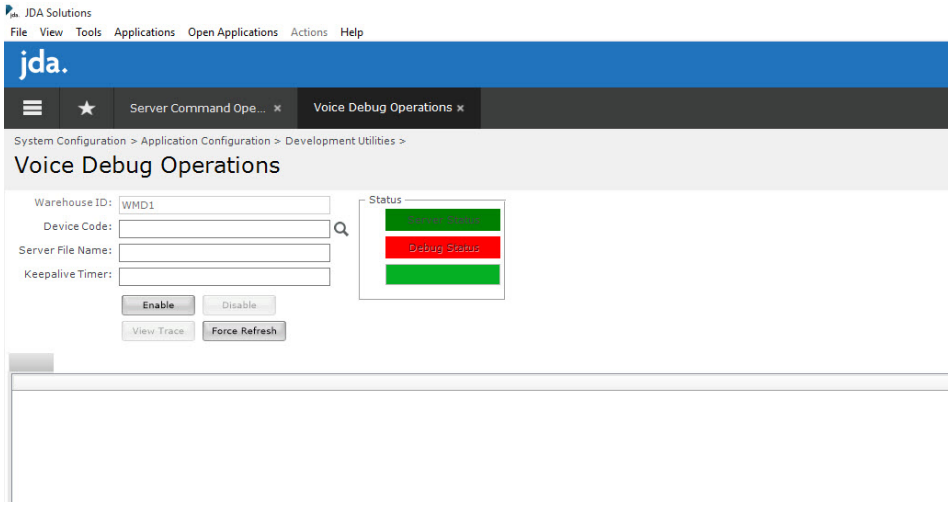
Voice Debug Operations

You will need to make sure you have permissions for Voice Debug Operations to begin with. From the RedPrairie GUI, Click the Locate button at the top, and search for "voice debug operations" or any partial variations of that.



Select "Voice Debug Operations" from the list and click OK.
(Note: Alternatively, you could navigate the menu structure on the left. This screen may be found under System Configuration, Application Configuration, Development Utilities, Voice Debug Operations.)

Once open, you will see the following screen.



In the **Device Code** field, enter the voice device number. This will be on a sticker on the device (if it is a Talkman device), or on the screen if it is a device with a screen (Note: Vocollect Voice for Handhelds must be running).
Alternatively, you can click the Search icon

and locate your Voice Device from the list.

In the **Server File Name** field, enter a unique filename. You do not need the .log extension at the end, it will be added automatically by the server.

In the **Keepalive Timer** field, enter a large number (ex. 1000000). If it is a low number, this can cause issues with logging.

When all fields have been entered, click the enable button.

> You do not need to trace an entire session for voice users. Often times, if an error occurs, your operator the operator will not be able to continue past the error without saying ready to replay the last transaction. Before the operator says ready, start tracing, wait for the error to occur again, then stop tracing.

> Please do not close the Voice Debug Operations screen while tracing is enabled. Tracing stops immediately after closing the screen.

> Do not run traces for very long. These traces have a lot of data in them, and they can quickly fill up drive space. Try to narrow the trace to a specific time the error occurs, or if you can reproduce the issue, start tracing just before it will occur and stop it after.

Identifying long running transactions

The following SQL statement can be run to find voice events that are considered "long running" You'll need to tweak the time stamps and possible the rows that are returned:

```
[select ed.evt_data_seq ,
        ed .evt_id ,
        ed .evt_dt as started ,
        idh .cmpl_dt as finished ,
        idh .ifd_data_seq ,
        (to_number( idh. cmpl_dt - ed.evt_dt) *86400 ) as seconds
   from sl_evt_data ed ,
        sl_ifd_data_hdr idh
  where ed.ins_dt > to_date('6/10/2018' , 'mm/dd/yyyy')
    and ed.ins_dt < to_date('6/11/2018' , 'mm/dd/yyyy')
    and ed.evt_data_seq = idh .evt_data_seq
    and ed.sys_id = 'VOICE'
    and rownum < 500
    and cmpl_dt is not null
  order by seconds desc]
```

This query finds the timestamp for when the event is created vs. the time stamp on the result IFD. Basically it is identifying processing time on the server. Voice, by default, will timeout waiting after 20 seconds, so anything that took longer then 20 seconds to process resulted in an error to the user.
In certain cases, you can identify a particular transaction that is consistently long running (for instance SAL_VOC_PRC_INV_DEP is the inventory deposit transaction) You can focus a trace on that particular event and see what is taking the longest to process.

> On earlier systems (2012 and newer) the ed.sys_id was actually 'VOCOLLECT'

Identifying processing errors and socket errors on transactions

The following SQL, in the first line, will pull the number of transactions where there was an error in processing.

The second line will show the total number of transactions.

The third line pulls the socket errors where the endpoint no longer exists (i.e. error performing send on socket). This typically means the transaction has taken so long that the voice device timed out and closed the socket.

```
[select count(*) from sl_dwnld where sys_id = 'VOICE' and dwnld_stat_cd =
'EPROC']
[select count(*) from sl_dwnld where sys_id = 'VOICE']
[select count(distinct dwnld_seq) from sl_msg_log where stat = 3701]
```

A more descriptive query for all of the integrator message log errors, the error description, and the message log data to help back-trace the issue.

```
[select distinct sml.dwnld_seq,
        sml.evt_sys_id,
        sml.stat,
        lmc.mls_text,
        sml.msg_log_data
   from sl_msg_log sml,
        les_mls_cat lmc
  where concat('err', sml.stat) = lmc.mls_id
    and lmc.locale_id = 'US_ENGLISH'];
```

Enabling log rotate for socket transactions

Creating a socket trace for the VOC_LUT and VOC_ODR server is a great way to get details of integrator overhead and network response time. It creates a very large file, though, often in excess of 300MB for a small pick for a single user. To help alleviate that issue, we utilize the log4j RollingFileAppender.

```
<appender name="MyRollingFileAppender"
class="org.apache.log4j.RollingFileAppender">
    <param name="fileName" value="myRollingFile.log" />
    <param name="maxFileSize" value="500MB" />
</appender>
```

This sample appender should allow you to create a rolling log file that does not grow to be more than 500MB. For more documentation on RollingFileAppenders see this document. And for more information on how to use MOCA to configure your appenders see this document.

To do this for the voice tasks, modify the **%MOCADIR%\data\logging\logging.xml** to add appenders for the voice tasks. In the below example, 10 files of 100MB each will be created for each task. LUT and ODR are two separate listeners, so you should plan for up to the maximum size of these logs so you do not consume all space on the log directory. In this example we will be using up to 2GB of space.

```
<?xml version="1.0" encoding= "UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration>

  <appender name="TraceFileAppender"
class="org.apache.log4j.DailyRollingFileAppender">
    <param name="datePattern" value="'.'yyyy-MM-dd" />
    <param name="append" value="true" />
    <param name="encoding" value="UTF-8" />
  </appender>

  <appender name="VOC_ODR" class="org.apache.log4j.RollingFileAppender">
    <param name="maxFileSize" value="100MB" />
    <param name="maxBackupIndex" value="10" />
    <param name="append" value="true" />
    <param name="encoding" value="UTF-8" />
    <param name="file" value="${LESDIR}/log/VoiceODRTask_new.log" />
  </appender>
  <appender name="VOC_LUT" class="org.apache.log4j.RollingFileAppender">
    <param name="maxFileSize" value="100MB" />
    <param name="maxBackupIndex" value="10" />
    <param name="append" value="true" />
    <param name="encoding" value="UTF-8" />
    <param name="file" value="${LESDIR}/log/VoiceLUTTask_new.log" />
  </appender>

  <logger name="com.redprairie.moca.socket">
 <priority value="TRACE"/>
    <appender-ref ref="VOC_ODR" />
 <appender-ref ref="VOC_LUT" />
  </logger>


  <root>
    <!--<appender-ref ref="TraceFileAppender" />-->
 <priority value="TRACE"/>
    <appender-ref ref="VOC_ODR" />
 <appender-ref ref="VOC_LUT" />
  </root>

  <logger name="org.eclipse.jetty">
        <level value="info"/>
  </logger>
</log4j:configuration>
```

Additional details can be found here Tracing and Logging 2012.2.

If you see the log file write a few lines and then stop, you may be experiencing a bug. Please reference: ~~MOCA-5547~~. It is possible to work around this issue by not using unique sessions (removing -s from the command line for the task). Additionally, while not specific to voice tasks, if you are working with jobs instead of tasks on 2012.2, reference MOCA-6077 for the fix.

It is possible to send all voice traffic through the VOC_LUT port if you want to keep all data in a single file. Since both tasks load all voice transactions, there is no harm in setting up your LUT and ODR ports on the voice task to the same port number.