# Troubleshooting OMA DM provisioning

|  |  |
|---|---|
| Document version: | 1.0 |
| Date: | 2010-07-02 |
| Circulation/Restrictions: | Internal/Excitor partners |
| Applies to: | DME Server 3.5 or later |

## Table of contents

## Disclaimer

# Troubleshooting OMA DM provisioning

This document is for DME administrators who have run into problems when bootstrapping devices (phones) and provisioning software to devices using DME. It is a short introduction to debugging OMA DM bootstraps and installations.

As stated throughout the DME documentation, phone manufacturers implement the OMA standard in different ways. There may even be differences between phone models of the same brand. Due to this, and due to the fact that each customer's setup of firewalls and DMZ is unique, there is a large number of possible OMA provisioning scenarios. Some of these scenarios cause problems. We hope that this guide will help you find the reasons for your OMA DM difficulties.

## What happens in the bootstrap?

### Bootstrapping

The DME server sends an OTA OMA message as an SMS (text) message. This is a special type of message – it contains all required settings for creating an OMA account on the device. This includes the DME DM server address, login name and password, among other things. OMA messages sent from DME also include an instruction to the device to contact the server right after creating the account. Users of *Symbian S60* devices see this as a question that pops up right after saving the settings. *WM devices* do not ask the user – they just connect. What has been described until now is an *OMA DM bootstrap*. When we talk about bootstrapping a device in DME, we include a little bit more than that, as we also create an *explore job* for the device.

#### The explore job

We have found that we cannot rely on the information supplied by the device concerning what type and model it is, so we need to explore the device ourselves. Another reason to do this is that we must consider the fact that the device manufactures may change the DM client on the phone. So to avoid maintaining separate information for each device in DME about how to install software on it, we ask the device for some specific information that will tell us which installation procedure to follow. The *explore job* checks for the existence of some nodes in the *DM tree*. This will tell DME if the device is a Windows Mobile, Java, or Symbian device. DME uses this information to *choose an installation method* and *pick the right DME client* to install.

### Installing

If you chose to install DME on the device along with bootstrapping it, DME will create an *installation job* immediately after completing the bootstrap. This will usually be done in the same session as the explore job described above. In some cases, the connection is lost after the bootstrap has finished. When the connection is lost, the installation job is saved to the DME database, and will enter the normal process for restarting a session with the client device.

### Starting a session with a device

The DME server is not allowed start a session with a device - the device has to be the initiating part. So in order to start a session, the server sends an SMS message to the device. This is called an *initiate message*, and it tells the device to start a session with the server whose name is part of the message. The device will check with the OMA account on the device to see if such a server exists. On S60 devices, the device might ask the user to allow opening a session.

## Debugging

When something goes wrong when bootstrapping a device or installing an application, there are two things you go to find information about what happened, apart from information you can find in the DME web administration interface. Be sure first to check the **Installation log** and **Errors** sections in the **Provisioning** tab.

### Finding OMA jobs in the DME database

By running the following SQL statement on the DME database, you get a list of existing jobs for a given device. You can get different results by changing the **`WHERE`** clause. We have included a number of possible **`WHERE`** clauses, and you can include or exclude any of them by adding or removing the comment marker **`#`**.

```
SELECT j.created AS job_created, j.jobID AS jobID,
       j.status AS job_status, t.id taskID,
       t.state AS task_state, o.command AS operation_command,
       o.statusCode AS operation_status,
       n.target, n.format, n.type, nd.data, d.devicePK
  FROM dm_job AS j
  JOIN dm_task AS t ON t.job = j.jobID
  JOIN dm_operation AS o ON o.task = t.id
  JOIN dm_node AS n ON n.id = o.node_id
  LEFT JOIN dm_nodeData AS nd ON nd.pk = n.nodeData
  JOIN dm_devices AS d ON j.device = d.devicePK
 WHERE
       d.phoneNumber = '30116718' AND
       #j.created > '2010-02-05' AND
       #j.jobID in ('8EBB50BF6AB97B3008770FACCE244678') AND
       #o.statusCode in (101) AND
       1=1
ORDER BY j.created, j.jobID, t.id, o.operationID;
```

The following table shows a possible result of the above SQL query. The table contains an example of a bootstrap job. The device has returned **`200 (OK)`** for all operations with a target like **`./SCM/`**, and will be identified as an S60 device.

The other targets do not exist on an S60 device, and thus the status **`404 (NOT_FOUND)`** is returned.

| job_ created | jobID | job Status | taskID | task_State | operation_ command | operation_ status | target | format | type | data | device PK |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2010-06-28 T13:48:21 | D17041E7CF3A2D9E5CC7E52B4A EE70C6 | Closed | 180 | NotSupported | Get | 404 | ./Com.SonyEricsson/Content/ JavaApplications | NODE | {null} | {null} | 30 |
| 2010-06-28 T13:48:21 | D17041E7CF3A2D9E5CC7E52B4A EE70C6 | Closed | 181 | Supported | Get | 200 | ./SCM/Inventory/Delivered | NODE | {null} | {null} | 30 |
| 2010-06-28 T13:48:21 | D17041E7CF3A2D9E5CC7E52B4A EE70C6 | Closed | 181 | Supported | Get | 200 | ./SCM/Download | NODE | {null} | {null} | 30 |
| 2010-06-28 T13:48:21 | D17041E7CF3A2D9E5CC7E52B4A EE70C6 | Closed | 181 | Supported | Get | 200 | ./SCM/Inventory/Deployed | NODE | {null} | {null} | 30 |
| 2010-06-28 T13:48:21 | D17041E7CF3A2D9E5CC7E52B4A EE70C6 | Closed | 181 | Supported | Add | 200 | ./SCM/Inventory/Delivered/d eleteMe | NODE | {null} | {null} | 30 |
| 2010-06-28 T13:48:21 | D17041E7CF3A2D9E5CC7E52B4A EE70C6 | Closed | 182 | NotSupported | Get | 404 | ./Software/Inventory/Native | NODE | {null} | {null} | 30 |
| 2010-06-28 T13:48:21 | D17041E7CF3A2D9E5CC7E52B4A EE70C6 | Closed | 183 | NotSupported | Get | 404 | ./Vendor/MSFT/SwMgmt/Dow nload | {null} | {null} | {null} | 30 |
| 2010-06-28 T13:48:21 | D17041E7CF3A2D9E5CC7E52B4A EE70C6 | Closed | 184 | Supported | Get | 200 | ./SCM/Inventory/Deployed | NODE | {null} | {null} | 30 |
| 2010-06-28 T13:48:21 | D17041E7CF3A2D9E5CC7E52B4A EE70C6 | Closed | 185 | Supported | Delete | 200 | ./SCM/Inventory/Delivered/d eleteMe | NODE | {null} | {null} | 30 |

For a list of the meaning of the status codes in the **`operation_status`** column, see “Status codes” at the end of this document.

## Enabling the DM XML log

All messages sent back and forth between the server and the device are in XML format, and logged in the class **`LogUtil`**. By enabling this class, you get a separate log where all XML messages are written. This is done in the **`log4j`** configuration files:

```
<appender name="DM" class="org.jboss.logging.appender.DailyRollingFileAppender">
    <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler" />
    <param name="File" value="${jboss.server.home.dir}/log/dm_server.log" />
    <param name="Append" value="true" />
    <param name="DatePattern" value="'.'yyyy-MM-dd" />
    <param name="BufferedIO" value="false"/>
    <layout class="org.apache.log4j.PatternLayout">
        <param name="ConversionPattern" value="%d %-5p [%c] %m%n" />
    </layout>
</appender>

<category name="dme.dm.util.LogUtil" additivity="false">
    <priority value="DEBUG" />
    <appender-ref ref="DM" />
</category>
```

For more information about log4j, see the log4j or JBoss documentation, or the **Changing log4j log level** technote at the Excitor Partner site.

## Reading the XML log

The XML is very verbose, and reading and understanding it is no easy matter. Knowledge of the OMA DM protocol would be an advantage. Here are a few hints:

### Who is sending, who is receiving

Here the device is sending to the server.

```
<Target>
    <LocURI>http://172.16.15.230:8080/dm/DMServlet?bid=
    EF8996E823334BB3BDEF8BCC5A265655&amp;phone=30116718</LocURI>
</Target>
<Source>
    <LocURI>IMEI:353261014679584</LocURI>
    <LocName>DM32</LocName>
</Source>
```

### Keeping track of messages

**`CmdID`**, **`MsgRef`** and **`CmdRef`** are sequential numbers. The server and client use them to refer to something the other part sent.

## Status codes

This is the complete list of possible status codes reported by a OMA DM client. For a more detailed description, see: SyncML Response Status Codes.

101, IN_PROGRESS

200, OK

201, ITEM_ADDED

202, FOR_PROCE

203, AUTHORITATIVE_RESPO

204, NO_CONTENT

205, RESET_CONTENT

206, PARTIAL_CONTENT

207, CONFLICT_RESOLVED_WITH_MERGE

208, CONFLICT_RESOLVED_WITH_CLIENTS_COMMAND_WINNING

209, CONFLICT_RESOLVED_WITH_DUPLICATE

210, DELETE_WITHOUT_ARCHIVE

211, ITEM_NOT_DELETED

212, AUTHENTICATION_ACCEPTED

213, CHUNKED_ITEM_ACCEPTED

214, OPERATION_CANCELLED

215, NOT_EXECUTED

216, ROLL_BACK_OK

300, MULTIPLE_CHOICES

301, MOVED_PERMANENTLY

302, FOUND

303, SEE_ANOTHER_URI

304, NOT_MODIFIED

305, USE_PROXY

400, BAD_REQUEST

401, INVALID_CREDENTIALS

402, PAYMENT_REQUIRED

403, FORBIDDEN

404, NOT_FOUND

405, COMMAND_NOT_ALLOWED

406, OPTIONAL_FEATURE_NOT_SUPPORTED

407, MISSING_CREDENTIALS

408, REQUEST_TIMEOUT

409, CONFLICT

410, GONE

411, SIZE_REQUIRED

412, INCOMPLETE_COMMAND

413, REQUESTED_ENTITY_TOO_LARGE

414, URI_TOO_LONG

415, UNSUPPORTED_MEDIA_TYPE_OR_FORMAT

416, REQUESTED_SIZE_TOO_BIG
417, RETRY_LATER
418, ALREADY_EXISTS
419, CONFLICT_RESOLVED_WITH_SERVER_DATA
420, DEVICE_FULL
421, UNKNOWN_SEARCH_GRAMMAR
422, BAD_CGI_SCRIPT
423, SOFT_DELETE_CONFLICT
424, SIZE_MISMATCH
425, PERMISSION_DENIED_ACL
426, PARTIAL_ITEM_NOT_ACCEPTED
427, ITEM_NOT_EMPTY
428, MOVE_FAILED
500, COMMAND_FAILED
501, COMMAND_NOT_IMPLEMENTED
502, BAD_GATEWAY
503, SERVICE_UNAVAILABLE
504, GATEWAY_TIMEOUT
505, DTD_VERSION_NOT_SUPPORTED
506, PROCESSING_ERROR
507, ATOMIC_FAILED
508, REFRESH_REQUIRED
509, RESERVED_FOR_FUTURE_USE_1
510, DATASTORE_FAILURE
511, SERVER_FAILURE
512, SYNCHRONIZATION_FAILED
513, PROTOCOL_VERSION_NOT_SUPPORTED
514, OPERATION_CANCELLED_REPEAT
516, ATOMIC_ROLLBACK_FAILED
517, ATOMIC_RESPONSE_TOO_LARGE