Software Management Client

Integrator’s Guide

Solution Version 10.2.1

Doc. Rev. 1.0.0

redbend

Table of Contents

1 **Introduction** **7**
1.1 This Document 7
1.2 Related Documentation 7
1.3 Support 8
1.4 Quick Start 8
2 **Overview** **9**
2.1 Download and Installation Overview 9
3 **Delivery Package Structure** **10**
3.1 Android Delivery Package Structure 10
4 **Installing and Rebuilding the Software Management Client** **15**
4.1 Requirements 15
4.2 Installing the Software Management Client 15
4.3 Integrating the Software Management Client with the Update Agent 16
4.3.1 Replacing the Android OSP (Full Client) 16
4.3.2 Integrating with the Android OSP (Remote Drop-In Client) 20
4.3.3 Integrating with Android OSP (Local Drop-In Client) 21
4.4 Modifying and Rebuilding the Software Management Client 22
4.4.1 Configuring the UI 22
4.4.2 Editing the Source Code Using Eclipse 23
4.4.3 Rebuilding and Signing the Client using Eclipse IDE (Kepler) 23
4.5 Registering for Google Cloud Messaging 25
4.6 Intents for Android Integration 26
4.6.1 Intents from the Device to the Software Management Client 26
4.6.2 Intents from the Software Management Client to the Device 26
4.7 Device Administrator Permissions 27
4.8 Supporting Delta Updates 27
5 **Configuring the Software Management Client** **29**
5.1 Configuring the DM Tree 29
5.1.1 Configuring the Client to Download Segments in Time Windows 29
5.2 Configuring the Software Management Client 30
5.3 LAWMO Configuration 34
5.3.1 Lock 34
5.3.2 Unlock 35
6 **Configuring the Device Integration Layer** **37**
6.1 DIL Architecture Overview 37
6.2 DIL Operation Overview 38
6.2.1 DIL Flow Overview 38
6.3 Events 38

6.3.1 Event Types 38
6.3.2 Event Structure 39
6.3.3 Event Streamer 39
6.3.4 Required Events 39
6.4 Android Implementation Overview 39
6.4.1 Primary vs Secondary Users 39
6.4.2 DIL Class Overview 40
6.4.3 Event Streamer Overview 42
6.4.4 Initializing and Terminating the Business Logic Layer 42
6.4.5 Event Streamer Initialization 43
6.4.6 Sending and Receiving Events 43
6.5 Generic Installers 45
6.5.1 Description 45
6.5.2 Generic Installer Installation Flow 46
6.5.3 Generic Installer Sync Inventory Flow 46
6.5.4 Generic Installer Example 48
6.6 Configuring Automatic Self-Registration 50
6.6.1 Android Self-Registration 50
6.7 Dynamic Proxy Configuration 50
6.8 Handling Large DPs 50
6.9 Authenticating Events 52
6.10 Configuring Silent Update and Installation 52
6.10.1 Android Example 53
6.11 Configuring the Rule Engine 53
6.11.1 Checking Rules in the Rule Engine Flow 54
6.11.2 Internally Managed Rules 54
6.11.3 Rule Engine File Format 55

7 Software Management DM Client Porting Layer Functions 58
7.1 Native Porting Layer Functions 58
7.1.1 Device Information Functions 58
7.1.2 File Search Functions 58
7.1.3 Event Authentication Functions 59
7.1.4 Deployment Package Wrapper Function 61
7.1.5 DP Access Functions 61
7.1.6 Self-Update Function 61
7.1.7 Generic Installer Functions 61
7.2 Java Porting Layer Functions 62
7.2.1 iplGetAutoSelfRegDomainInfo 62
7.2.2 getDevModel 63
7.2.3 getManufacturer 63
7.2.4 getFwVersion 63

7.2.5 getDeviceId 64
7.2.6 getUserAgent 64
7.2.7 getNextComponent 64
7.2.8 getComponentAttribute 65
7.2.9 getDefaultValue 65
7.3 Update Agent Porting Layer Functions 66
7.4 Configuration Parameters in Tree.xml 67
7.5 Other Configuration Parameters 76

Appendix A Reference DIL 79
A.1 Files Created by the Reference DIL 79
A.2 Running the Command Line Reference DIL 79

Appendix B Flows 85
B.1 Overview Flow 85
B.2 DM Session Flow 86
B.3 Download DD Session Flow 88
B.4 Download DP Session Flow 90
B.5 Installation Flow 91
B.6 Update Agent Handoff Flow 92
B.7 DP Flow 93
B.8 Boot Flow 95

Appendix C Session Tracking 97
C.1 Status Notification 97
C.2 Statuses and Details 99
C.3 Typical Generic Alert Data Structure 104

Appendix D Get Client Logs 105

Appendix E Software Management Client as a Library 105
E.1 Copying the DM Tree 105
E.2 Copying AndroidManifest.xml Items 105
E.3 Managing Events Outside the Software Management Client 105
E.4 Software Management Client as a Library Reference Application 106

Appendix F Authenticating a Self-Certified Server 110

Appendix G Verifying DP Authenticity 111

Appendix H Installer Types 112

Appendix I Features to Implement When Using the HTML5 DIL 114
I.1 Move to Background 114
I.2 Background State Persistence 114
I.3 Move to Foreground 114
I.4 Background Notification Alerts 115
I.5 Status Change Events 115

I.6 Server Initiated Sessions 115
I.7 Battery Level Check........ 116
I.8 Porting Layer Callbacks 116
I.9 Lock and Wipe (LAWMO)........ 116

Appendix J Terms and Abbreviations 117

Table of Tables

Table 5-1: Configuration Parameters 31
Table 7-1: Configuration Parameter Descriptions........ 76
Table H-2: Installer Types 112
Table I-1: State Change Events........ 115

Table of Figures

Figure 4-1: Device Administrator Dialog Box 27
Figure 6-1: Extended Framework Architecture 37
Figure 6-2: DIL Class Relationships 41
Figure 6-3: Generic Installer Installation Flow 46
Figure 6-4: Generic Installer Sync Inventory Flow........ 47
Figure B-1: Flow Overview 85
Figure B-2: DM Session Flow 86
Figure B-3: Download Session Flow, Part 1 88
Figure B-4: Download Session Flow, Part 2 90
Figure B-5: Installation Flow........ 91
Figure B-6: Update Agent Handoff Flow 92
Figure B-7: Large DP Download Flow 94
Figure B-8: DP Installation Flow 95
Figure B-9: Boot Flow 95

1 Introduction

Redbend's Software Management Client is a set of software components on a device that installs, removes, and updates software components on a device. The Client receives instructions from Redbend's Software Management Server. The Client can be downloaded by the end-user.

The Software Management Client delivery package includes:

- The Software Management Client, Redbend's OMA DM Protocol Engine Client
- The Redbend Update Agent, which performs system-level installations on the device. You can replace Redbend's Update Agent with your own. The Software Management Client does not contain or require an update agent.

The Client must be downloaded by the end-user or copied to the file system.

Redbend provides a complete Software Management Client for Android. You can install the Android Client as an APK, or build the Client as a library to integrate into your own application.

1.1 This Document

This document presents how to install and configure the Software Management Client on the device. If you want to customize or create your own version of the DIL, this document describes how to pass and receive events to and from the Software Management Client's business logic, as well as the Porting Layer functions that you can change.

This document also presents how to integrate Redbend's Update Agent with the DM Client. For complete information about installing and configuring the Update Agent, see the *Update Agent Integrator's Guide*.

This document is intended for engineers and support personnel who will install and/or configure the Software Management Client.

1.2 Related Documentation

The Software Management Solution includes the following documents:

Category	Document Name
Getting Started	• *Software Management Solution Description* • *Software Management Server Hardware and Software System Configuration Alternatives*
Installation	• *Software Management Server Installation and Operation Guide*
Integration	• *Software Management Client Integrator's Guide* • *Software Management Client Quick Integrator's Guide* • *Software Management Server Integrator's Guide* • *Software Management Server API Reference*
User Interfaces	• *Software Management Server Administrator User Guide* • *Software Management Server CLI Documentation (CMS_DOCs)*

Additional relevant documentation includes:

- *Update Agent Integrator’s Guide*
- If you are using your own update agent, and your update agent uses Redbend’s Update Installer, see the *Update Generator and Installer Integrator’s Guide*.
- *OMA-DM Protocol Engine Framework Integrator’s Guide*
- *OMA-DM Protocol Engine Extended Framework Integrator’s Guide*
- *OMA-DM Protocol Engine API Documentation*

1.3 Support

If you have a question or require further assistance, contact Redbend’s customer support at support@redbend.com.

1.4 Quick Start

To install the Software Management Client as delivered onto an Android device, with the Update Agent installed alongside the Android Open Source Project update mechanism, see the *Software Management Client Quick Integrator’s Guide*.

2 Overview

The Software Management Client performs operations in response to requests from the Software Management Server. One common use-case is when the Server asks the Client to download and install a software installation or update. Another is when the Client asks the Server if there is any new software to install. The Server can also ask the Client to remove software, or to lock, unlock, or wipe data from the device.

Updating firmware requires a device reboot.

2.1 Download and Installation Overview

The Software Management Client can initiate an update in response to three triggers:

- The end-user checks for updates. This is a *user-initiated* process.
- The device checks for updates periodically, such as every 24 hours. This is a *device-initiated* process, also called *polling*.
- The Software Management Server sends a message (such as a WAP push) to the device indicating that there is an update. This is a *server-initiated* process.

Some updates are marked as *silent*. For silent updates, the update proceeds without any indication to the end-user. At all points in the process where end-user input is normally expected, the Client must simply return a response (such as “OK”) on behalf of the end-user.

Some updates are marked as *critical*. For critical updates, the Client does not allow the end-user to cancel the update or postpone it indefinitely.

The download and update process works as follows:

1 The Client checks for updates, either in response to a request from the user or the Software Management Server, or at the specified time since the last check for updates.

2 If there are no updates, the Server indicates this and the process ends. If there are updates, the Client is instructed to get a *Download Descriptor* from a certain location. The Download Descriptor contains information about the updates.

3 The Client gets the Download Descriptor, which contains the names, vendors, and sizes of the updates, the URLs from which to download the updates, and an optional URL for release notes with additional information about the updates (“What’s new”, etc.). The Client can get the release notes from the URL in either HTML or XML format.

4 The end-user can reject, accept, or postpone the download. Not all of these options may be available, and the end-user can only postpone the download a limited number of times.

5 The Client downloads a *Deployment Package* (DP) containing the updates into a directory. A DP may contain multiple updates.

6 Like the download, the end-user can reject, accept, or postpone the installations. Not all of these options may be available, and the end-user can only postpone the installations a limited number of times.

7 The Client hands off each installation or update to a specific installation method depending on its installer type (see Authenticating a Self-Certified Server).

8 Redbend provides functions that handle the installation of various types of software, including Android applications and ECU software.

9 Redbend hands off firmware updates to an update agent. The Client continues with the installation after the update agent restarts the device.

10 After success or failure, the Client sends a report back to the Server with the results.

The Client handles interruptions in a failsafe manner. Interrupts can occur when:

- The device leaves the network range
- The system is defined to limit the total number of concurrent Client download sessions and the limit is reached.
- Note: This feature is not supported when implementing a CDN architecture.
- The client is configured to limit the segment size that can be downloaded within a specific time window, and the segment size limit is reached.

Generally, after the interruption is resolved (for example, when the network connectivity is restored) the process continues from where it left off.

3 Delivery Package Structure

The Software Management Client is delivered as a zip file; unzip the file to an empty directory. The delivery package contains the following directories and files.

3.1 Android Delivery Package Structure

Structure	Description
com.redbend.client.apk	Pre-built sample Software Management Client application, built with the release option If you are including the Software Management Client as a library in your own application, ignore this file
`Docs/doxygen/html/index.html`	Documentation on all header files This includes a list of events that must be sent by the DIL to the business logic, or handled by the DIL when received from the business logic
`setup/`	
com.redbend.client	An empty file that indicates to the Software Management Server that the Software Management Client is installed
rb_ua	Update Agent executable
`maguro/`	Update Agent configuration files for maguro (example)

Structure	Description
rb_recovery.fstab	Example Update Agent configuration file that defines partitions
rb_ua.conf	Example Update Agent configuration file with configuration parameters
`<toolchain>/<target-platform>/`	Platform libraries
`rls/`	
libbl.a	Business logic library
libdmacoapp.a	General functions for the DIL
libdma_jni.a	JNI functions used by the DIL
libdmammi.a	MMI functions
libsmm.so	A compiled shared object library compromising all the static libraries in this directory
libswmcadapter.a	Interface functions between the business logic and the installers
libswmcinstallers.a	Installers
libvdmcomm.a	Communication functions
libvdmdescmo.a	internal
libvdmengine.a	Engine functions
libvdmfumo.a	FUMO operations functions
libvdmipc.a	IPC functions
libvdmlawmo.a	LAWMO operations functions
libvdmplat.a	Platform-specific Porting Layer functions
libvdmplclient.a	Client Porting Layer functions
libvdmscinv.a	SCOTA inventory operations functions
libvdmscomo.a	SCOTA general operations functions
libvdmsmm.a	State Machine Manager (SMM) functions

Structure	Description
libvdmsmmpl.a	SMM Porting Layer functions
libvdmswmcpldevice.a	Device Porting Layer functions
libvdmswmcpldir.a	Directory Porting Layer functions
libvdmswmcpldp.a	DP Porting Layer functions
libvdmswmcplecu.a	ECU Installer Porting Layer functions
libvdmswmcplgeneric.a	Generic Installer Porting Layer functions
libvdmswmcplselfupgrade.a	Self Upgrade Porting Later functions
libvdmswmcplua.a	Handoff Porting Layer functions
libvdmutil.a	Utility functions
`rls_dbginfo/`	The same files as the `rls/` directory compiled with the `rls_dbginfo` (show debug log prints) option
`common/`	Common infrastructure classes used by the application, built as library For details, see DIL Architecture Overview
`redbend/`	Main Android application project directory (including source files of the DIL) For details, see DIL Architecture Overview
`assets/`	
nias.txt	Sample NIAs to simulate WAP push using SMS If you send the strings in this file to a device using SMS, you get the listed NIAs
`files/`	Files that are copied to the application directory
tree.xml	The DM Tree
`html5/`	HTML5 UI
`css/`	HTML5 CSS files
`img/`	HTML5 picture files

Structure	Description
`js/`	HTML5 JavaScript files
`doT/`	Template engine
`i18n/`	Internationalized screen texts
`jquery/`	External JS library
`platform/`	Platform-specific Porting Layer files For details, see Configuring the UI
`android/`	Android Porting Layer files
`stub/`	
`common/`	Porting Layer files for any platform
`redbend/`	JS library files
`simulator/RedBendDmcApp.bar`	QNX CAR 2 DIL executable for a simulator
`strings/`	HTML5 strings for all languages
index.html	HTML5 application file All pages are implemented here
`libs/`	Internal JARs of the Java project files for the application
`jni/`	JNI makefiles
`gota_config/`	Scripts and configurations used to generate the file needed for Android Open Source Project OTA (used when Redbend's Update Agent coexists with the Android Open Source Project update mechanism)
signapk.jar	Java utility used to sign the OTA zip file used for drop-in integration For details, see Installing and Rebuilding the Software Management Client
`res/`	Android resource files
`src/`	DIL Java source files
`sdk/`	Engine Porting Layer files

Structure	Description
`import/`	Porting Layer header files that can be modified
`include/`	Porting Layer header files that don’t require modification
`swm_common/`	This directory contains an extension of the content in the `redbend` directory—it is extracted from the `redbend` directory and is for internal use only For details, see DIL Architecture Overview
`swm_container_demo/`	(Only applicable to the Software Management Client as a library delivery) A reference application that demonstrates how to include the Software Management Client as a library in an Android application
`swmc/`	Software Management Client Porting Layer files
`import/`	Porting Layer header files that can be modified
`src/`	Porting Layer source code

4 Installing and Rebuilding the Software Management Client

A pre-built sample Software Management Client Android app for pre-installation is provided in the delivery package. This chapter presents how to install or rebuild the Software Management Client Android app, either as a pre-installable client or as a downloadable client. Basic configuration includes optionally modifying the UI elements, adding support for the Android Open Source Project update mechanism (if you are not replacing it with Redbend's Update Agent), and setting a few configuration parameters.

The downloadable client works only with user applications; it cannot manage embedded applications or firmware. To manage embedded applications and firmware, use the pre-installed client.

If you want to modify the DIL, or create your own, the delivery package includes Java sources that you can use to build your own Android app. Before making any changes, refer to the following chapters in this document for important design requirements.

NOTE: If you intend to include the Software Management Client as a library in your own Android application, skip the marked sections in this chapter and follow the instructions in Software Management Client as a Library . The remaining sections in this chapter are still applicable.

4.1 Requirements

The Android app must be built with Android SDK revision 21 (Jelly Bean MR1.4.2) and is supported for Android version 4.2 or later. Before installing or modifying the Software Management Client, you must first install Android build tools revision 22 or later. For the complete list of SDK system requirements, see *System Requirements* at https://developer.android.com/sdk/index.html.

To see the Android permissions required by the Software Management Client, refer to **AndroidManifest.xml** (in the delivery package).

4.2 Installing the Software Management Client

For a downloadable Client, provide the end-user a location (such as a URL) from which to download. No installation is required.

NOTE: If you plan to include the Software Management Client as a library in your own Android application, skip this section.

The delivery package includes a sample pre-built Software Management Client Android app that can be used as a pre-installed client for managing embedded applications and firmware.

For information about building a new version of the Client, see Modifying and Rebuilding the Software Management Client.

To pre-install the Client:

1 Copy the Android app to `/system/app` on the device:

```
adb push com.redbend.client.apk /system/app
```

- For Android KitKat, copy the Android app to `/system/priv-app`.
- For Android Lollipop and higher, copy the Android app to `/system/priv-app/Redbend/`.

2 Copy **libsmm.so** to `/system/lib` on the device:

```
adb push android_ndk46/android_native_R8b_jb/rls/libsmm.so
/system/lib
```

NOTE: Use the library in `rls_dbginfo/` if you would like the Client to produce log files.

3 Reboot the device.

```
adb reboot
```

You can test the installation by running a campaign to get information about the device using the Software Management Server. For more information, see the *Software Management Server Administrator's Guide*.

4.3 Integrating the Software Management Client with the Update Agent

For complete information about integrating and configuring Redbend's Update Agent, see the *Update Agent Integrator's Guide*

If the device will not be updating firmware or embedded applications – for example, if you are creating a downloadable client – skip this section.

There are two different procedures for integrating the Software Management Client with the Update Agent:

- **Replace the recovery image of your device**. In this procedure (the *Full Client*), you replace the Android Open Source Project update mechanism with Redbend's Update Agent. The integration requires that you modify the recovery and boot images to include an integrated init script.
- **Do not replace the recovery image**, instead install Redbend's Update Agent alongside the Android Open Source Project update mechanism (the *Drop-In Client*). Updates sent to the device are handed off to the Android Open Source Project update mechanism, which then hands them back to Redbend's Update Agent. There are two methods to implement this:
  - **Local**: Place an OTA zip file in the device file system, modify configuration files, and copy files to the device.
  - **Remote**: Upload the firmware delta to the Software Management Server after embedding the delta into an OTA zip file. The Client downloads the DP, extracts the embedded firmware delta, and hands it off to the Android Open Source Project update mechanism.

4.3.1 Replacing the Android OSP (Full Client)

Perform the following steps if you replace the Android Open Source Project update mechanism with Redbend's Update Agent.

The Full Client integration requires changes to the **recovery.img**, where most of the update is done and to the **boot.img**, where the update of **recovery.img** is performed. The **recovery.img** must be updated from the main system and not from the recovery system to ensure successful completion of the update if there is a power failure.)

The full integration requires including the init script in the image file.

Before integrating the Update Agent, make sure that you have the following:

- A Linux/QNX build environment containing relevant Android source code.
  See http://source.android.com/source/initializing.html for a complete guide on how to set the build environment.
- Ability to manipulate main system and recovery system images.

Note: Redbend Full Integration:

- Is primarily for non-Android systems
- The Update Agent is part of the FOTA in self-updates
- Updates are preformed via the Redbend Update Mechanism
- SYSAPK is supported with encrypted user data (no dm-verity)

4.3.1.1 Editing the DM Tree

In the DM Tree, make sure that **./DevDetail/Ext/RedBend/RecoveryType** is set to `RB`. If you change the tree, rebuild the Client, as described in Rebuilding and Signing the Client using Eclipse IDE (Kepler)

4.3.1.2 Configuring the Software Management Client

NOTE: On encrypted devices, the data partition cannot be accessed by the recovery kernel. To support encryption, use the cache partition. If the cache partition is not large enough for the DP, you must use the data partition (and encryption is then not supported).

Also, on some systems running SELinux, SELinux blocks access to the data partition from recovery.

Set the following configuration parameters in the Software Management Client configuration file, **dma_config.txt** (see Configuring the Software Management Client):

```
gota_config_path=/cache
handoff_dir=/cache
dp_path_file=/cache/dp
ua_result_file=/cache/result
dp_full_path=<designated-directory>
```

Where:

- `<designated-directory>` is either: `/cache`
  or:
  `/data/data/com.redbend.client/files`

4.3.1.3 Configuring the Update Agent

NOTE: When modifying the Software Management and Update Agent configuration files (**dma_config.txt** and **rb_ua.conf**), make sure that the following parameter values in **dma_config.txt** are identical to the listed parameter values in **rb_ua.conf**:

dma_config.txt	rb_ua.conf
dp_path_file	delta_location
ua_result_file	result_file

Create two Update Agent configuration files, both named **rb_ua.conf**, one for the boot image and one for the recovery image. Set the following configuration parameters:

```
work_dir=/cache/workdir
log_path=debug:/cache/logs/ua_log
delta_location=/cache/dp
update_dir=/system
result_file=/cache/result
update_flavor=std
fw_installer_type=9
fs_installer_types=11,250,251,252,253,254
part_list_fstab_format=1
```

For the boot image configuration file add:

```
in_recovery_kernel=0
```

For the recovery image configuration file add:

```
in_recovery_kernel=1
```

For non-dm-verity devices, create only one configuration file and leave out the line containing `in_recovery_kernel`.

For any Android device, configure the partition list. For example, for a Nexus 5 boot image add:

```
partitions_list=/fstab.hammerhead
```

For dm-verity or SELinux-enforced devices, add:

```
exec_path=/tmp/rb/
```

4.3.1.4 Modifying the Device Recovery System

The main integration task is to start the Update Agent from the recovery system. Do this by modifying the device's *recovery system image init script*, **init.rc**. The **init.rc** syntax is described in http://www.kandroid.org/online-pdk/guide/bring_up.html; see *Android Init Language*.

To modify the recovery system image:

1 For non-image updates, in the **init.rc** script file in the `on boot` section, mount the **system** and **userdata** partitions.

For example:

```
mount ext4 /dev/block/platform/omap/omap_hsmmc.0/by-name/system
/system wait ro
```

2 In the **init.rc** script file, replace the recovery service with **rb_ua**.

Remove this:

```
service recovery /sbin/recovery
```

Add this:

```
service rb_ua <Update Agent executable> <parameter list>
```

Where:

- `<Update Agent executable>` is the full path to the Update Agent executable.
- `<parameter list>` is one or more parameters to be used by the Update Agent.

3 In the Update Agent configuration file, add the following line:

```
in_recovery_kernel=1
```

For SELinux, also add the following line:

```
exec_path=/tmp/rb/
```

4.3.1.5 Modifying the Device Main System

If **recovery.img** must be updated, modify the main system image init script, **init.rc**. **recovery.img** is updated after the device is updated and started into the main system.

To modify the main system image:

1 In the **init.rc** script file, replace the `install_recovery` service with **rb_ua**.

Delete the lines that define the `install_recovery` service:

Add the following lines to add **rb_ua** as a service:

```
service rb_ua <Update_Agent_executable> <parameter_list>
   class main
   seclabel u:r:init:s0
   oneshot
```

Where:

- `<Update_Agent_executable>` is the full path to the Update Agent executable.
- `<parameter_list>` is one or more parameters to be used by the Update Agent.

For SELinux-disabled devices, the line defining `seclabel` is not required.

2 In the configuration file for running the Update Agent, add the following line:

```
in_recovery_kernel=0
```

NOTE: This may be the same configuration file used in the recovery system. `in_recovery_kernel=0` can be specified in the parameter list.

4.3.1.6 Copying the Update Agent and its Configuration Files to the Device

This section lists the steps required to copy the update agent and configuration files to the device.

For dm-verity or SELinux-enforced devices:

1 Unpack the main and recovery images.

2 Copy the recovery configuration file to `<recovery-image-ramdisk>/etc/rb_ua.conf`

3. Copy the boot configuration file to `<main-image-ramdisk>/rb_ua.conf`
4. Copy `rb_ua` to `<image-ramdisk>/sbin/` in each image.
5. If a `tmp` directory does not exist, create it in each image:

   ```
   mkdir /<image-ramdisk>/tmp
   ```
6. Create an SELinux policy for the boot image, as described in Modifying the Device Main System, copy it to the boot ramdisk, and name it `sepolicy`:

   ```
   cp <boot-sepolicy> <boot-ramdisk>/sepolicy
   ```

   For more information, see the *Update Agent Integrator's Guide*.
7. Repack the image.

4.3.2 Integrating with the Android OSP (Remote Drop-In Client)

Perform the steps outlined in the subsections below if you send Redbend's Update Agent from the Software Management Server to work alongside the Android Open Source Project update mechanism.

Note: The Drop in Client for Android supports the integration modes listed here. In addition, update package size is increased by 300 - 600 KB.

- DM-verity and encrypted systems
- The Update Agent is transmitted from the Server during each update
- Updates are performed via the Android OTA update mechanism
- The Increases the update package size by 300-600KB

4.3.2.1 Configuring the Software Management Client

Set the following configuration parameters in the Software Management Client configuration file **dma_config.txt** (see Configuring the Software Management Client):

```
dp_split_support=True
dp_full_path=/data/data/com.redbend.client/files
handoff_dir=/cache
dp_path_file=/cache/dp
ua_result_file=/cache/result
gota_config_path=/cache
```

4.3.2.1.1 Android L

For Android L, the location of the file which will contain the dp path (the **dp_path_file**) *must be aligned with the* **rb_ua.conf**.

The DP itself (the **dp_full_path**) must be located in a *wr permission folder*, (i.e., `/cache/Redbend`), *not* in `/data/data.com.redbend.client/files`.

Specifically, the following line may NOT be used:

```
dp_full_path=/data/data/com.redbend.client/files
```

Instead use:

```
dp_full_path=/cache/dp
```

Example:
dp_path_file=/cache/dp

```
dp_full_path=/cache/__SCOMO_DP
```

Additional Steps

- Edit the DM Tree: Make sure that **./DevDetail/Ext/RedBend/RecoveryType** is set to `GOTA_REMOTE`.

4.3.3 Integrating with Android OSP (Local Drop-In Client)

Perform the following steps if you install Redbend's Update Agent alongside the Android Open Source Project update mechanism.

In this scenario, the Update Agent runs as part of the Android Open Source Project update mechanism. You must add the Update Agent and its configuration files to an Android Open Source Project OTA package.

4.3.3.1 Configuring the Software Management Client

Set the following configuration parameters in the Software Management Client configuration file **dma_config.txt** (see Configuring the Software Management Client):

```
handoff_dir=/cache
dp_path_file=/cache/dp
ua_result_file=/cache/result
gota_config_path=/cache
dp_full_path=<designated-directory>
```

Where `<designated-directory>` is either `/cache` or `/data/data/com.redbend.client/files`

4.3.3.1.1 Android L

For Android L, the

```
dp_full_path=<designated-directory>
```

Where: `<designated-directory>` is:
`/cache only.`

4.3.3.2 Configuring the Update Agent

NOTE: When configuring the Software Management and Update configuration files (**dma_config.txt** and **rb_ua.conf**), make sure that the following parameter values in **dma_config.txt** are identical to the listed parameter values in **rb_ua.conf**:

dma_config.txt	rb_ua.conf
dp_path_file	delta_location
ua_result_file	result_file

Create an Update Agent configuration file named **rb_ua.conf**. Set the following configuration parameters:

```
work_dir=/cache/workdir
log_path=debug:/cache/logs/ualog
```

```
delta_location=/cache/dp
update_dir=/system
result_file=/cache/result
update_flavor=std
part_list_fstab_format=1
in_recovery_kernel=1
no_reboot=1
recovery_command=rb_ignore
set_boot_to_recovery=0
fw_installer_type=9
fs_installer_types=11,250,251,252,253,254
```

When using SELinux, add:

```
exec_path=/tmp/rb/
```

4.3.3.3 Additional Steps

The Software Management Client sends a signed file **rb_ota.zip** to Android Open Source Project’s update mechanism.

To perform the additional steps:

1. Create **rb_ota.zip** as described in the *Integrating with the Android Open Source Project Update Mechanism (Drop-In Client)* section of the *Update Agent Integrator’s Guide*.
2. Move the newly signed **rb_ota.zip** to `assets/files`.
3. Edit the DM Tree: Make sure that **./DevDetail/Ext/RedBend/RecoveryType** is set to `GOTA_LOCAL`.
4. Rebuild the Client, as described in Rebuilding and Signing the Client using Eclipse IDE (Kepler).

4.4 Modifying and Rebuilding the Software Management Client

The following sections present common scenarios that involve modifying the Software Management Client. After any of these modifications, you must rebuild the Client as described in Rebuilding and Signing the Client using Eclipse IDE (Kepler).

4.4.1 Configuring the UI

The DIL displays UI screens according to the DIL event name.

Message texts, icons, and screens are located in `redbend/res/`. Messages and screens are provided in English.

The UI is built using HTML5. Message texts, icons, and screens are located under `html5/`. The HTML5 codes function as a DIL, sending and receiving events to and from the business logic. All screens are displayed using **index.html** and JavaScript in **js/redbend/dil_actions.js**.

- **i18n**: Message texts are located under `strings/` in files that correspond to a language name. The default language is English, and texts are in **en.js**. For other languages, copy **en.js** to **<locale code>-<language code>.js** (for example, **zh-CH.js**) and change the texts. Open **js/redbend/dil_actions.js** to locate specific texts for specific screens.

- **Porting Layer**: HTML5 Porting Layer functions are under `js/platform/`. In **js/redbend/configuration.js** set the configuration parameter `platform` to your platform and complete the stubs in **js/platform/<platform name>**, using the code in **js/platform/android/** as a reference. Details about each HTML Porting Layer function can be found in the files in this directory.
- **Sockets**: If your platform supports web sockets, in **js/redbend/configuration.js** set the configuration parameter `use_web_socket` to `true`. Socket code is in **js/redbend/ws-client.js**. If your platform does not support web sockets, set `use_web_socket` to `false`. By default, the Android client is not configured to use web sockets.

  The Software Management Client provides a mechanism for authenticating events sent and received using sockets. For more information, see Event Authentication Functions.
- **Additional features**: The HTML5 DIL does not implement all features presented in this document. You must implement the remaining features, as described in Features to Implement When Using the HTML5 DIL.

If you change any UI elements, rebuild the Client, as described in Rebuilding and Signing the Client using Eclipse IDE (Kepler).

4.4.2 Editing the Source Code Using Eclipse

Before editing the DM Client using Eclipse, install the following:

- Eclipse
- The Eclipse plug-in for Android
- Android API level 21

NOTE: If you plan to include the Software Management Client as a library in your own Android application, skip this section.

To edit the source code using Eclipse (Kepler Version):

1 Open Eclipse, and select **File** → **New** → **Project**

  The **New Project** dialog box appears.

2 Select **Android** → **Android Project From Existing Code**, and click **Next**.

  The **New Android Project** dialog box appears.

3 Browse to the main delivery directory and click **Finish**.

4 Select the projects: `smw_common`, `common`, and `StartupActivity` and click **Finish**.

After editing the source code, rebuild the Client as described in Rebuilding and Signing the Client using Eclipse IDE (Kepler).

4.4.3 Rebuilding and Signing the Client using Eclipse IDE (Kepler)

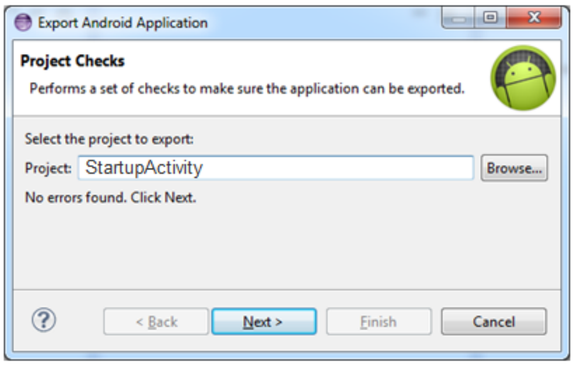
NOTE: If you plan to include the Software Management Client as a library in your own application, skip this section.

To rebuild the Software Management Client:

1 Make any of the changes described in this document and/or edit the source code using Eclipse or any other IDE.

2 Right-click the `StartupActivity` project and select **Android Tools → Export Signed Application Package**

The **Project Checks** window appears.



Export Android Application

Project Checks

Performs a set of checks to make sure the application can be exported.

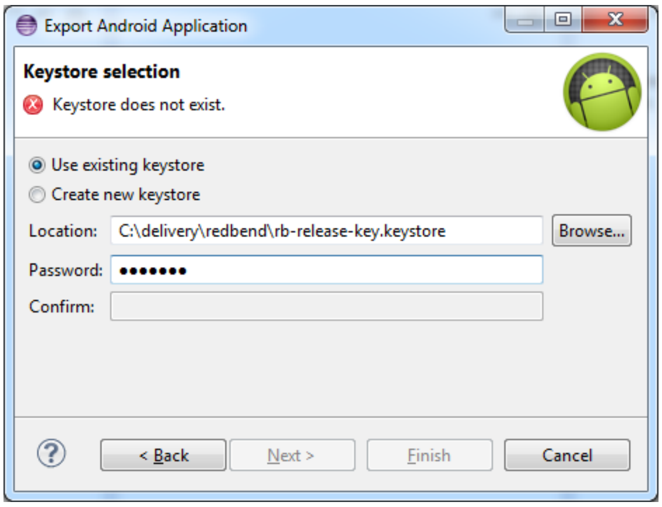
Select the project to export:

Project: StartupActivity

No errors found. Click Next.

3 Type `StartupActivity` and click **Next**.

The **Keystore selection** window appears.



Export Android Application

Keystore selection

Keystore does not exist.

(●) Use existing keystore
( ) Create new keystore

Location: C:\delivery\redbend\rb-release-key.keystore

Password: •••••••

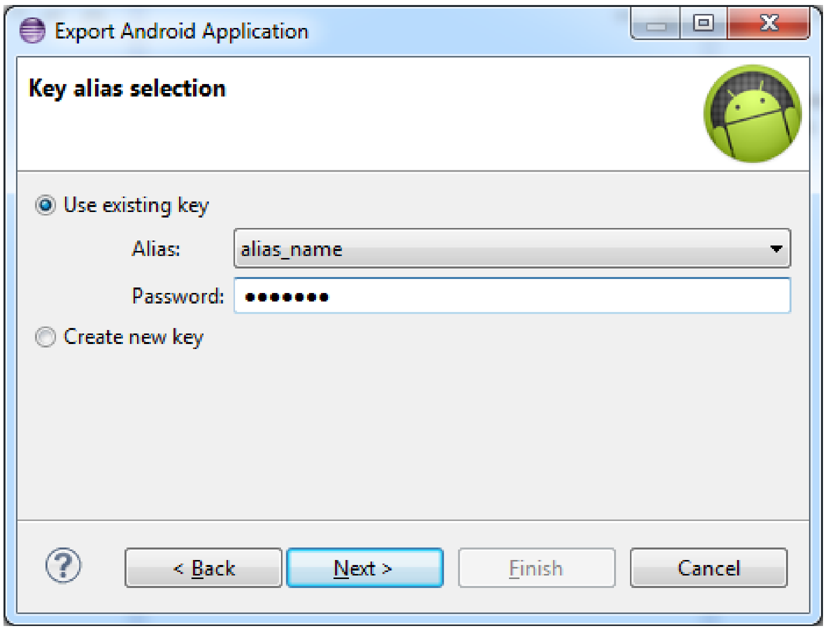
Confirm:

4 For **Location**, click **Browse**, and browse to the `redbend` directory and select the Redbend keystore **rb-release-key.keystore** or a keystore that you have generated.

5 If you are using the Redbend keystore, in **Password** type: `redbend`. Otherwise, type the keystore password.

6 Click **Next**.

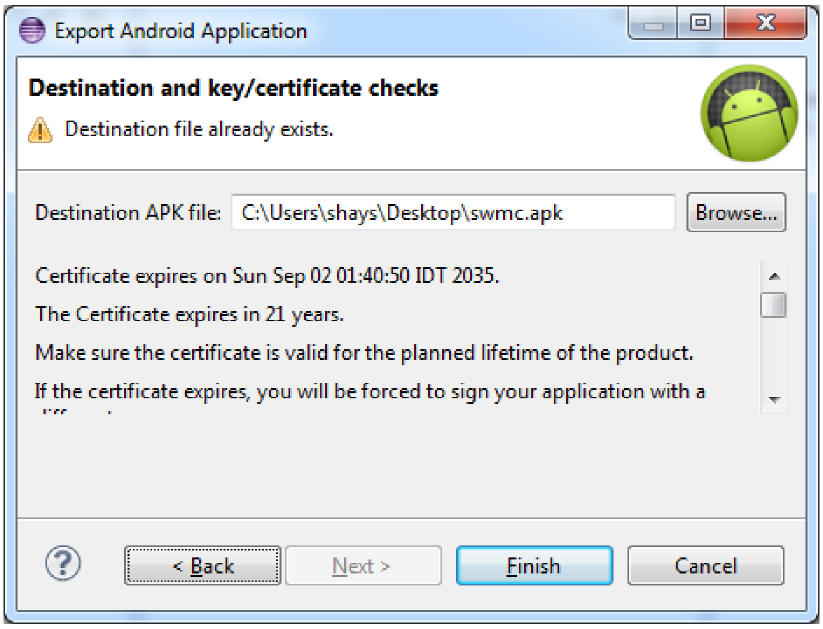
The **Key alias selection** window appears.



7 If you are using the Redbend keystore, select the `alias_name` alias. Otherwise, select the required alias from the list.

8 Type the alias password in **Password** (for Redbend, type: `redbend`) and click **Next**.

The **Destination and key/certificate checks** window appears.


Export Android Application
Destination and key/certificate checks
Browse...
< Back
Next >
Finish
Cancel

Destination file already exists.

Destination APK file: C:\Users\shays\Desktop\swmc.apk

Certificate expires on Sun Sep 02 01:40:50 IDT 2035.
The Certificate expires in 21 years.
Make sure the certificate is valid for the planned lifetime of the product.
If the certificate expires, you will be forced to sign your application with a

9 Browse to the location and name of the target Android app.

10 Click **Finish**.

The Client is rebuilt. After rebuilding, proceed with the installation as described in Installing the Software Management Client .

4.5 Registering for Google Cloud Messaging

The Software Management Server can be configured to use Google Cloud Messaging (DIL) service as a third-party gateway to send custom OMA DM notifications to devices. The Software Management Client includes support for GCM.

If the Software Management Server uses GCM, each device must also be registered with the GCM service to receive OMA DM notifications from the Software Management Server.

To register a device for GCM:

- The Software Management Client uses the default Google account to automatically register the device with the GCM service.

4.6 Intents for Android Integration

4.6.1 Intents from the Device to the Software Management Client

The Software Management Client receives the following intents to enable you to integrate the Software Management Client with Android device settings.

- `SwmClient.CHECK_FOR_UPDATES_NOW`: Check if there is a new update for the device.
- `SwmClient.ENABLE_PERIODIC_CHECK_FOR_UPDATES`: Enable periodic checking for updates.
- `SwmClient.DISABLE_PERIODIC_CHECK_FOR_UPDATES`: Disable periodic checking for updates.
- `SwmClient.CHANGE_PERIODIC_CHECK_FOR_UPDATES`: Change the interval of the periodic checking for updates. Set the new interval in hours in the event variable `DMA_VAR_SCOMO_DEVINIT_NEW_POLLING_INTERVAL`.
- `SwmClient.RB_ANALYTICS_STATE`: Enable or disable analytics. Set the new state in the event variable `enable_analytics`.

Android example code:

```
// Check for updates
Intent intent = new Intent();
intent.setAction("SwmClient.CHECK_FOR_UPDATES_NOW");
sendBroadcast(intent);

// Enable analytics (true to enable, false to disable)
Intent intent = new Intent();
intent.setAction("SwmClient.RB_ANALYTICS_STATE");
intent.putExtra("enable_analytics",true);
sendBroadcast(intent);
```

4.6.2 Intents from the Software Management Client to the Device

The Software Management Client sends broadcast intents to notify external Android components about the status of updates:

- `SwmClient.NEW_UPDATE_AVAILABLE`: Update available on the Software Management Server
- `SwmClient.UPDATE_SESSION_END`: Update finished successfully

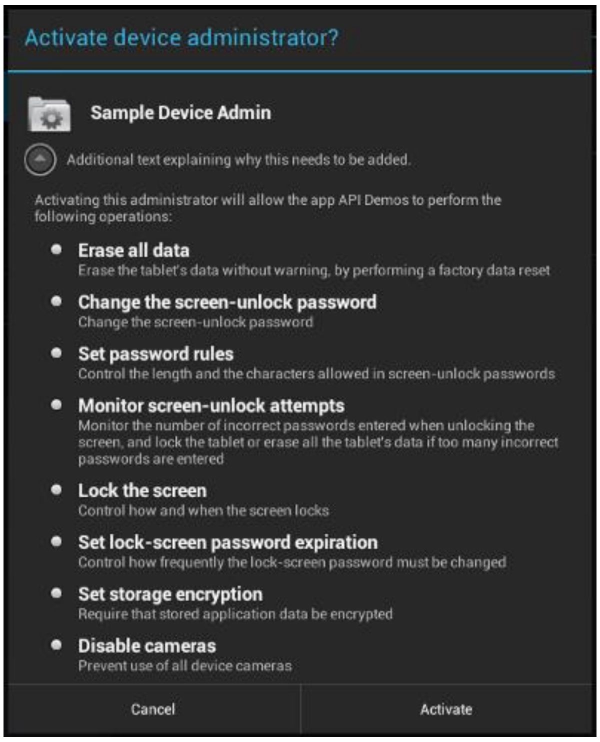
If other applications wish to catch these intents, they must implement an Android receiver and add the intents to the receiver filter.

4.7 Device Administrator Permissions

This enables the Software Management Client to do the following:

- Locking the device
- Unlocking the device
- Performing a factory reset on the device

When the Software Management Client is started during device initialization, the Software Management Client requests permission from the end-user to activate these permissions. If permission is not given or, at a future time, rescinds administrator permissions from the Client, the Client will request enabling them *each time* the end-user taps the Software Management Client icon (a Device Administrator dialog box will be displayed).



Activate device administrator?

Sample Device Admin

Additional text explaining why this needs to be added.

Activating this administrator will allow the app API Demos to perform the following operations:

- **Erase all data**
  Erase the tablet's data without warning, by performing a factory data reset
- **Change the screen-unlock password**
  Change the screen-unlock password
- **Set password rules**
  Control the length and the characters allowed in screen-unlock passwords
- **Monitor screen-unlock attempts**
  Monitor the number of incorrect passwords entered when unlocking the screen, and lock the tablet or erase all the tablet's data if too many incorrect passwords are entered
- **Lock the screen**
  Control how and when the screen locks
- **Set lock-screen password expiration**
  Control how frequently the lock-screen password must be changed
- **Set storage encryption**
  Require that stored application data be encrypted
- **Disable cameras**
  Prevent use of all device cameras

Figure 4-1: Device Administrator Dialog Box

4.8 Supporting Delta Updates

By default, the Software Management Client handles only full updates for APKs. Delta updates allow the Software Management Server to send smaller updates.

To support delta updates:

1. Create an Update Agent configuration file, **rb_ua.conf**, and place it in `redbend/assets/files`.
2. Place the Update Agent executable (**rb_ua**) in `redbend/assets/files`.
3. Set the parameter in the Software Management Client configuration file that points to the location of the Update Agent:

```
ua_exec=/data/data/com.redbend.client/files/rb_ua
```

For more information about configuration parameters, see Configuring the Software Management Client.

4 Build the DM client APK. At initialization, the DM client application copies **rb_ua.conf** and **rb_ua** to `data/data/com.redbend.client/files/`.

5 Configuring the Software Management Client

This chapter explains how to modify the DM Tree and the Software Management Client configuration file.

5.1 Configuring the DM Tree

Much of the DM functionality, including server and client authentication, is controlled using the DM Tree (**tree.xml**). For more information, see the *OMA-DM Protocol Engine Framework Integrator’s Guide*.

As an example of a change that you might make that involves changing the DM Tree, you can optimize RAM usage on the device by removing the `FUMO` branch of the tree if your client does not support FUMO or the `SCOMO` branch if your client does not support SCOMO.

You must *always* configure the following values:

- **./DMAcc/callup/AppAddr/APPSRV/Addr**: IP address of the Software Management Server
- **./DMAcc/callup/AppAddr/APPSRV/Port/Port/PortNbr**: Port number of the Software Management Server
- **./DevInfo/Ext/RedBend/DomainName**: Domain name
- **./DevInfo/Ext/RedBend/DomainPIN**: Domain PIN

Do not remove any nodes under the **./Ext/RedBend** branch.

For more information about setting the device model, vendor, and ID on Android, see Initializing and Terminating the Business Logic Layer.

If you change the DM Tree, rebuild the Client as described in Rebuilding and Signing the Client using Eclipse IDE (Kepler).

5.1.1 Configuring the Client to Download Segments in Time Windows

This feature limits the amount of data, in KBs, (SegmentSize) that the Software Management Client can download within a configurable timeframe (SegmentWindow). The segment is downloaded until the data limit is reached (SegmentSize).

If the downloaded data within the defined time frame reaches the SegmentSize limit, the download stops and will not resume until the beginning of the next time frame.

If the download is interrupted before downloading the full SegmentSize, it can be resumed within the same time frame (SegmentWindow), until the SegmentSize limit is reached.

If the time window has passed and the amount of data downloaded is less than the SegmentSize, the download operation will continue in the next time frame.

This feature is *disabled* by default.

When enabled, the feature uses the tree node parameters (SegmentWindow and SegmentSize).

For more details, see getDefaultValue.

5.2 Configuring the Software Management Client

The Software Management Client provides configuration options in a separate configuration file. Changing these configuration options does not require you to rebuild the Client. However, after changing the configuration file you must restart the Client on the device.

To configure the Software Management Client:

- Create a configuration file, **dma_config.txt**.
  - The default location for **dma_config.txt** is `/system/bin`; a different location can be set when initializing the Software Management Client by *calling the function initEngine*; for details see Initializing and Terminating the Business Logic Layer.

NOTE: This document often uses **dma_config.txt** to refer to this file.

The format of the configuration file is as follows:

- Each line is a key-value pair:
  ```
  key=value
  ```
- No spaces are allowed, not before and not after the equals sign (=).
- No comments are allowed after the value.
- Lines may be empty.
- A comment line begins with the hash sign (#). No spaces are allowed before the hash sign.

Example:

```
dp_path_file=DP_Path_File.txt
# Error logging; print only error messages
```

The following table presents the available parameters that are specific to the Software Management DM Client. You can configure many additional standard configuration parameters. For more information about these parameters, see the *OMA-DM Protocol Engine Framework Integrator's Guide*.

NOTE: The **dma_config.txt** parameters/values listed below must be identical to the corresponding parameter/values in **rb_ua.conf**:

dma_config.txt	rb_ua.conf
dp_path_file	delta_location
ua_result_file	result_file

Table 5-1: Configuration Parameters

Parameter	Description	Value
dp_full_path	The path to the file (`dp_path_file`) that contains the directory in which DPs are stored after they are downloaded. For example: `/cache/dp`	Absolute path The default is `null`, which means that DPs are downloaded to the engine's working directory.
dp_path_file	The name of the file that contains the directory in which DPs are stored after they are downloaded. For example: `DP_Path_File.txt`	File name The file contains an absolute path to the directory used by the Porting Layer function `VDM_SWMC_PL_UA_handoff` The default is `/data/redbend/dp`
dynamic_proxy_config	Whether the engine asks for and expects to receive DM Server and Download Server proxy information using events.	Boolean The default is `false`. See Dynamic Proxy Configuration.
external_dp_support	Whether the engine asks if the DP is too large to be downloaded in a single chunk and handled internally.	Boolean The default is `false`. See Handling Large DPs.
gota_config_path	Absolute path to the directory to which the signed configuration zip file (**rb_ota.zip**) will be copied.	Absolute path The default is `/cache`
handoff_dir	Absolute path to the directory that contains the files specified by `dp_path_file` and `ua_result_file`. For example: `/data/redbend/bin`	Absolute path The default is `/data/redbend/workdir`
init_installers_max_retry	The maximum number of times to retry initializing the installer's init function. If set to `0`, don't retry the installer's init function.	The number of times to retry. The default is `3`.

Parameter	Description	Value
maxnetretries	The maximum number of times to try to reconnect following: • Socket read / write errors • TCP timeout: ◦ Host cannot be reached ◦ Connection refusal ◦ Unresolved address	The number of *additional* times to retry (one retry will always be made) The default is `3`.
report_persistency_max_counter	The maximum number of times to retry uploading a report to the Software Management Server. **NOTE**: If you specify `report_persistency_max_counter` and `report_persistency_timeout`, the Software Management DM Client retries as long as both values are valid. Retries stop if either limit is reached (whichever comes first). Set to 0 to disable report uploading retries.	Integer The default is `10`
report_persistency_timeout	The time, in minutes, to retry uploading a report to the Software Management Server. **NOTE**: If you specify `report_persistency_timeout` and `report_persistency_max_counter`, the Software Management DM Client retries while both values are valid. Retries stop if either limit is reached (whichever comes first). Set to 0 to disable report uploading retries.	Integer The default is `1440` (24 hours)
scomo_battery_threshold	The minimum percentage of the battery charge required for an installation to start.	Integer The default is `0`

Parameter	Description	Value
scomo_clean_dp_node	During the report session, the newly added SCOMO nodes of the DPs are deleted from the tree.	Boolean The default is `False`
is_auto_add_dp_nodes	Adds standard DP nodes automatically when the server requests that a new DP package be added.	Boolean The default is `True`
scomo_ins_confirm_timer_seconds	The time in seconds for the end-user to respond to an installation request.	Integer The default is `300`
ua_exec	Absolute path to the Update Agent executable. For example: `/system/bin/rb_ua`	Absolute path to a file The default is `/system/bin/rb_ua`
ua_result_file	The name of the file into which the Update Agent writes the result of a DP update. For example: `result.txt`	Absolute path to a file The default is `/data/redbend/result`
dp_split_support	Whether the downloaded DP can be split into multiple sections (useful for encrypted devices).	Boolean The default is `false`
trigger_dm_after_install	Whether to automatically trigger a new DM session after an update has been completed successfully (except when the first session is LAWMO).	Boolean The default is `false`
enable_download_segment_limitation	Whether the download segment size and timeframe limitation is enabled.	Boolean The default is `false`
scomo_root_uri	Configurable value for the SCOMO root URI.	String The default is ./SCOMO
enable_lawmo_bl	Whether to enable/disable the LAWMO feature.	Boolean The default is false
enable_lawmo_bl	Whether to enable/disable the LAWMO feature.	Boolean The default is true

Parameter	Description	Value
enable_descmo_bl	Whether to enable/disable the DESCMO feature.	Boolean The default is false
enable_http_bl	Whether to enable/disable the http feature.	Boolean The default is false
enable_http_ui_bl	Whether to enable/disable the http UI	Boolean The default is false
enable_manage_bl	Whether to start the installation phases manager.	Boolean The default is false

5.3 LAWMO Configuration

The Software Management Server supports LAWMO and can lock / unlock or wipe a device (for example, when the device is stolen).

5.3.1 Lock

There are two ways to execute a lock operation:

- The Software Management Server sends the unlock password

  To execute the lock operation on encrypted devices, you *must* set the value in the node **/LAWMO/Ext/RedBend/Password/PwdFromServer** to `true`.

  If the node is set to `true`, the device receives the unlock password from the Software Management Server.
- The Software Management Client generates a new password

  If the node **/LAWMO/Ext/RedBend/Password/PwdFromServer** is set to `false`, the Software Management Client ignores the password sent from the Software Management Server and generates a random password to lock it.

The lock operation is executed by the Software Management Server via a lock campaign.

DM Tree PwdFromServer leaf example:

```
<leaf>
   <name>PwdFromServer</name>
   <get/><replace/>
   <format>bool</format>
   <value>true</value>
</leaf>
```

5.3.1.1 Locking with a Password from the Server

To lock a device using the password sent from the Software Management Server:

1 Make sure that the following nodes exist in the DM Tree. If they don't exist, add them.

- **/LAWMO/Ext/RedBend/Password/PwdFromServer**
- **/LAWMO/Ext/RedBend/Password/Policy**
- **/LAWMO/Ext/RedBend/Password/Pwd**

2 Set the **/LAWMO/Ext/RedBend/Password/PwdFromServer** node in the tree to `true`.

Example:

```
<leaf>
   <name>PwdFromServer</name>
   <get/><replace/>
   <format>bool</format>
   <value>true</value>
</leaf>
<leaf>
   <name>Policy</name>
   <get/><replace/>
   <format>chr</format>
   <value></value>
</leaf>
<leaf>
   <name>Pwd</name>
   <get/><replace/>
   <format>chr</format>
</leaf>
```

3 The node **Pwd** is replaced by the password sent by the Software Management Server. This password is not saved in the tree or anywhere else.

4 The node **Policy** is replaced with the password policy sent by the Software Management Server. The password sent by the Software Management Server must meet this policy.

5 The Software Management Server executes the `FullyLock` operation and the end-user can unlock the device using the password set during the Lock operation.

6 When the device is successfully locked, the Software Management Client returns the result code 1250.

5.3.1.2 Locking with a Random Password

To lock a device using a random password:

1 Set the **/LAWMO/Ext/RedBend/Password/PwdFromServer** node in the DM Tree to `false`.

2 The Software Management Client receives a lock campaign from the Software Management Server.

3 A new password is randomly generated and contains 20 alphanumerical characters.

The Software Management Client does not save the password for security reasons.

4 When the device is successfully locked, the result code returned by the Software Management Client is 1200.

5.3.2 Unlock

There are two ways to unlock the device:

- The end-user enters the unlock password
- Via an unlock campaign from the Software Management Server.

5.3.2.1 Unlocking by the End-User

To unlock a device by the end-user:

- If the leaf **/LAWMO/Ext/RedBend/Password/PwdFromServer** in the DM Tree is set to `true`, the end-user can login into the device with the password set during the Lock operation.

  An unlock campaign from the Software Management Server is rejected if the leaf is set to `true`. The result code returned by the Software Management Client is 1452.

5.3.2.2 Unlocking by the Software Management Server

To unlock a device with a campaign from the Software Management Server:

- If the leaf **/LAWMO/Ext/RedBend/Password/PwdFromServer** is set to `false`, the only way to unlock the device is via an unlock campaign from the Software Management Server.

  The Software Management Client sets the password to an empty string.

6 Configuring the Device Integration Layer

This chapter presents the information you need to configure or create a Device Integration Layer (DIL).

All of the logic of downloading the Download Descriptor (DD), downloading the update, starting installation, reporting the result, and handling interruptions are handled by the business logic or the Engine.

The DIL hands off installations to the installers and gets results from the installers when they are done. The DIL sends information about external events (such as incoming messages or the end-user tapping a key) to the business logic and responds to any requests by the business logic with the requested information.

The DIL collects information about the device (such as battery level, network connectivity) to send to the business logic, displays any required notification or screens to the end-user, and receives any activity that the end-user performs. Essentially, the DIL acts as a tunnel between the business logic and the outside world.

6.1 DIL Architecture Overview

The Software Management Client is built using a layered architecture. The following diagram presents an overview of this architecture.

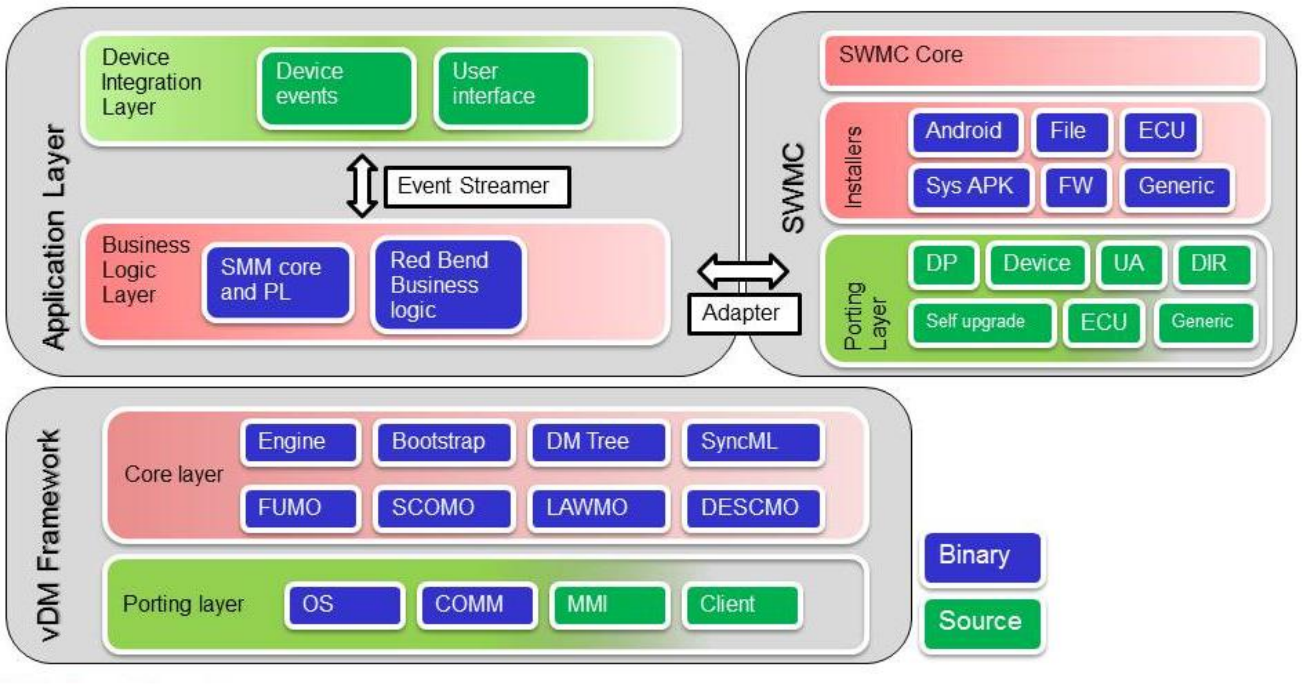


Figure 6-1: Extended Framework Architecture

All layers except for the Porting Layer and the Device Integration Layer (DIL) are platform-independent.

The Porting Layer contains functions that set or get information from the platform at a low level, such as persistent storage and memory management. Redbend provides a Porting Layer for many platforms. You do not need to modify this layer.

The layer that you can configure is the DIL. Redbend provides a complete production-ready DIL for Android and an HTML5 DIL for use on vehicles that support HTML5.

A critical step in implementing the DIL includes sending and receiving the required events to and from the Business Logic Layer (BLL).

6.2 DIL Operation Overview

The DIL initiates and terminates the Software Management Client. Platform traps listen for external events, such as incoming phone calls or messages and, as required, generate BL events to send to the Business Logic Layer.

The DIL contains platform-dependent code required to handle events external to DM Client. For instance, the DIL uses a UI to display information to the end-user and read events (selections, key presses, and so on) generated by the end-user.

The DIL sends and receives events to and from the Business Logic Layer using the Event Streamer.

6.2.1 DIL Flow Overview

For any communication between the Software Management Server and the Software Management Client, the device or the Server initiates a DM session.

When there is an update to install (whether the update is used to upgrade, downgrade, install, or remove the software), the DIL first downloads a download descriptor (DD) with information about the update, including the size and location of the file to download. The Software Management Client then downloads, validates and installs the update. Installation is either performed using a general installer or handed off to the Update Agent.

When installing firmware or other system apps, installation may require two reboots: one to boot into recovery mode to perform the installation and a second to boot from recovery to the new firmware.

Detailed flows are presented in Flows.

6.3 Events

The Software Management Client is fully event-driven, activated by triggering events. Not everything that occurs on the device automatically triggers an event; the DIL decides when to trigger BL events, and the business logic decides when to trigger DIL events.

6.3.1 Event Types

The Client uses three types of events:

- **BL events**: BL events may be generated by the DIL after a message is received from the DM Server, the device receives an incoming call or message, or the end-user interacts with the device (power on, key press, and so on). BL events may also be generated by the DIL in response to DIL events. BL events are sent from the DIL to the Business Logic Layer.

  All BL events are queued in the Business Logic Layer for processing.
- **DIL event**: DIL events are generated by the business logic and sent to the DIL. A DIL event either produces a visible change to the end-user's screen (a new screen, an update to the progress bar, and so on) or triggers another action that must be performed by the platform.

- **Internal event**: Internal events are sent from the Business Logic Layer to itself. Internal events may also be generated by the OMA-DM Protocol Engine when invoking a callback. You can safely ignore internal events that appear in any log files.

6.3.2 Event Structure

An event contains an event name and, optionally, variables:

Event name: Event names are defined within the Application Layer.

Variables: Variables are attached to the event in an associative array. When receiving a Business Logic event, the business logic uses the variables associated with the event to decide what to do: perform an action, transition to another state, or send a DIL event back to the DIL. When receiving a DIL event, the DIL uses the variables associated with the event to determine an action to perform or the change to make on the device screen.

6.3.3 Event Streamer

The Event Streamer sends events between DILs and the business logic.

The Event Streamer is divided into two parts: one platform-independent side resides in the Business Logic Layer and one platform-dependent side resides in the DIL.

On Android, the Event Streamer is implemented using an internal API. If the DIL and Business Logic Layer are running as independent processes, events are first serialized before being sent between the two processes.

6.3.4 Required Events

For the list of required events and their descriptions and variables, refer to the Reference documentation (HTML) included in the delivery package.

The DIL must send the BL events and receive the DIL events listed in this file when the relevant information or request is received. See Flows for common request and response flows.

NOTE: Beside the BL events and DIL events listed in the documentation, the application uses internal events for communication between the different parts of the business logic. You may safely ignore these events if they appear in the logs.

You can also ignore events that are irrelevant to your specific implementation. For example, if the device has no UI, the DIL can safely ignore or send a default response for any DIL event that instructs the DIL to display a screen or prompt the end-user.

6.4 Android Implementation Overview

This section presents an overview of the Android DIL.

6.4.1 Primary vs Secondary Users

Software Management functionality is available to primary users only. If a secondary user attempts to open or run the application, a message will be displayed stating "Only the primary user can run Software Management". The application will open only for the primary user.

6.4.2 DIL Class Overview

The DIL Java implementation contains two main packages:

- **com.redbend.app** (in `common/src/com/redbend/app`)

  The package contains the generic implementation of DIL logic, including:

  - Defining events and event variables
  - Initiating the SMM.

    The SMM manages the business logic in the Business Logic Layer.
  - Sending BL events and receiving DIL events to and from the SMM
  - Assigning class methods to handle incoming DIL events
  - Managing Android Tasks and Activities required by the DIL
  - Defining abstract Activity and broadcast receiver classes, which include functionalities that implement the logic
- **com.redbend.client** (in `redbend/src/com/redbend/client`)

  The package contains specific DIL methods, including:

  - All classes that define the behavior when a certain DIL event is received:
    - Android Activity classes that define UI screens
    - Android broadcast receiver classes that define device traps that send BL events
    - Other general handlers, including end-user notifications
  - A service that extends the generic implementation (`com.redbend.app`), declares all DIL event handlers, and performs other initializations that are needed

The following partial class diagram presents the relationships between some of the main classes.

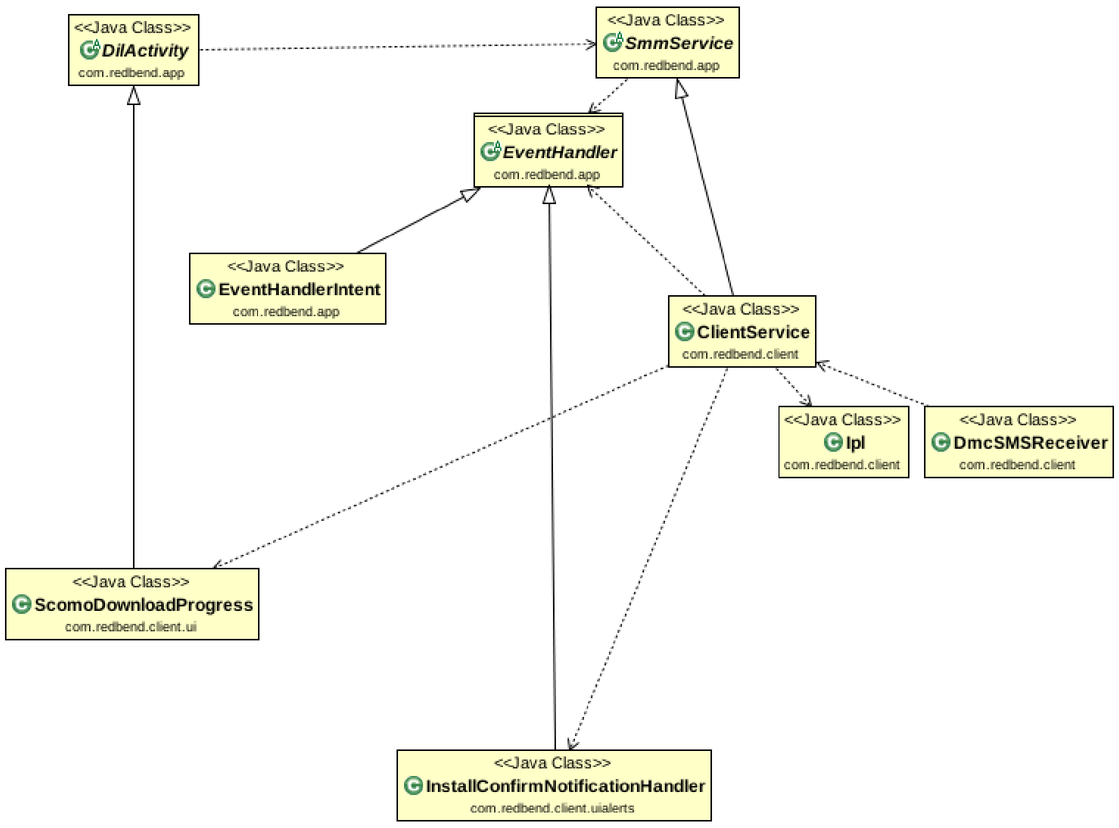


Figure 6-2: DIL Class Relationships

The classes in the diagram are as follows:

- **com.redbend.app**:
  - `SmmService`: This class is an abstract implementation of Android Service. The class manages the DIL.
  - `EventHandler`: This abstract class is common for all classes that receive one or more DIL events.
  - `EventHandlerIntent`: This class is an implementation of `EventHandler`. The class starts an Android Activity when a DIL event is received.
  - `DilActivity`: This class contains common logic for every Android Activity that is displayed as a result of a DIL event. An instance of the class is started from an Intent generated by `EventHandlerIntent`. This class communicates with `SmmService` to create Android Tasks.

    `SMMService` communicates with `DilActivity` to know what's happening with every Activity so that the screens are handled properly. For example, the **Back** button on the FUMO download progress screen (an Activity) must not return to the FUMO download confirmation screen (another Activity).
- **com.redbend.client**:
  - `ClientService`: This is a specific implementation of `SmmService` that declares all `EventHandler` instances.
  - `Ipl`: This class includes Integration Porting Layer functions that you must implement.

- `DmcSMSReceiver`: This class is a specific implementation of `BroadcastReceiver`. The class receives and processes SMSs and creates BL events as required.
- `ScomoDownloadProgress`: This class is a specific implementation of `DilActivity`. The class handles DIL events about download progress during SCOMO.
- `InstallConfirmNotificationHandler`: This class is a specific implementation of `EventHandler`. The class implements how the Android Notification is displayed when the end-user is supposed to confirm an installation.

6.4.3 Event Streamer Overview

In Redbend's Android implementation, the Business Logic Layer is launched as a sub-thread of the DIL application. The DIL is written in platform-dependent Java while the Business Logic Layer is written in native C. The DIL uses JNI to launch a separate thread that manages the Business Logic Layer. This thread launches the Business Logic Layer and provides a callback for returned DIL events.

To pass a BL event to the Business Logic Layer, the DIL serializes the BL event and uses JNI to pass the serialized BL event to the Business Logic Layer thread, which de-serializes the BL event and passes it to the Business Logic Layer. To pass a DIL event to the DIL, the Business Logic Layer invokes a callback that serializes the DIL event and calls Java using JNI. The DIL de-serializes the DIL event for processing.

The DIL and Business Logic Layer can be joined using a variety of other implementations that are not covered in this guide.

6.4.3.1 Event Streamer Deliverables

The following files contain code associated with the Event Streamer:

- **Event.java**: Java event class. Includes (de)serialization of events.
- **EventVar.java**: Java event variables class. Includes (de)serialization of event variables.
- **BasicService.java**: Java class that sends serialized BL events to the Business Logic Layer and receives serialized DIL events using JNI. This class receives DIL events from the Business Logic Layer, wraps them inside Android intents, and broadcasts them to `ClientService.java`. In the reverse direction, this class receives intents from `ClientService.java`, unwraps the events, and passes them to the Business Logic Layer.
- **dma_jni.c**: JNI functions that receive, de-serialize, and send BL events from the DIL to the Business Logic Layer (using **dma_sm_exec.c**), and receive, serialize, and send DIL events from the Business Logic Layer to the DIL (using a callback).

Events and the methods used to act on them are defined by the classes **Event.java** and **EventVar.java**.

6.4.4 Initializing and Terminating the Business Logic Layer

The DIL must initialize and terminate the SMM of the Business Logic Layer.

To Initialize using BasicService.java:

1 The DIL initializes the engine using `initEngine and` passes two parameters:
`initEngine(filesDir, configFile);`
  - `filesDir`: The absolute path to the engine's working directory; a String value.

- `configFile`: The path and file name of the configuration file, a String value. When null, the default configuration file **/system/bin/dma_config.txt** is used.

2 The DIL starts the SMM.
`startSmm(deviceId, userAgent, deviceModel, deviceManufacturer);`

- `userAgent` is a String value.
- `deviceId, deviceModel`, and `deviceManufacturer` are `DevNodeValue` class instances:

```
class DevNodeValue{
   String value;
   boolean forceReplace;
   DevNodeValue(String inValue, boolean inForceReplace) {
      value = inValue;
      forceReplace = inForceReplace;
   }
}
```

- If `inValue` is null, read the value from the IPL (see Device Information Functions).
- If `inValue` is specified, take the value from the DM Tree. In this case, if `forceReplace` is set to `true`, the value replaces any existing value in the DM Tree. Otherwise, the value is set in the DM Tree only if it does not already exist.

6.4.5 Event Streamer Initialization

The Software Management DM Client initializes the Business Logic Layer side of the Event Streamer by calling the following function, passing a callback to process DIL events.

```
typedef void (VDM_SMM_sendUIEventFunc)(VDM_SMM_event_t*
event_name);
int VDM_SMM_init(DMA_sendUIEventFunc sendFunc);
```

The DIL includes the classes `Event` and `EventVar` used to process the events in Java.

6.4.5.1 Initial BL Events

On initialization, the DIL sends to the Business Logic Layer the current network state using the following BL events:

`DMA_MSG_STS_MOBILE_DATA`

`DMA_MSG_STS_ROAMING`

`DMA_MSG_STS_WIFI`

`DMA_MSG_PRODUCT_TYPE`

6.4.6 Sending and Receiving Events

The following sections present an overview of how to send and receive events using the Event Streamer.

6.4.6.1 Sending BL Events from the DIL

The DIL must first construct the event. An event has a name and associated variables that are passed with the event. Variables include a variable name, integer value or string value.

```
Event event   = new Event(event_name) // Construct the event
```

```
EventVar var  = new EventVar(event_name,intval) // or
EventVar var  = new EventVar(event_name,strval)
                                    // Create event variable value
event_name.addVar(var)              // Add variable/value to event
```

The method used to send the event depends on the use case:

- To send an event originating from the Android platform, use `SmmReceive.sendEvent`:

  ```
  SmmReceive sendEvent(context, class, event_name)
  ```

  Where `context` is the context of the receiver required for transmitting the event, `class` is the class of the associated service (`ClientService`) that is required for transmission, and `event` is the event.
- To send an event originating from end-user input in a UI screen, use `DmActivity.sendEvent`:

  ```
  DmActivity sendEvent(event_name)
  ```
- To send an event originating from a UI screen process, but not as a result of end-user interaction, use `SmmService.sendEvent`:

  ```
  SmmService sendEvent(event_name)
  ```

The class that passes the BL event using JNI is a private method `SmmService.ipcSendEvent`. The non-serialized BL event (which is encapsulated in the `Event` class) is passed to the public method `SmmService.sendEvent`, which serializes the BL event and then calls the native `ipcSendEvent`. Using JNI, this executes the native function `Java_com_redbend_app_SmmService_ipcSendEvent` (in **dma_jni.c**).

6.4.6.2 Receiving DIL Events in the DIL

To handle DIL events, Redbend's DIL first registers a handler for the event in `ClientService.eventHandlersRegister` as follows:

```
// Construct the event
Event event = new Event(event_name)
EventVar var = new EventVar(event_name,intval) // or
EventVar var = new EventVar(event_name,strval)
event_name.addVar(var)
...

// Define the handler:
// To handle a certain event by displaying an activity on the UI
screen
// use:
event_handler = new EventHandlerIntent(context,
ActivityClass.class);

// If you have a class FooHandler that implements the EventHandler
// interface, use:
event_handler = new FooHandler(...);

// Register the handler
registerHandler(flowId, event_name, ui_mode, event_handler);
```

Where `flowId` is a number representing a logical flow (all handlers belonging to the same logical flow have the same flow ID) and `ui_mode` is one of:

- `UI_MODE_FOREGROUND`: This handler only handles events when the application is in the foreground.
- `UI_MODE_BACKGROUND`: This handler only handles events when the application is in the background.
- `UI_MODE_BOTH_FG_AND_BG`: This handler handles events regardless of whether the application is in the foreground or background.

The Java method `SmmService.recvEvent` de-serializes and handles the DIL event. The DIL event is de-serialized using the `Event` constructor that receives a byte array as a parameter.

6.5 Generic Installers

6.5.1 Description

Generic installers are installers used to handle specific types of software. The Software Management Client passes all updates that are marked with a generic installer type to that installer and waits for the results of the update. The Software Management Client can handle generic installers that reboot the device.

The engine gets each *software instance* in the DP using the Porting Layer function:

- `getNextComponent` (Android)

The engine gets *software attributes* (such as software version) using the Porting Layer function:

- `getComponentAttribute` (Android)

For updates and installations, whenever software uses a generic installer, the business logic passes the location of the delta or DP to the DIL using `B2D_MSG_SCOMO_GENERIC_INSTALL_REQUEST`. The business logic also passes the offset and size of the software within the file (i.e., for a delta, the offset is 0 and the size is the file size; for a DP the offset can be larger than 0 and the size can be smaller than the DP size). The procedure is to open the file and read (starting from the offset) the number of bytes specified as the size (to avoid unneeded file system access) and to check whether the *specified* file size is the *actual* file size).

The DIL then passes operation to the generic installer. After the installation has succeeded or failed, the DIL returns the results using `D2B_MSG_SCOMO_INSTALL_RESULT`.

Generic installers that require the device to reboot must follow the same flow as the Redbend Update Agent and require the same Porting Layer functions. For more information, see Update Agent Handoff Flow and Backward Compatibility.

For more information about installer types, see Appendix Verifying DP Authenticity.

6.5.2 Generic Installer Installation Flow

The flow describes the generic installer installation process. The process requires implementation of the Porting Layer functions described in Generic Installer Functions.

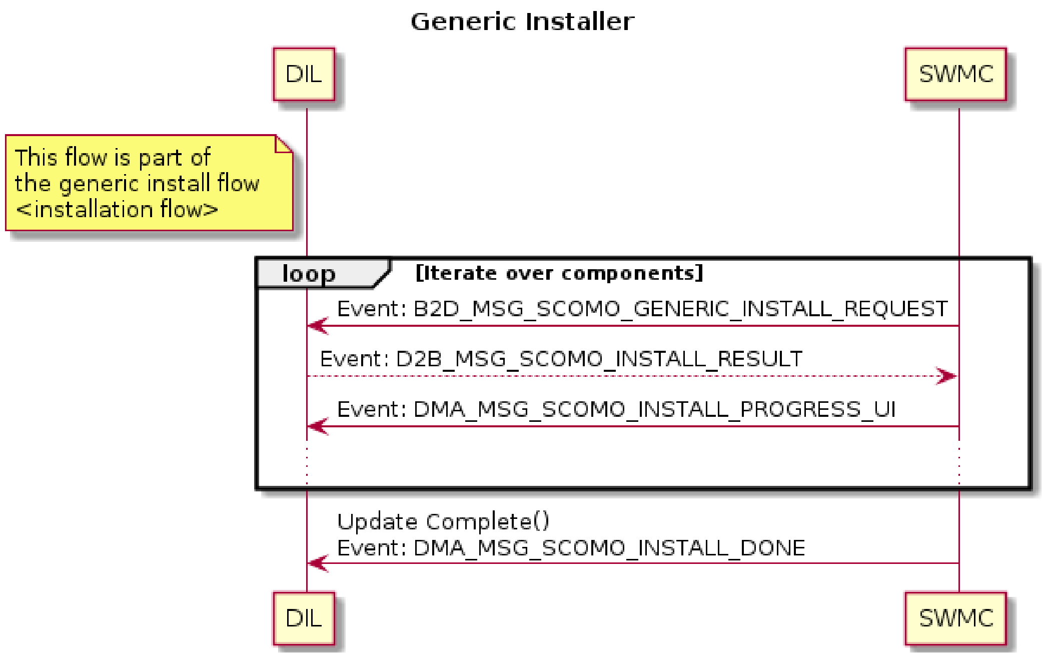

Generic Installer
DIL
SWMC
This flow is part of the generic install flow <installation flow>
loop
[Iterate over components]
Event: B2D_MSG_SCOMO_GENERIC_INSTALL_REQUEST
Event: D2B_MSG_SCOMO_INSTALL_RESULT
Event: DMA_MSG_SCOMO_INSTALL_PROGRESS_UI
Update Complete()
Event: DMA_MSG_SCOMO_INSTALL_DONE
DIL
SWMC

Figure 6-3: Generic Installer Installation Flow

Generic Installer Installation Flow:

This flow details part of the installation flow (see Installation Flow).

1 From the START LOOP (see Figure Generic Installer Installation Flow): The generic installer gets the DP location and starts a loop to iterate each software component in the DP.

2 The generic installer assigns an installer number for the software component, gets the software component offset and length in the DP and sends a "block" event to the business logic requesting that it install the software component. The DIL invokes an installer to install the software component once it receives the event.

3 When the DIL installer finishes installing the component, the DIL sends an event that notifies the business logic of the installation result.

4 The business logic notifies the DIL with the current installation progress.

END LOOP: The generic installer iterates over the next software component and repeats from step 1.

5 The business logic notifies the DIL that the installation is complete.

6.5.3 Generic Installer Sync Inventory Flow

A *sync software inventory* process updates the DM Tree to ensure that it matches the software installed on the device.

A *generic installer sync inventory* synchronizes only the software installed using a generic installer.

This flow describes how the Software Management DM Client passes control to, and reads results from, the generic installer performing a generic installer sync inventory. The process requires you to implement the Porting Layer functions described in Generic Installer Functions.

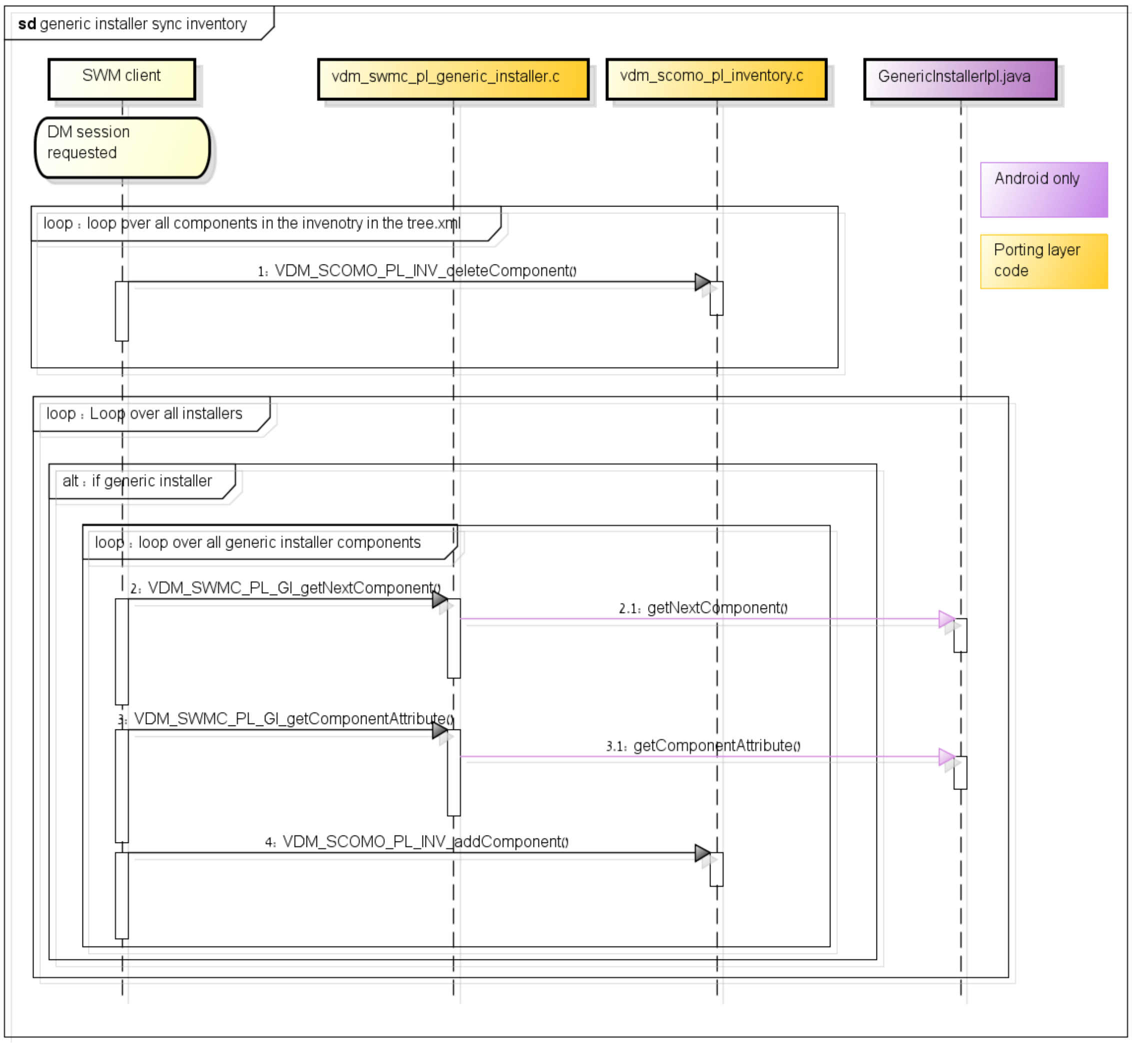

sd generic installer sync inventory
SWM client
vdm_swmc_pl_generic_installer.c
vdm_scomo_pl_inventory.c
GenericInstallerIpl.java
DM session requested
Android only
Porting layer code
loop : loop pver all components in the invenotry in the tree.xml
1: VDM_SCOMO_PL_INV_deleteComponent()
loop : Loop over all installers
alt : if generic installer
loop : loop over all generic installer components
2: VDM_SWMC_PL_GI_getNextComponent()
2.1: getNextComponent()
3: VDM_SWMC_PL_GI_getComponentAttribute()
3.1: getComponentAttribute()
4: VDM_SCOMO_PL_INV_addComponent()

Figure 6-4: Generic Installer Sync Inventory Flow

Generic Installer Sync Inventory Flow:

1 A DM session is initiated by the end-user, device or server. The Software Management Client loops over and removes all software in the DM Tree.

The Software Management Client loops over all installers. For each generic installer:

2 The Software Management Client invokes the Porting Layer function `VDM_SWMC_PL_GI_getNextComponent` to get the software instance ID.

3 The Software Management Client invokes the Porting Layer function `VDM_SWMC_PL_GI_getComponentAttribute` to get the software instance version.

4 The Software Management Client invokes the Porting Layer function `VDM_SCOMO_PL_INV_addComponent` to add the software instance to the DM Tree.

The loop continues until there are no more software components. **Note**: This process is quick, however, if the DIL sends an event to the business logic during this process, the event is ignored.

6.5.4 Generic Installer Example

We can use the generic installer mechanism to update maps on the device. Pseudo code for implementing the functions defined in **VDM_Client_PL_Storage_sync.h** (see section 7.1.7) is given below:

```
// Initialize the generic installer helper.
//In Android,this initialzies the JNI object.
VDM_Error VDM_SWMC_PL_GI_initInstallerHelper(void *pUserData)
{
   initMapsInstaller();
}

// Iterate over all installed software instances.
VDM_Error VDM_SWMC_PL_GI_getNextComponent(
void     *installerType,// Generic installer type; this must match
                        // the number in the DP
void     **ioIt,        // Current software
UTF8Str  outId,         // Next software ID
IU32     *ioIdSize      // Buffer size for ID
)
{
   SWMC_GenericInstall_t * comp;
   ...
   // Initialize the comonent list if needed
   If (!(*ioIt)
      initializeComponentsList();

   comp = (SWMC_GenericInstall_t *)*ioIt;
   // Verify comp is valid
   ...
   // Set get the next component and return its values
   *ioIt = comp->next;
   *ioIdSize = comp->iLen;
    VDM_PL_strncpy((char*)outId, (char*)comp->id, comp->iLen + 1);
   ...
   return VDM_ERR_OK;
}

// Get the installed software attribute.
VDM_Error VDM_SWMC_PL_GI_getComponentAttribute(
void                *installerType, // Generic installer type
UTF8CStr        inId,           // Software ID
SWM_component_attr  inAttr,         // Attribute type
UTF8Str             outBuffer,      // The attribute
IU32                inBufferSize    // Buffer size for attribute
)
{
   // Return the required attribute based on installer type
   // and attribute type
   switch(inAttr)
      {
         case(SWM_COMP_ATTR_NAME):
            ret = getCompName(installerType, inId, outBuffer,
               inBufferSize);
            break;
         case(SWM_COMP_ATTR_DESC):
            ret = getCompDescription(installerType, inId, outBuffer,
               inBufferSize);
            break;
         case(SWM_COMP_ATTR_VER):
            ret = getCompVersion(installerType, inId, outBuffer,
               inBufferSize);
         break;
         case(SWM_COMP_ATTR_TYPE):
            ret = getCompType(installerType, inId, outBuffer,
            inBufferSize);
            break;
         default:
            ret = VDM_ERR_INVALID_CALL;
            break;
       }
}
```

DIL Implementation (pseudo code):

```
// DIL needs to handle B2D_MSG_SCOMO_GENERIC_INSTALL_REQUEST sent from
// SMM.
VDM_Error handleGenericInstallRequest(..)
{
   // Assign the install request parameters to local varaibles
   extractEventVariables(..);

   // Use the required installer to install the component
   // In this example we are using instller type 200 for
   // installing maps component
   switch(installerType)
   {
       case (IT_GENERIC_TYPE_200):
            result = installMapsComp(path, offset, length, mode, compId);
            break;
   }

    // Send the install result back to BL
    ret = VDM_SMM_postEventOverIpcEx("D2B_MSG_SCOMO_INSTALL_RESULT",
            VDM_SMM_allocVarUintEx("DMA_VAR_SCOMO_INSTALL_COMP_RESULT",
                    (IU32)result), NULL);
}
```

6.6 Configuring Automatic Self-Registration

The DIL uses DMA_MSG_AUTO_SELF_REG_INFO to set the domain name and domain PIN in the DM Tree.

6.6.1 Android Self-Registration

The DIL can be set to automatically self-register the device to a domain in `ClientService.onCreate`. Default registration credentials (domain name and PIN) can be set in the DM Tree.

To change the credentials without having to rebuild the Client, the DIL also retrieves credentials by calling `Ipl.iplGetAutoSelfRegDomainInfo`, passing an empty array to store the credentials. This function reads the information from the file **<sdcard_directory>/private/Credentials.txt**. The first line in the file must be the domain name, and the second line must be the PIN. If the credentials file does not exist, no event is sent to the DIL.

NOTE: The credentials file is read only on client service creation, meaning that the device must be rebooted after the file is pushed (or the application process can be killed).

6.7 Dynamic Proxy Configuration

If the configuration parameter `dynamic_proxy_config` is set to `true`, the engine sends the DIL event `B2D_MSG_PROXY_CONFIGURATION_REQUEST` before every DM session and expects the DIL to return the Business Logic event D2B_MSG_PROXY_CONFIGURATION_RESPONSE with DM Server and Download Server proxy URLs.

6.8 Handling Large DPs

When the DP is too large to store in one piece on the device (internally), the Software Management Client can iteratively download parts of the DP to *external storage*.

When the configuration parameter `external_dp_support` is set to "True", the business logic layer sends the event `B2D_GET_INSTALL_TYPE` to decide whether or not to download the DP in parts.

The DIL returns `D2B_SET_INSTALL_TYPE`, which specifies either internal or external storage.

There are now three types of DPs:

- Internal installation
- External DL and External Installation
- External Installation

External DP Support

For externally stored DPs, the engine sends `B2D_GET_EXTERNAL_CONFIGURATION`; the DIL returns `D2B_SET_EXTERNAL_CONFIGURATION`, which includes the buffer size. If the download is interrupted and part of the DP is already stored, the event also includes the remaining space available and the size of the partially stored DP.

As each part of the DP is received, the business logic sends the buffer to the DIL using `B2D_BUFFER_READY`. The DIL stores the buffer in external storage and returns `D2B_BUFFER_TRANSMITTED` with the number of bytes that were written to storage.
For the flow used to handle large DPs, see Large DP Flow.

6.8.1.1 Download - External

The DIL responds with `D2B_SET_INSTALL_TYPE`, indicating either internal or external storage.

For an externally stored DP, the engine sends `B2D_GET_EXTERNAL_CONFIGURATION` and the DIL returns `D2B_SET_EXTERNAL_CONFIGURATION`, which includes the buffer size. If the download was interrupted and part of the DP was already stored, this event also includes the remaining space available and the size of the partially stored DP.

As each part of the DP is received, the business logic sends the buffer to the DIL using `B2D_BUFFER_READY`. The DIL stores the buffer in external storage and returns `D2B_BUFFER_TRANSMITTED` with the number of bytes that were written to the storage.

For the flow used to handle large DPs, see DP Flow.

6.8.1.2 Installation - External

The Business Logic Layer requests external installations using this event and its corresponding variables:

Event Name

B2D_MSG_SCOMO_EXTERNAL_INSTALL_REQUEST

Variables

- DMA_VAR_SCOMO_DP_PATH
  This is the path to one (1) or multiple update packages, e.g., /tme/redbend/
- DMA_VAR_SCOMO_DP_NAME
  This variable is for one (1) or multiple update package names.

Note: Each name must be separated by the delimiter: **0x1f**.

6.8.1.3 FUMO Installations

FUMO installations use these events and variables.

Event Name (Single Installation File)

D2B_MSG_SCOMO_EXTERNAL_INSTALL_RESULT

Variable

DMA_VAR_SCOMO_INSTALL_COMP_RESULT

Result

SUCCESS or FAILIURE. After receiving the result, the session continues to the report and finishes.

Event Name (Multiple Installation Files)

D2B_MSG_SCOMO_EXTERNAL_INSTALL_DC_RESULT

Variables

- DMA_VAR_SCOMO_DC_ID
- DMA_VAR_SCOMO_DC_NAME
- DMA_VAR_SCOMO_DC_VERSION
- DMA_VAR_SCOMO_DC_DESC
- DMA_VAR_SCOMO_DC_ENVTYPE
- DMA_VAR_SCOMO_DC_ISACTIVE
- DMA_VAR_SCOMO_DC_INSTALL_RESULT

The DIL responds with results for each installed component. Even when a specific variable is not relevant, it *must be sent* but can be: Empty, NULL, or 0.

The Business Logic Layer will be in a state that enables it to receive component results until it receives the "Done" event with the installation result:

Event Name (Done)

D2B_MSG_SCOMO_EXTERNAL_INSTALL_DONE

Variables

DMA_VAR_REPORT_RESULT. The DIL reports error / success using customer codes.

6.9 Authenticating Events

The Software Management Client provides a mechanism to authenticate events when using sockets to communicate between the DIL and the Business Logic Layer. The mechanism adds a signature to each event package so that the receiver can ensure that the event is valid. In addition, you can create a whitelist of events that are accepted for processing even if there is no signature or the signature is invalid.

Use `VDM_IPC_registerSignatureCalcFunc` to register a callback function `VDM_IPC_calculateSignatureCB_t` that creates the signature, either to add to the event package or after receiving an event. Use `VDM_IPC_registerEventFilterFunc` to register a callback function `VDM_IPC_filterEventCB_t` that evaluates whether to proceed with the event after the signature's validity has been evaluated.

For more information about these functions, see Event Authentication Functions.

6.10 Configuring Silent Update and Installation

The DIL handles DIL events marked for *silent installation* (`DMA_VAR_SCOMO_ISSILENT=1`) without notifying the end-user. The DIL must return an "accept" Business Logic event on behalf of the end-user (`DMA_MSG_USER_ACCEPT`) when required, as if the end-user had tapped **OK** on a notification screen.

6.10.1 Android Example

The following code snippets from `ClientService.eventHandlersRegister()` define two of the event handlers.

- **Non silent installation**: If the DIL event `DMA_MSG_SCOMO_DL_CONFIRM_UI` is received, *and* it is *not* marked for silent installation (`DMA_VAR_SCOMO_ISSILENT=0`), then handle the event with `ScomoConfirm`.

```
registerHandler(
   1,
   new Event("DMA_MSG_SCOMO_DL_CONFIRM_UI").addVar(new
      EventVar("DMA_VAR_SCOMO_ISSILENT", 0)),
   UI_MODE_BOTH_FG_AND_BG,
   new EventHandlerIntent(this, ScomoConfirm.class)
);
```

- **Silent installation**: If the DIL event `DMA_MSG_SCOMO_DL_CONFIRM_UI` is received, *and* it *is* marked for silent installation (`DMA_VAR_SCOMO_ISSILENT=1`), then return `DMA_MSG_SCOMO_ACCEPT` on behalf of the end-user.

```
registerHandler(
   1,
   new Event("DMA_MSG_SCOMO_DL_CONFIRM_UI").addVar(new
      EventVar("DMA_VAR_SCOMO_ISSILENT", 1)),
   UI_MODE_BOTH_FG_AND_BG,
   new EventHandler(this) {
      @Override
      protected void genericHandler(Event ev) {
         sendEvent(new Event("DMA_MSG_SCOMO_ACCEPT"));
      }
   }
);
```

To change this functionality, remove the second event handler and the variable requirement (`.addVar(new EventVar("DMA_VAR_SCOMO_ISSILENT", 0))`) in the first handler.

Do this for all locations where alternate handling is performed for `DMA_VAR_SCOMO_ISSILENT`.

6.11 Configuring the Rule Engine

The business logic determines when a DM session can start or an installation can proceed. By default, a DM session starts when there is network connectivity or connectivity to a Wi-Fi point. Also by default, an installation proceeds when the end-user has given approval during a time slot specified by the Software Management Server, if the battery is above a minimum threshold.

You can modify or define additional configuration requirements using the Rule Engine. The Software Management Client loads a Rule Engine configuration file before proceeding with the relevant flow. The following Rule Engine configuration files may exist:

- DM Session: **before_dm_conditions.xml**
- Installation:
    - Pre-installation phase: **before_pre_install_conditions.xml**

- Installation phase: **before_install_conditions.xml**
- Post-installation phase: **before_post_install_conditions.xml**

6.11.1 Checking Rules in the Rule Engine Flow

When a process that checks rules in the Rule Engine is set to start, the business logic sends `B2D_GET_ALL_CONDITIONS_VALUES` to the DIL. For some rules the business logic already knows the value (see Internally Managed Rules); the DIL does not have to do anything about them. For all other rules, the DIL has 60 seconds to return the values of these rules. You must add code that checks and returns these values using the appropriate handler for this DIL event.

The DIL returns the values using a series of Business Logic events `D2B_CONDITION_VALUE_UPDATE(DMA_VAR_CONDITION_VAR_NAME, DMA_VAR_CONDITION_VAR_VALUE)`, for example, `D2B_CONDITION_VALUE_UPDATE("DIL_AMBIENT_LIGHT", 25)`. The business logic parses the Rule Engine configuration file and compares the values returned by the DIL with the required, minimum, or maximum values specified in the file.

The Software Management Client proceeds as follows:

- **DM Session**: If the business logic receives values for all of the rules listed in the configuration file within 60 seconds and they evaluate to true, the Software Management Client continues with the process. Otherwise the Software Management Client suspends the process and sends `DMA_MSG_DM_ERROR_UI` with the error text set in the configuration file.
- **Installation**: If the business logic receives values for all of the rules listed in the configuration file within 60 seconds and all top level clauses evaluate to true, the Software Management Client sends `DMA_MSG_SCOMO_INS_UI_CONDITIONS` with `DMA_VAR_CONDITIONS_MET=1` and continues with the process. Otherwise the Software Management Client suspends the process and sends `DMA_MSG_SCOMO_INS_UI_CONDITIONS` with `DMA_VAR_CONDITIONS_MET=0` requesting that the DIL display an error screen. The texts regarding each of these rules are set in the configuration file.

6.11.2 Internally Managed Rules

The rules handled internally by the business logic are:

- `NETWORK_STATUS`: `0`: No data connection. `1`: Data connection.
- `SERVER_NOTIFICATION_TYPE`: How the message was received. `0`: GCM. `1`: WAP message. `2`: N/A.
- `ROAMING_STATUS`: `0`: Not roaming. `1`: Roaming.
- `SESSION_INITIATOR`: `0`: Server-initiated. `1`: Device-initiated. `3`: End-user-initiated.
- `SESSION_ACTION_TYPE`: `0`: Regular session. `1`: LAWMO session. `2`: Purge Update session. Session types `1` and `2` require the Software Management Client to cancel any in-process session.

  NOTE: LAWMO unlock operations are currently marked as type 0 (regular session).
- `SYNC_INSTALLERS_INIT_STATUS`: `0`: The installer's initialization flow completed successfully. `Non-zero`: The installer's initialization flow completed unsuccessfully. See the configuration parameter `init_installer_max_retry` in Configuring the Software Management Client.
- `WIFI_STATUS`: `0`: Not connected to a Wi-Fi point. `1`: Connected to a Wi-Fi point.

- UNEXPECTED_REBOOT_DURING_INSTALL: 0: There were no unexpected reboots during the installation session. 1: An unexpected reboot occurred while installing software components. The flag is cleared when the installation session finishes (after the report is sent) and also before installing the first software component.

Note that if you want these rules to be checked, they must still be added to the Rules Engine configuration file. However, you do not have to return these values when prompted for rules by the business logic.

6.11.3 Rule Engine File Format

Rule Engine files must be in XML format; the tags are described in the following table. The files must be located in the assests/files directory in the Android delivery package. Additional requirements, if any, are presented in the examples that follow the table.

Rules managed internally are described in the preceding section. Rules that you create must be prefixed with DIL_, for example DIL_AMBIENT_LIGHT.

Tag	Containing Tags	Description
AND	None, IF	Group IF clauses, or group EQ, GT, and/or LT clauses If all of the clauses within this clause are true, the group evaluates to true. Otherwise, the group evaluates to false.
OR	None, IF	Group IF clauses, or group EQ, GT, and/or LT clauses If any of the clauses within this clause are true, the group evaluates to true. Otherwise, the group evaluates to false.
IF	None, AND, OR	Denotes a clause that can evaluate to true or false based on the EQ clause it contains. Must contain exactly one THEN tag and one ELSE tag, and one of the following tags: AND, OR, EQ, GT, or LT.
EQ	IF, OR, AND	Evaluates to true if the current value of the specified condition (ID) exactly matches the specified value (INT). Otherwise evaluates to false. For example, when the business logic requests the current status of all of the conditions, one of the DIL's responses may be D2B_CONDITION_VALUE_UPDATE("AMBIENT_LIGHT", 25). If the ID specified in this EQ clause is "AMBIENT_LIGHT" and the INT specified is 25, the EQ clause evaluates to true. If the INT value is 10, the EQ clause evaluates to false. Must contain exactly one ID tag and one INT tag.
GT	EQ, OR, AND	Evaluates to true if the current value of the specified condition (ID) is greater than the specified value (INT). Otherwise evaluates to false. Must contain exactly one ID tag and one INT tag.
LT	EQ, OR, AND	Evaluates to true if the current value of the specified condition (ID) is less than the specified value (INT). Otherwise evaluates to false. Must contain exactly one ID tag and one INT tag.

Tag	Containing Tags	Description
ID	EQ, GT, LT	A rule to evaluate. The string must match a string returned by the DIL using D2B_CONDITION_VALUE_UPDATE.
INT	EQ, GT, LT	An integer value to compare to the current value of the condition.
THEN	IF	A message (string) to display on the screen is the EQ, GT, or LT clause evaluates to true.
ELSE	IF	A message (string) to display on the screen is the EQ, GT, or LT clause evaluates to false.

6.11.3.1 Example DM Session Rule Engine File

All rules for this file must be contained in a single IF/THEN/ELSE clause.

```
<IF>
   <AND>
      <GT>
         <ID>DIL_Rule1</ID>
         <INT>20</INT>
      </GT>
      <LT>
         <ID>DIL_Rule1</ID>
         <INT>30</INT>
      </LT>
      <EQ>
         <ID>WIFI_STATUS</ID>
         <INT>1</INT>
      </EQ>
   </AND>
   <THEN>true message [currently not used]</THEN>
   <ELSE>false message header;;false message subheader</ELSE>
</IF>
```

6.11.3.2 Example Installation Rule Engine File

If not specified, all top level rules are assumed to be within an AND clause.

```
<AND>
   <IF>
      <EQ>
         <ID>DIL_Rule1</ID>
         <INT>10</INT>
      </EQ>
      <THEN>true message</THEN>
      <ELSE>false message header;;false message subheader</ELSE>
   </IF>
   <IF>
      <AND>
         <GT>
            <ID>DIL_Rule2</ID>
            <INT>20</INT>
         </GT>
         <LT>
            <ID>DIL_Rule2</ID>
            <INT>30</INT>
         </LT>
      </AND>
      <THEN>true message</THEN>
      <ELSE>false message header;;false message subheader </ELSE>
   </IF>
   <IF>
      <OR>
         <EQ>
            <ID>DIL_Rule3</ID>
            <INT>40</INT>
         </EQ>
         <EQ>
            <ID>DIL_Rule4</ID>
            <INT>50</INT>
         </EQ>
      </OR>
      <THEN>true message</THEN>
      <ELSE>false message header;;false message subheader</ELSE>
   </IF>
</AND>
```

7 Software Management DM Client Porting Layer Functions

The Software Management DM Client is delivered with a complete set of Porting Layer functions for Android. The Client Porting Layer Functions (see the *Client Porting Layer Functions* chapter of the *OMA-DM Protocol Engine Framework Integrator's Guide*) are delivered in source format and can be modified as required.

This chapter presents some additional Porting Layer functions delivered in source format that can be modified. For more information about each function, refer to the Reference documentation (HTML) included in the delivery package.

7.1 Native Porting Layer Functions

7.1.1 Device Information Functions

These functions, defined in **vdm_swmc_pl_device.h**, retrieve basic device information.

```
// Get device model VDM_Error VDM_SWMC_PL_Device_getModel(
   UTF8CStr   outModel,
   IU32       *ioModelSize);

// Get device vendor
VDM_Error VDM_SWMC_PL_Device_getManufacturer(
   UTF8CStr   outMan,
   IU32       *ioManSize);

// Get current firmware version
VDM_Error VDM_SWMC_PL_Device_getFWVersion(
   UTF8CStr   outFWVersion,
   IU32       *ioFWVersionSize);

// Get device ID
VDM_Error VDM_SWMC_PL_Device_getId(
   UTF8CStr   outId,
   IU32       *ioIdSize,
   void       *context);
```

You must implement these functions to return the indicated information.

7.1.2 File Search Functions

These functions, defined in **vdm_swmc_pl_dir.h**, allow the Client to search for a file that matches a search string (may include the `*` and `?` wildcards).

```
// Create a handle to a list of files in a directory
```

```
VDM_Error VDM_SWMC_PL_Dir_create(
   void       **outHandle,
   UTF8CStr   inPath);

// Get next file name in the handle
VDM_Error VDM_SWMC_PL_Dir_getNextFile(
   void       *inHandle,
   UTF8Str    outBuffer,
   IU32       *ioBufferLen);

// Close the handle
VDM_SWMC_PL_Dir_destroy(void* inHandle);
```

7.1.3 Event Authentication Functions

These functions, defined in **vdm_ipc.h**, enable you to add a signature with an event to ensure the authentication of the sender.

```
// Create authentication context.
void VDM_IPC_createAuthenticationContext(void *inContext);

// Callback to calculate signatures for buffer.
typedef int (*VDM_IPC_calculateSignatureCB_t)(
   void                   *inContext,
   const unsigned char    *inAuthBuffer,
   IU32                   inAuthBufferSize,
   char                   *outSignature,
   IU32                   *ioSignatureSize
);

// Register callback function for calculating signature
void
VDM_IPC_registerSignatureCalcFunc(VDM_IPC_calculateSignatureCB_t
inCb);

// Evaluate whether to continue processing an event after
evaluating the
// event buffer's signature. You might continue processing certain
events
// even after a failed signature evaluation.
typedef IBOOL (*VDM_IPC_filterEventCB_t)(
   void                   *intContext,
   const VDM_SMM_Event_t      *inEvent,
   IBOOL                  inIsAuthorizeEvent);

// Register callback function for events filter.
```

```
void VDM_IPC_registerEventFilterFunc(VDM_IPC_filterEventCB_t inCb);
```

Example implementation (there is a more detailed example in **dma/app/linux2/common**):

```
typedef struct
{
   char *key;
   int size;
} VDM_IPC_Secret_t;

static int eventAuthFunc(
   void                 *inContext,
   const unsigned char*inAuthBuffer,
   IU32                 inAuthBufferSize,
   char                 *outSignature,
   IU32                 *ioSignatureSize)
{
   VDM_IPC_Secret_t * currentKey = (VDM_IPC_Secret_t *)inContext;
   *ioSignatureSize = currentKey->size;
   VDM_PL_memcpy(outSignature, currentKey->key, *ioSignatureSize);
}

static IBOOL eventFilterFunc(
   void                     *intContext,
   const VDM_SMM_Event_t        *inEvent,
   IBOOL                    inIsAuthorizeEvent)
{
   if (VDM_PL_strncmp ("DMA_MSG_NET_BOOTSTRAP ", inEvent->name,
fileFilterEventSize) == 0)
   {
// We match it to the black list; this event is not authorized!!!
      return FALSE;
   }
   return TRUE;
}

int main()
{
   VDM_IPC_Secret_t secretKey;
   secretKey.key = (char*)malloc(10);
   secretKey.size = 10;
   for (int ind=0; ind<10; ind++)
   secretKey.key[ind] = ind;

   VDM_IPC_createAuthenticationContext((void *)&secretKey);
   VDM_IPC_registerSignatureCalcFunc(eventAuthFunc);
   VDM_IPC_registerEventFilterFunc(eventFilterFunc);
}
```

7.1.4 Deployment Package Wrapper Function

This function, defined in **vdm_swmc_pl_dp.h**, validates the signature of the DP.

Set the `outOffset` parameter with the actual offset of the beginning of the DP.

It is recommended that you use `SWM_DP_readBufferFromDP()` and `SWM_DP_getDPSize()`, described in the following section, to read the DP.

The function contains sample code for signature validation.

```
VDM_Error VDM_SWMC_PL_Dp_validateExternalSignatureDp(
     IU32          *outOffset);
```

7.1.5 DP Access Functions

These functions, defined in **swm_dp_access.h**, wrap DP management functions.

```
// General function to get data from a Deployment Package
extern VDM_Error SWM_DP_readBufferFromDP(IU32 in_offset,
   void* out_buffer,
   IU32  in_buffer_len,
   IU32 *out_read_count);

// Returns the DP size, as read from the header of the DP,
including signature offset
IU32 SWM_DP_getDPSize(void);
```

7.1.6 Self-Update Function

This function, defined in **vdm_swmc_pl_self_upgrade.h**, is called by the engine for each DM Tree node that must be added to the DM Tree after a self-update. This function calls the Java equivalent.

```
// Get values of new nodes after self-update
VDM_Error VDM_SWMC_PL_SelfUpgrade_getNodeDefaultValue(
   const char *inUri,
   char       *outDefaultValue,
   IU32       *ioValueSize);
```

7.1.7 Generic Installer Functions

These functions, defined in **vdm_swmc_pl_generic_installer.h**, are called by generic installer to get installed software information.

```
// Initialize the generic installer helper. For exmample, in Android,
// this initialzies the JNI object.
VDM_Error VDM_SWMC_PL_GI_initInstallerHelper(
void  *pUserData        // The application context
);
```

```
// Iterate over all installed software instances.
VDM_Error VDM_SWMC_PL_GI_getNextComponent(
void     *installerType,// Generic installer type; this must match
                        // the number in the DP. For more information
                        // on installer types,
                        // see Appendix Authenticating a Self-Certified Server

void     **ioIt,        // Current software
UTF8Str  outId,         // Next software ID
IU32     *ioIdSize      // Buffer size for ID
);

// Get the installed software attribute.
VDM_Error VDM_SWMC_PL_GI_getComponentAttribute(
void                *installerType, // Generic installer type
UTF8CStr        inId,           // Software ID
SWM_component_attr  inAttr,         // Attribute type
UTF8Str             outBuffer,      // The attribute
IU32                inBufferSize    // Buffer size for attribute
);
```

7.2 Java Porting Layer Functions

The following IPL functions are in `android/redbend/src/com/redbend/client`.

7.2.1 iplGetAutoSelfRegDomainInfo

Description

This function gets self-registration credentials from the external file **Credentials.txt** (see Generic Installer Installation Flow) and, if they exist, sets them in the DM Tree.

Declaration

```
public static int iplGetAutoSelfRegDomainInfo(String
   []autoSelfRegDomainInfo)
```

Parameters

Parameter	Description
autoSelfRegDomainInfo	An array of two elements: a domain name and a PIN.

Return Values

Value	Description
-1	Error
0	Success

7.2.2 getDevModel

Description

This function gets the device model.

Declaration

```
public static String getDevModel()
```

Parameters

None

Return Values

Value	Description
String	The device model

7.2.3 getManufacturer

Description

This function gets the device vendor.

Declaration

```
public static String getManufacturer()
```

Parameters

None

Return Values

Value	Description
String	The device vendor

7.2.4 getFwVersion

Description

This function gets the firmware version.

Declaration

```
public static String getFwVersion()
```

Parameters

None

Return Values

Value	Description
String	The firmware version

7.2.5 getDeviceId

Description

This function gets the device ID: the IMEI for devices with a SIM card, the MAC address (for tablets, for example) .

Declaration

```
public static String getDeviceId(Context ctx)
```

Parameters

Parameter	Description
ctx	Service context

Return Values

Value	Description
String	The device ID

7.2.6 getUserAgent

Description

This function gets the device User Agent, which is used in the user agent header during HTTP transactions.

Declaration

```
public static String getUserAgent(Context ctx)
```

Parameters

Parameter	Description
ctx	Service context

Return Values

Value	Description
String	The device User Agent

7.2.7 getNextComponent

Description

This function iterates over all installed software instances.

Declaration

```
public String getNextComponent(int type, int[] iter)
```

Parameters

Parameter	Description
type	Generic installer type; this must match the number in the DP. For more information about installer types, see Installer Types.
iter	The first time this function is called, it contains an array with the first element set to -1. Increment this by 1 each time the function is called. The same array is sent as is on each subsequent call.

Return Values

Value	Description
String	Next software ID, or null when there is no more software

7.2.8 getComponentAttribute

Description

This function gets all attributes of an installed software instance.

Declaration

```
public ComponentInfo getComponentAttribute(int type, String
componentId)
```

Parameters

Parameter	Description
type	Installer type
componentId	Software ID

Return Values

Value	Description
ComponentInfo	The software information structure

7.2.9 getDefaultValue

Description

This function gets the current default value for a DM Tree node. It is called by the engine for each DM Tree node that must be added to the DM Tree after a self-update.

Declaration

```
public static String getDefaultValue(String Uri)
```

Return Values

Value	Description
-1	Error
0	Success

Note: Refer to Configuration Parameters in Tree.xml.

7.3 Update Agent Porting Layer Functions

These functions, defined in **vdm_swmc_pl_ua.h**, provide a means for the Software Management DM Client to send and receive information to and from the Update Agent.

The Software Management Client uses `VDM_SWMC_PL_UA_deltaApply` to invoke the Update Agent.

```
VDM_Error VDM_SWMC_PL_UA_deltaApply(
char  *inSourcePathFile,      // Path to the file to update
char  *inDeltaPathFile,       // Path to the delta
char  *outTargetPathFile);    // Path to place the updated file (it
can't
                              // be the same as inSourcePathFile)
```

The Software Management DM Client uses `VDM_SWMC_PL_UA_handoff` to set the location on the device to which DPs are stored after they are downloaded. The location is stored in a file. The Update Agent can then use this information to perform an update after a reboot.

```
VDM_Error VDM_SWMC_PL_UA_handoff(
   char  *inDpPath,        // Path to DP storage
   char  *inDpPathFile,    // File in which to store the path to the
                              DP
   UTF8CStrinHandoffDir); // Directory in which to store the file
```

After an update, the Software Management DM Client uses `VDM_SWMC_PL_UA_getResult` to get the result and remove the DP.

```
VDM_Error VDM_SWMC_PL_UA_getResult(
   IU32     *outUpdateResult, // Pointer to file to store result
   char     *inResultFile,    // File which currently stores
result;
                              // removed by this function
   char     *inDpPathFile);   // DP to remove
```

7.4 Configuration Parameters in Tree.xml

Parameter	Description	Default Value	Immediate Effect
ActiveStatuses	The list of statuses sent from the client can be preset as “none” or ”basic” or “full”. In addition, **additional statuses** can be added (or subtracted) to (or from) the above presets (see Statuses and Details). **Example 1**: ststrk.active_statuses=none No statuses are sent. **Example 2**: ststrk.active_statuses= basic -DL_Started +INST_Deferred The statuses sent by this definition are those in the basic preset *minus* the status “DL_Started” *plus* the status “INST_Deferred”. **Example 3**: ststrk.active_statuses= full -DL_Interrupted Management Server sets the node: ./Ext/RedBend/StatusTracking/ActiveStatuses	N/A	Yes
DlResumeMaxCounter	The maximum number of times to retry an interrupted download (Integer). The parameter is now configurable by the server and is saved in the tree. For backwards compatibility it is still configurable by **dma_config.txt**. **Note**: If you specify dl_resume_max_counter and dl_resume_timeout, the Software Management DM Client retries while both values are valid. Retries stop if either limit is reached (whichever comes first). Set to 0 to disable download resume. Management Server sets the node /ext/RedBend/DlResumeMaxCounter	10	Yes

Parameter	Description	Default Value	Immediate Effect
DlResumeTimeout	The time, in minutes, to retry an interrupted download (Integer). The parameter is now configurable by the server and is saved in the tree. For backwards compatibility it is still configurable by **dma_config.txt**. **Note**: If you specify **dl_resume_timeout** and **dl_resume_max_counter**, the Software Management DM Client retries while both values are valid. Retries stop if either limit is reached (whichever comes first). Set value 0 to disable download resume. Management Server sets the node /ext/RedBend/DlResumeTimeout	1440 minutes (24 hours)	Yes
DmBootupMinDelay	This parameter enables slow systems to start properly before starting a DM session that follows any power up event. This delay is the minimum time to wait prior to starting a DM session following a bootup. It applies to all DL sessions and DM sessions: server initiated, client initiated, and user initiated. **Note**: The default delay is 0 seconds. The default value is 0s. You can define this value as: hours (h), minutes (m), seconds (s). Reads the node: /ext/RedBend/DmBootupMinDelay	0 (sec)	
ExternalDownloadTimeout	The timeout for downloading an update. The **default value** is 72h. You can define this value as: hours (h), minutes (m), seconds (s). The Software Management Server sets the node: **./Ext/RedBend/ExternalDownloadTimeout**	72h	Yes

Parameter	Description	Default Value	Immediate Effect
LastDLFailTime	Status of last backend connection. The Software Management Server sets the node: **./Ext/RedBend/Diagnostics/**LastDLFailTime / **Note**: The following parameters must exist in dma_config and be "True" (default is "false"): • enable_diagnostics_bl = True • enable_tree_bl = True	N/A	N/A
LastDLStatus	Status of last map OTA download. The Software Management Server reads the node: **/ext/RedBendDiagnostics/**LastDLStatus. Allowed values are: • Completed • DL_FAILED:<error number> **Note**: Error numbers are located in the file **`vdm_error.h`**. • DL_INTERRUPTED:<error number> **Note**: Error numbers are located in the file **`vdm_error.h`**. • Never_performed (initial value) • Resumed • Started **Note**: The following parameters must exist in dma_config and be "True" (default is "false"): • enable_diagnostics_bl = True • enable_tree_bl = True	N/A	N/A

Parameter	Description	Default Value	Immediate Effect
LastDLSuccessTime	Date and timestamp of last successful OTA map download. The Software Management Server reads the node: **/ext/RedBend/Diagnostics/**LastDLSuccessTime. **Note**: The following parameters must exist in dma_config and be "True" (default is "false"): • enable_diagnostics_bl = True • enable_tree_bl = True	N/A	N/A
LastDLTime	Date and timestamp of last map OTA download attempt. The Software Management Server reads the node: **/ext/RedBend/Diagnostics/**LastDLTime. **Note**: The following parameters must exist in dma_config and be "True" (default is "false"): • enable_diagnostics_bl = True • enable_tree_bl = True	N/A	N/A

Parameter	Description	Default Value	Immediate Effect
LastDMConnStatus	Status of last backend connection. The Software Management Server reads the node: **/ext/RedBend/Diagnostics/**LastDMConnStatus. Allowed values are: • Cancelled • Completed • DM_FAILED:<error number> **Note**: Error numbers are located in the file **`vdm_error.h`**. • Never_Performed (initial value) • No_Update • Started • Updated_Needed **Note**: The following parameters must exist in dma_config and be "True" (default is "false"): • enable_diagnostics_bl = True • enable_tree_bl = True	N/A	N/A
LastDMConnSuccessTime	Date and timestamp of last successful backend connection. The Software Management Server reads the node: **/ext/RedBend/Diagnostics/**LastDMConnSuccessTime. **Note**: The following parameters must exist in dma_config and be "True" (default is "false"): • enable_diagnostics_bl = True • enable_tree_bl = True	N/A	N/A

Parameter	Description	Default Value	Immediate Effect
LastDMConnTime	Date and timestamp of last backend connection attempt. The Software Management Server reads the node: **/ext/RedBend/Diagnostics/**LastDMConnTime. **Note**: The following parameters must exist in dma_config and be "True" (default is "false"): • enable_diagnostics_bl = True • enable_tree_bl = True	N/A	N/A
PollingIntervalInHours	The interval value, in hours, between a successful poll and the next one. The initial default value is 168 (hours) Set to 0, to disable. Previously this value was configured through dma_config.txt. Currently, if the node does not exist in the tree, it is added on startup and includes the default value. The Software Management Server sets the node: **/ext/RedBend/PollingIntervalInHours**.	168 (One week)	Yes
PostponeMaxTimes	The maximum number of times that an end-user can postpone an action. Set to 0, to disable. The Software Management Server sets the node: **/ext/RedBend/PostponeMaxTimes**.	3	
PostponePeriod	The time, in minutes, that an action can be postponed by the end-user. The initial default value is 60 (minutes). Set to 0 to disable. The Software Management Server sets the node: **/ext/RedBend/PostponePeriod**.	60	

Parameter	Description	Default Value	Immediate Effect
RecoveryPollingMaxCounter	• This is the maximum number of times to attempt to perform a Recovery Polling. Each unsuccessful DM/DL attempt increments the counter by one and doubles the amount of time before the next attempt until reaching the amount of time that is specified by ./Ext/RedBend/RecoveryPollingTimeout . • This reattempt process continues until the number of attempts reaches either the value of the Counter or until reaching the delay time specified by the parameter RecoveryPollingTimeout. At that point, reattempts occur at the timeout intervals set in by RecoveryPollingTimeout. Set to 0 to disable. Server and client initiated. • **Note**: The initial timeout delay is 1 minute. Sets the node /ext/RedBend/RecoveryPollingMaxCounter. **Note**: Both **RecoveryPollingTimeout** and **RecoveryPollingMaxCounter** are persistent.	10	Yes

Parameter	Description	Default Value	Immediate Effect
RecoveryPollingTimeout	• The maximum delay time between recovery retries. The default value is 1440m (1 day). You can define this value as: hours (h), minutes (m), seconds (s) • The regular polling interval is replaced by a recovery polling interval when a DM or a DL has failed whether initiated by a device or by a server. The recovery polling interval uses a back off algorithm; the first DM retry occurs after 1 minute; the second retry occurs after 2 minutes, then 4, 8, 16, …, minutes until reaching (or exceeding) the value of the RecoveryPollingTimeout. Once the maximum delay time is reached (or exceeded), retries occur at the delay time specified by the RecoveryPollingTimeout parameter. Attempts continue until a successful DM/DL. Server and client initiated. • **Note**: If you specify both RecoveryPollingTimeout and RecoveryPollingMaxCounter, each polling attempt increments a counter until the counter reaches the value set in the parameter. The maximum delay time is determined by the first of the RecoveryPolling parameters that reaches its maximum first (i.e., the smaller of the two). Recovery will continue with this value (and should not be reset to the Polling interval in hours). • Set to 0 to disable this timeout. Management Server sets the node: /ext/RedBend/RecoveryPollingTimeout. • **Note**: Both **RecoveryPollingTimeout** and **RecoveryPollingMaxCounter** are persistent.	1440m (1 day)	Yes

Parameter	Description	Default Value	Immediate Effect
ReserveDownloadTime	A download time slot during which the Software Management Client is permitted to download from the Software Management Server. This value is ignored if it is not valid or if the device is within Wi-Fi range. The Software Management Server sets the node: **/ext/RedBend/ReserveDownloadTime**.	00:00 – 23:59	
SegmentSize	Number of KBs that can be downloaded within the timeframe defined by the **SegmentWindow** parameter. Set to 0 to disable the Download Segments in Time Windows functionality. The Software Management Server sets the node: **./Ext/RedBend**/DownloadLimit/**SegmentSiz e**	Integer The default is 125000	Yes
SegmentWindow	The size of the time limitation window in **minutes** to which the parameter **SegmentSize** applies. Set to 0 to disable the Download Segments in Time Windows functionality. The Software Management Server sets the node: **./Ext/RedBend**/DownloadLimit/**SegmentWi ndow**	Integer The default is 120m	Yes
UserInteractionTimeoutInterval	The interval value, in minutes, that the Software Management Client application will wait for the end-user to approve a download, start an installation, or provide additional input. During this timeout interval, the Software Management Server cannot update the device. The initial default value is 1440 (minutes). The Software Management Server sets the node: **/ext/RedBend/UserInteractionTimeoutInte rval**.	1440 minutes (1 day)	

Polling Persistency

The values of all polling parameters are time persistent; this includes:

- Polling interval
- Recovery polling
- External DL

Backward Compatibility

Backward compatibility is maintained for remote upgrade of devices running older Software Management Client versions in which modified parameters are held in **dma_config**. Following an upgrade, the relevant parameters are read from **dma_config** and are written into **tree.xml**.

7.5 Other Configuration Parameters

The Software Management Client uses parameters that can be configured; they are found in **/data/data/com.redbend.client/files/dma_config.txt.** Refer also to Session Tracking.

Table 7-1: Configuration Parameter Descriptions

Parameter	Description	Default Value
dl_resume_max_counter	The maximum number of times that the system tries to resume a download session.	`10`
dl_resume_timeout	The maximum time (in minutes) to try and resume a download session.	`1440` (One day)
log_pack_work_dir	This parameter consists of a string which is the name of the base directory in which the zipped log files are created before they are sent to the server.	The current directory
maxnetretries	The number of *additional* times (one retry is always made) to try to reconnect following: • Socket read / write errors • TCP timeout: ◦ Host cannot be reached ◦ Connection refusal ◦ Unresolved address No retries are made following a fatal error	`3`
report_persistency_max_counter	The number of times that the Software Management Client tries to send a report to the Software Management Server.	`10`

Parameter	Description	Default Value
report_persistency_timeout	The maximum time (in minutes) that the Software Management Client tries to send a report to the Software Management Server. The time is from the moment an interrupt occurs.	`1440` (One day)
scomo_battery_threshold	The minimum percentage of the battery charge required for an installation to start.	`0`
scomo_ins_confirm_timer_seconds	The timeout (in seconds) of the display of the Start Installation screen for a critical update; if the end-user is inactive during this period, the installation starts automatically.	`300`
ststrk.report_interval_sec	The Minimum Report Interval defines the minimum amount of time between the sending of status updates. It is managed and configured by the Client. The default time interval for this is 30 seconds. Increments are made in units of seconds. This parameter is also available in the Tree.xml (configurable from the Client). Configurable from both the Tree.xml and dma_config.txt. **To implement:** Add these parameters (see Session Tracking): • enable_concurrent_dm_dl=TRUE • ststrk.active_statuses=<as defined in ststrk.active_statuses.>	`30s`

Parameter	Description	Default Value
ststrk.active_statuses	The list of statuses sent from the client can be preset as "none" or "basic" or "full". In addition, **additional statuses** can be added (or subtracted) to (or from) the above presets (see Statuses and Details). **Example 1**: ststrk.active_statuses=none No statuses are sent. **Example 2**: ststrk.active_statuses= basic -DL_Started +INST_Deferred The statuses sent by this definition are those in the basic preset *minus* the status "DL_Started" *plus* the status "INST_Deferred". **Example 3**: ststrk.active_statuses= full -DL_Interrupted	None

Appendix A Reference DIL

A.1 Files Created by the Reference DIL

On Android, the reference DIL writes engine log output to **vdm.log** in the working directory.

A.2 Running the Command Line Reference DIL

To run the Client:

NOTE: To have the Client start automatically, add the executables to **init.rc**.

1 Make sure that the DM Tree is in the same directory as the executables.

2 Start **smm.exe** in the background.

```
./smm.exe &
```

3 Start **client.exe**.

```
./client.exe
```

A menu is displayed:

```
1)FUMO Session
2)Accept Event (download or installation)
3)Cancel Event (download or installation)
4)Send Device Data Available (TRUE)
5)Send Device Data Available (FALSE)
6)Send WIFI Available (TRUE)
7)Send WIFI Available (FALSE)
8)Send Power Up Event
9)Send B2B_EXTERNAL_DL_COMPLETED Success Event
10)Send B2B_EXTERNAL_DL_COMPLETED Failure Event
11)Send AT Command
12)Send External Install Type Event
13)Send External Install Configuration Event
14)Simulate External Install Done Event
15)Send Install Success Event
16)Send Install Failed Event
17)Send Number of Bytes Transmitted during an External Install
Event
18)Send DIL Start Event
19)Send Fullylock Success Event
20)Send Fullylock Success with reboot
21)Send Fullylock Failed Event
22)Send UnLock Success with reboot
23)Send UnLock Success Event
```

```
24)Send UnLock Failed Event
25)Send User Login Event
26)Send external dl connection available
27)Send external dl connection not available

1000)Command Line Console
```

A.2.1 Using the Command Line Console

The command line console provides a means to send events to the business logic.

To run the command line console:

- Start the Client and type `1000`.

  There is no response to this. Instead you can now enter events to send to the business logic using the command `event`.

  ```
  event <BL event name> <var type>(<var> <value>)[...]
  ```

  Variables and values cannot have spaces. Each command ends with a newline.

  For example, to tell the business logic that the device is within Wi-Fi range:

  ```
  event DMA_MSG_STS_WIFI uint(DMA_VAR_STS_IS_WIFI_CONNECTED,1)
  ```

To display help:

- In the command line console, type `help event`.

  A help message is displayed:

  ```
  event - Post a event over ipc.
   Usage: event <name> [<var0> <var1> ... <var63>]
   The following form of event variable are supported:
    str(key,value)  - string event variable
    int(key,value)  - integer event variable
    uint(key,value) - unsigned integer event variable
    b64(key,value)  - base64 encoded binary event variable
    hex(key,value)  - hexadecimal encoded binary event variable
  ```

Examples:

- Notify about Wi-Fi connectivity change.

  ```
  event DMA_MSG_STS_WIFI uint(DMA_VAR_STS_IS_WIFI_CONNECTED,1)
  event DMA_MSG_STS_WIFI uint(DMA_VAR_STS_IS_WIFI_CONNECTED,0)
  ```

- Notify about mobile data connectivity change.

  ```
  event DMA_MSG_STS_MOBILE_DATA
  uint(DMA_VAR_STS_IS_MOBILE_DATA_CONNECTED,1)
  event DMA_MSG_STS_MOBILE_DATA
  uint(DMA_VAR_STS_IS_MOBILE_DATA_CONNECTED,0)
  ```

- Notify about roaming status change.

  ```
  event DMA_MSG_STS_ROAMING uint(DMA_VAR_STS_IS_ROAMING,1)
  event DMA_MSG_STS_ROAMING uint(DMA_VAR_STS_IS_ROAMING,0)
  ```

- Notify about device reboot.

  ```
  event DMA_MSG_STS_POWERED
  ```

- Notify that end-user initiated a SCOMO session.

```
event DMA_MSG_SESS_INITIATOR_USER_SCOMO
uint(DMA_VAR_START_DM,1)
```

- Notify that end-user accepted or rejected a confirmation.

```
event DMA_MSG_SCOMO_ACCEPT
event DMA_MSG_SCOMO_CANCEL
```

A.2.2 Simulating a Generic Installation from the Menu

To run a generic installation from the menu:

1 Create a plan file, **<SCOMO_ID>.plan**, to simulate responses from the DIL to the SWM Client. For the format of plan files, see Simulating Generic Installations: Plan Files.

2 Place **<SCOMO_ID>.plan** in the same directory as **client.exe**.

3 Set up a non-push campaign in the Software Management Server for a generic installation.

4 Start the Client and enter the following data, in the order shown. Unless noted, enter the next item after receiving the responding events from the Software Management Server.

   a 6

   This notifies that the device is within Wi-Fi range (`DMA_MSG_STS_WIFI`). There is no response to this, so continue on to the next step without waiting.

   b 1

   This starts a FUMO session (`DMA_MSG_SESS_INITIATOR_USER_SCOMO`).

   c 2

   This confirms the download (`DMA_MSG_SCOMO_ACCEPT`).

   d **2**

   This confirms any Rule Engine installation rules.

   e 2

   This confirms the installation (`DMA_MSG_SCOMO_ACCEPT`).

On success, **generic_installer_components.txt** is updated with the software instance name (this file is created automatically on installation and located in the `/client` directory). Each line in this file lists a software version as `software ID=version name`.

A.2.3 Simulating Generic Installations: Plan Files

A *plan* file is a file named **<SCOMO_ID>.plan** that contains simulated responses from a device to the Client. Each line in the file is a response, and can be one of the following:

- `E_FILE_Command_progress=<percent>`, where `<percent>` is an integer from 0 to 100. This simulates an installation progress response.
- `E_FILE_Command_sleep=<seconds>`, where `<seconds>` is an integer indicating the number of seconds that the reference application sleeps before parsing the next line. This simulates a response delay.
- `E_FILE_Command_updateResult=0`, indicating a successful installation.

For example:

```
E_FILE_Command_progress=12
E_FILE_Command_progress=17
E_FILE_Command_sleep=5
E_FILE_Command_progress=40
```

```
E_FILE_Command_progress=66
E_FILE_Command_updateResult=0
```

When simulating installation phases (`pre_install_phase` or `post_install_phase` in **dma_config.txt**), create addition plan files for each device:

- **<SCOMO_ID>_PreInstall.plan**
- **<SCOMO_ID>_PostInstall.plan**

These files have the same format as **<SCOMO_ID>.plan**.

A.2.4 Simulating Generic Installations: Detail Files

Detail files (**<SCOMO_ID>.data)** contain additional information (e.g., part number, etc.,) regarding a specific component. The values are integer with the same format as **<SCOMO_ID>.plan**, above. For example:

- `E_FILE_Command_partNumber=9999,Simulating an External Installation from the Menu`

To simulate an external installation from the menu:

1 Set up a non-push campaign in the Software Management Server for a generic installation.

2 Start the Client and enter the following, in order. Unless noted, enter the next item after receiving the responding events from the Software Management Server.

a 6

This notifies that the device is within Wi-Fi range (`DMA_MSG_STS_WIFI`). There is no response to this, so continue on to the next step without waiting.

b 1

This starts a FUMO session (`DMA_MSG_SESS_INITIATOR_USER_SCOMO`).

c 12

You are prompted to indicate if this is an internal or external installation (`B2D_GET_INSTALL_TYPE`).

d 1

This indicates an external installation (`D2B_SET_INSTALL_TYPE`).

e 13

This sets the external installation configuration. You are prompted to enter external installation configuration settings (`B2D_GET_EXTERNAL_CONFIGURATION`).

f 0

This sets the default buffer size, in bytes (300,000) (`DMA_VAR_BUFFER_SIZE`).

g 0

This sets the default max free size for a DP (`DMA_VAR_MAXSIZE`).

h 0

This sets the simulated package size (`DMA_VAR_FILESIZE`).

i 2

This confirms the download (`DMA_MSG_SCOMO_ACCEPT`).

j 17

This prompts you to indicate whether to manually send the size of each chunk received.

k 0

This disables sending responses to "buffer ready" events (D2B_BUFFER_TRANSMITTED).

l 2

This confirms the installation (DMA_MSG_SCOMO_ACCEPT).

m 14

This simulates that the result of the external installation. You are prompted for the result (B2D_MSG_SCOMO_EXTERNAL_INSTALL_REQUEST).

n 0

This simulates external installation success (D2B_MSG_SCOMO_EXTERNAL_INSTALL_RESULT).

On success, **extern_installer_components.txt** is updated with the software instance name. Each line in this file lists a software version as `software ID=version name`. The full DP is placed into the file **RB_DP_APPEND** in the same directory as **client.exe**.

Simulate Reading/Replacing Tree Node Values from the Menu

The event mechanism can be used to simulate reading and replacing the values of tree nodes from the menu.

Formats Supported

The following formats are supported:

- bool and chr
- uri key=value;format key=value; value key=value
  For example, key=value;key1=value1; key2=value2;

The information collected, such as that shown below, is saved into the file *nodesDefine.txt*.

- uri=./DevInfo/Lang;format=chr;value=ENG
- uri=./DevInfo/DmV;format=chr;value=DAI_DM_NTG6_01
- uri=./DevInfo/Ext/IsProductive;format=bool;value=true
- uri=./DevDetail/HwV;format=chr;value=1234567890_001
- uri=./DevDetail/SwV;format=chr;value=1234567890_001_123456

A.2.5 Simulating a Download Using an External Connection from the Menu

To simulate a download using an external connection from the menu:

1 Set up a non-push campaign in the Software Management Server for a generic installation.

2 Start the Software Management Client and enter the following, in the order listed. Unless noted otherwise, enter the next item after receiving the responding event from the Software Management Server.

a 26

This notifies that the device has an external connection available (D2B_DMA_MSG_STS_EXTERNAL_CONNECTIVITY).

b 1

This starts a FUMO session (DMA_MSG_SESS_INITIATOR_USER_SCOMO). Once the session completes, you are prompted to confirm the download (DMA_MSG_SCOMO_DL_CONFIRM_UI).

c 2

This confirms the download (`DMA_MSG_SCOMO_ACCEPT`). The external download starts now `(B2B_START_EXTERNAL_DL)`.

d 9

This indicates that the external download completed successfully `(B2B_EXTERNAL_DL_COMPLETED)`. You are prompted to confirm installation `(DMA_MSG_SCOMO_INS_CONFIRM_UI)`.

e 2

This confirms the installation (`DMA_MSG_SCOMO_ACCEPT`). You are prompted to confirm reboot (`DMA_MSG_SCOMO_REBOOT_CONFIRM_REQUEST)`.

f 2

This confirms the reboot (`DMA_MSG_SCOMO_ACCEPT`).

g 15

This simulates the installation success event after reboot.

Appendix B Flows

The following diagrams illustrate typical flows for Software Management Client flow processes, including many of the events used.

B.1 Overview Flow

This diagram presents a general overview of the Software Management Client flows.

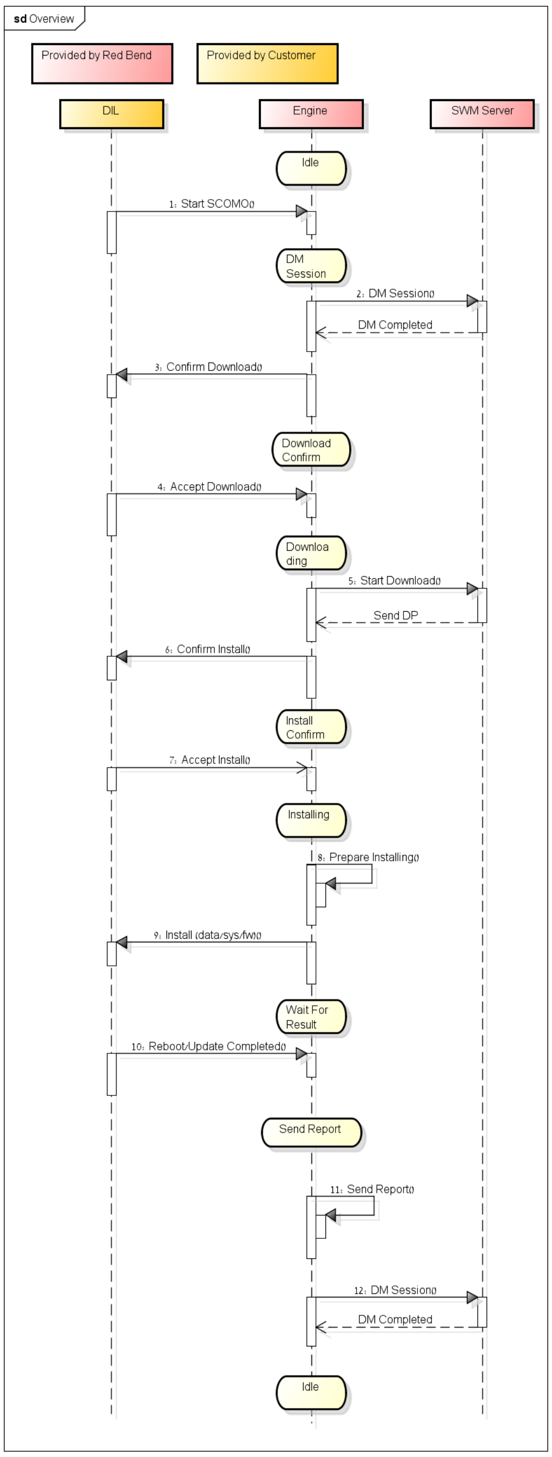


Figure B-1: Flow Overview

B.2 DM Session Flow

This flow describes a DM session.

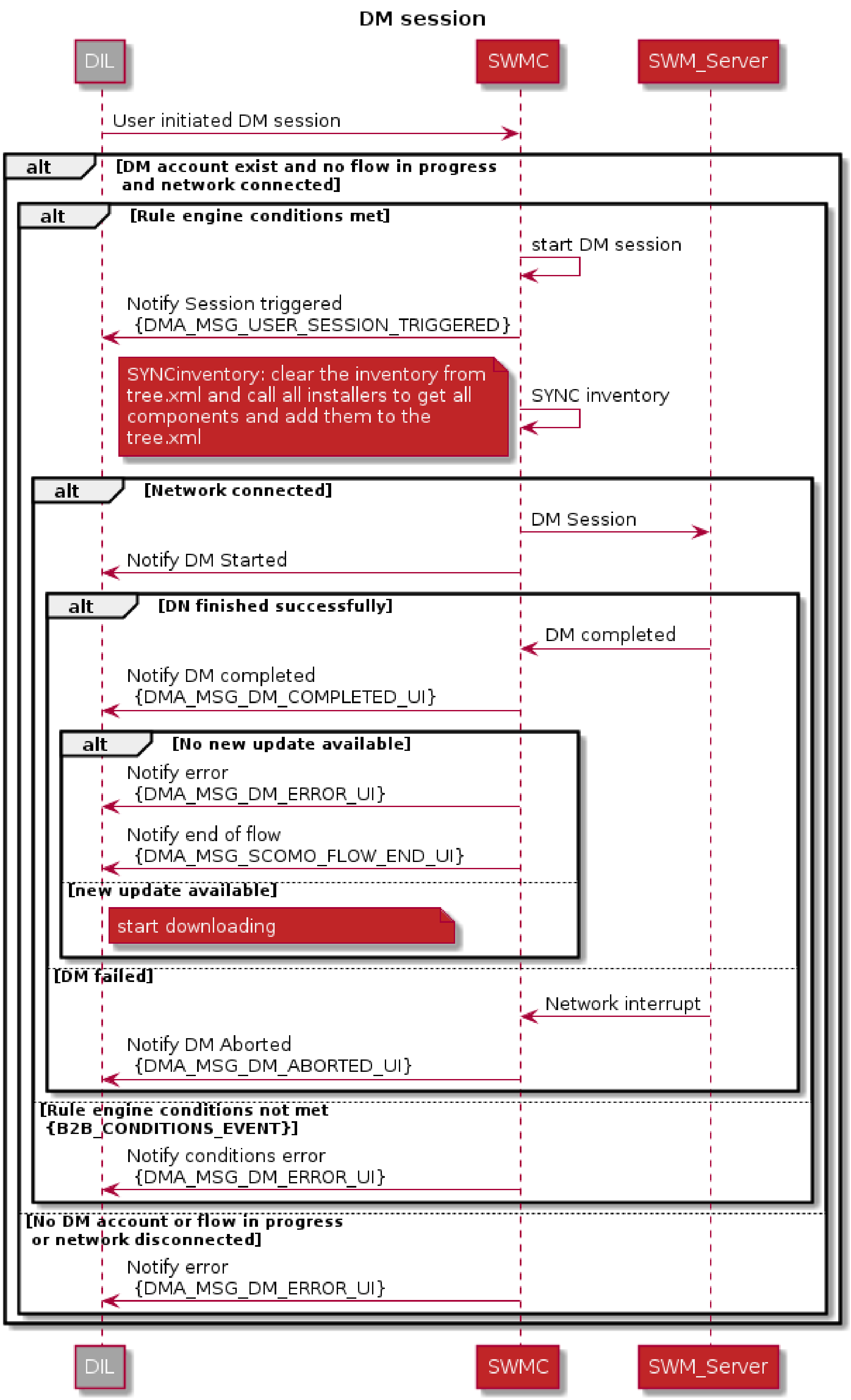

DM session
DIL
SWMC
SWM_Server
User initiated DM session
alt [DM account exist and no flow in progress and network connected]
alt [Rule engine conditions met]
start DM session
Notify Session triggered {DMA_MSG_USER_SESSION_TRIGGERED}
SYNCinventory: clear the inventory from tree.xml and call all installers to get all components and add them to the tree.xml
SYNC inventory
alt [Network connected]
DM Session
Notify DM Started
alt [DN finished successfully]
DM completed
Notify DM completed {DMA_MSG_DM_COMPLETED_UI}
alt [No new update available]
Notify error {DMA_MSG_DM_ERROR_UI}
Notify end of flow {DMA_MSG_SCOMO_FLOW_END_UI}
[new update available]
start downloading
[DM failed]
Network interrupt
Notify DM Aborted {DMA_MSG_DM_ABORTED_UI}
[Rule engine conditions not met {B2B_CONDITIONS_EVENT}]
Notify conditions error {DMA_MSG_DM_ERROR_UI}
[No DM account or flow in progress or network disconnected]
Notify error {DMA_MSG_DM_ERROR_UI}

Figure B-2: DM Session Flow

The flow is as follows:

1. The user initiates a DM session.
   - If no DM account exists, the network is not available, or another flow is in progress, the business logic notifies the DIL that there is a DM session error (`DMA_MSG_DM_ERROR_UI`).
   - If the Rule Engine conditions are not met, or another flow is in progress, the business logic notifies the DIL that there is a Rule Engine conditions error (`DMA_MSG_DM_ERROR_UI`).
2. The business logic starts a DM session.
3. The business logic notifies the DIL that a DM session was triggered in response to the user’s request (`DMA_MSG_USER_SESSION_TRIGGERED`).
4. The business logic removes all inventory nodes from the DM Tree, and adds all current inventory to the DM Tree.

   This process is quick, but if the DIL does send an event to the business logic during this process, the event is ignored.
5. The business logic starts a DM session with the Software Management Server.
6. The business logic notifies the DIL that a DM session started.
   - If a network interrupt occurs during communication with the Software Management Server, the business logic notifies the DIL that the DM session was aborted (`DMA_MSG_DM_ABORTED_UI`).
7. When the DM communication with the Software Management Server completes, the business logic notifies the DIL that the DM session completed successfully (`DMA_MSG_DM_COMPLETED_UI`).
8. If no new update is available in the Software Management Server, the business logic notifies the DIL that no update is available (`DMA_MSG_DM_ERROR_UI`).
   - If a new update is available in the Software Management Server, the business logic does one of the following:
     - If the session is “critical” or is “silent”, the business logic starts to download the new update.
     - Otherwise, the user receives a “Confirm download” event (DMA_MSG_SCOMO_DL_CONFIRM_UI).
9. The business logic notifies the DIL that the DM session flow has ended (`DMA_MSG_SCOMO_FLOW_END_UI`); the download flow then begins.

B.3 Download DD Session Flow

This flow is the first part of the Download session flow; it describes downloading the DD.

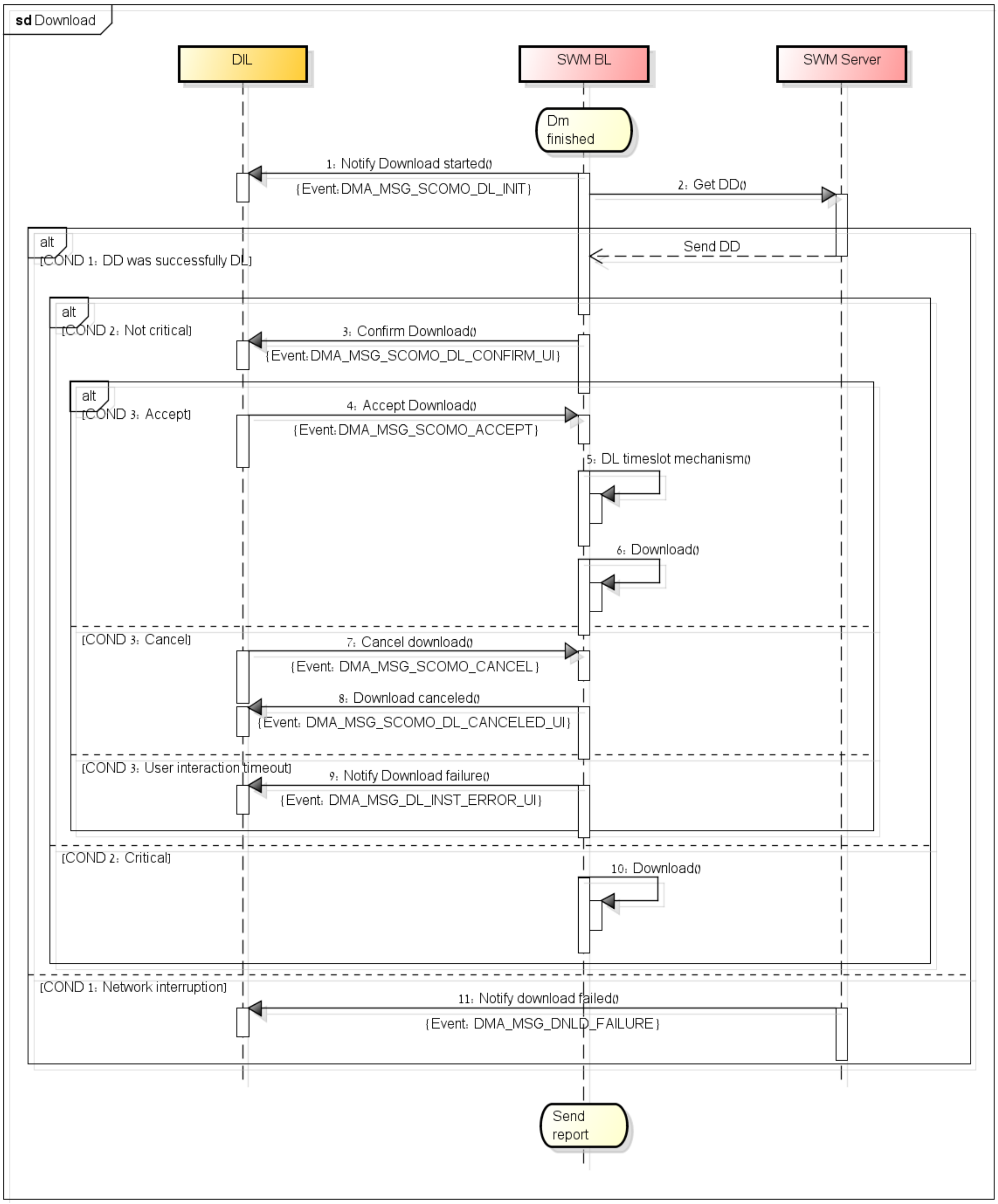

sd Download
DIL
SWM BL
SWM Server
Dm finished
1: Notify Download started()
{Event:DMA_MSG_SCOMO_DL_INIT}
2: Get DD()
Send DD
alt
[COND 1: DD was successfully DL]
alt
[COND 2: Not critical]
3: Confirm Download()
{Event:DMA_MSG_SCOMO_DL_CONFIRM_UI}
alt
[COND 3: Accept]
4: Accept Download()
{Event:DMA_MSG_SCOMO_ACCEPT}
5: DL timeslot mechanism()
6: Download()
[COND 3: Cancel]
7: Cancel download()
{Event: DMA_MSG_SCOMO_CANCEL}
8: Download canceled()
{Event: DMA_MSG_SCOMO_DL_CANCELED_UI}
[COND 3: User interaction timeout]
9: Notify Download failure()
{Event: DMA_MSG_DL_INST_ERROR_UI}
[COND 2: Critical]
10: Download()
[COND 1: Network interruption]
11: Notify download failed()
{Event: DMA_MSG_DNLD_FAILURE}
Send report

Figure B-3: Download Session Flow, Part 1

The flow is as follows:

1 The business logic notifies the DIL that the download has initialized (`DMA_MSG_SCOMO_DL_INIT`).

2 The business logic downloads the DD from the Software Management Server.

COND 1: If the DD was successfully downloaded and the DD is valid…

COND 2: If this is not a critical campaign…

3 The business logic requests the DIL to request that the end-user to confirm the download (`DMA_MSG_SCOMO_DL_CONFIRM_UI`).

COND 3: The end-user accepts the download…

4 The end-user accepts the download. The DIL passes this information to the business logic. (`DMA_MSG_SCOMO_ACCEPT`).

5 The business logic checks the download timeslot algorithm and decides that the download can proceed.

6 The business logic proceeds with the download.

COND 3: The end-user rejects the download…

7 The end-user cancels the download. The DIL passes this information to the business logic. (`DMA_MSG_SCOMO_CANCEL`).

8 The business logic notifies the DIL that the end-user canceled the download (`DMA_MSG_SCOMO_DL_CANCELED_UI`).

COND 3: The end-user does not respond (times out)…

9 The business logic notifies the DIL that the download failed (`DMA_MSG_DL_INST_ERROR_UI`).

COND 2: If this is a critical campaign…

10 The business logic proceeds with the download.

COND 1: If the DD was not successfully downloaded…

11 The business logic notifies the DIL that the download failed (`DMA_MSG_DNLD_FAILURE`).

B.4 Download DP Session Flow

This flow is the second part of the Download session flow; it describes downloading the DP.

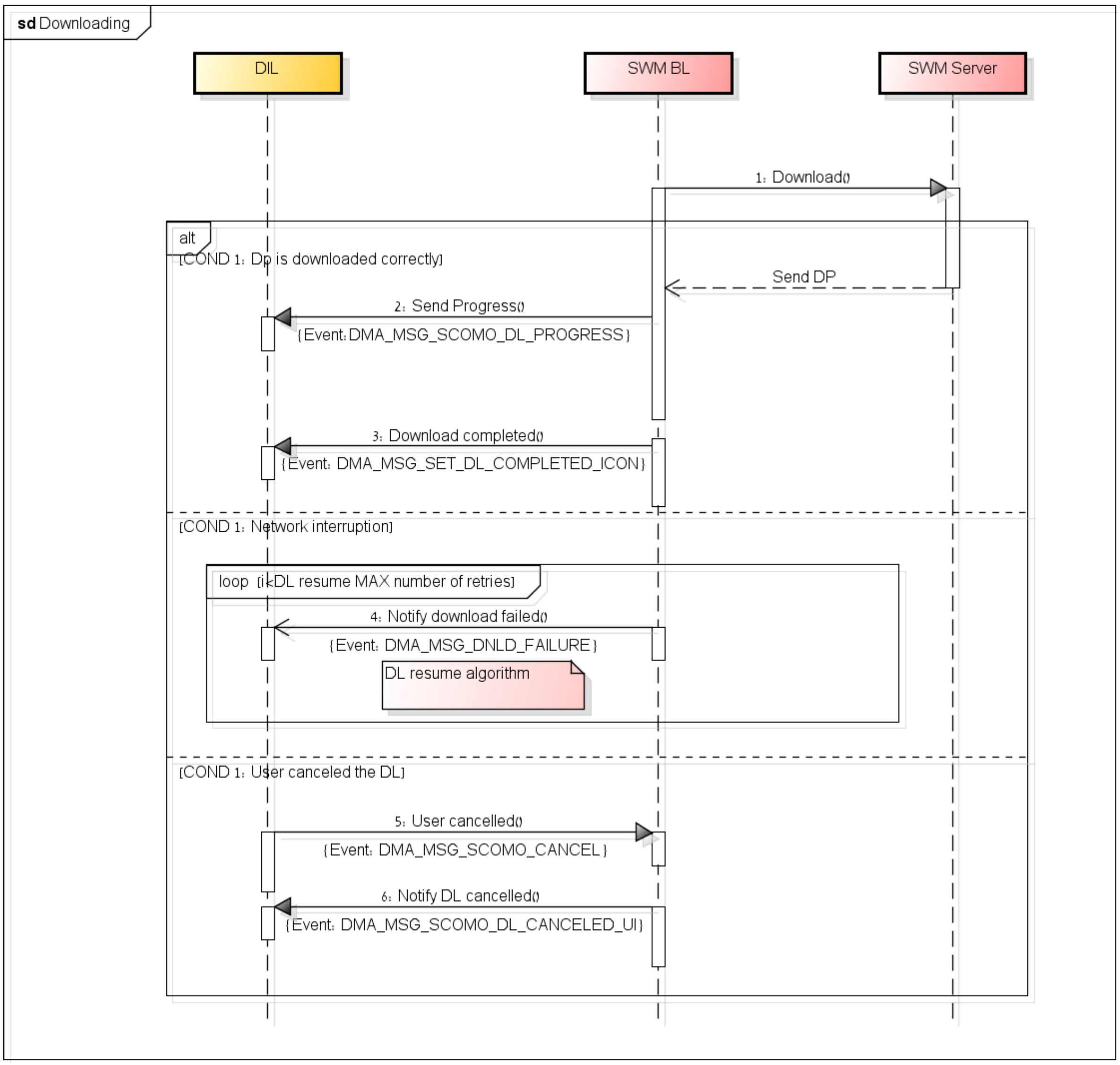

sd Downloading
DIL
SWM BL
SWM Server
1: Download()
alt
[COND 1: Dp is downloaded correctly]
Send DP
2: Send Progress()
{Event:DMA_MSG_SCOMO_DL_PROGRESS}
3: Download completed()
{Event: DMA_MSG_SET_DL_COMPLETED_ICON}
[COND 1: Network interruption]
loop [i<DL resume MAX number of retries]
4: Notify download failed()
{Event: DMA_MSG_DNLD_FAILURE}
DL resume algorithm
[COND 1: User canceled the DL]
5: User cancelled()
{Event: DMA_MSG_SCOMO_CANCEL}
6: Notify DL cancelled()
{Event: DMA_MSG_SCOMO_DL_CANCELED_UI}

Figure B-4: Download Session Flow, Part 2

The flow is as follows:

1 The DP is downloaded from the Download Server.

COND 1: The DP was successfully downloaded…

2 Notify the DIL with the download progress while the download is underway.

3 Notify the DIL once the download is finished.

COND 1: The DP was not successfully downloaded…

4 Notify the DIL that the download failed, and start the download resume process.

COND 1: The end-user cancels the download…

5 The end-user cancels the download.

6 Notify the DIL that the download was canceled.

B.5 Installation Flow

This diagram presents the overall installation flow. Detailed flows for the various stages of the installation are provided in following sections.

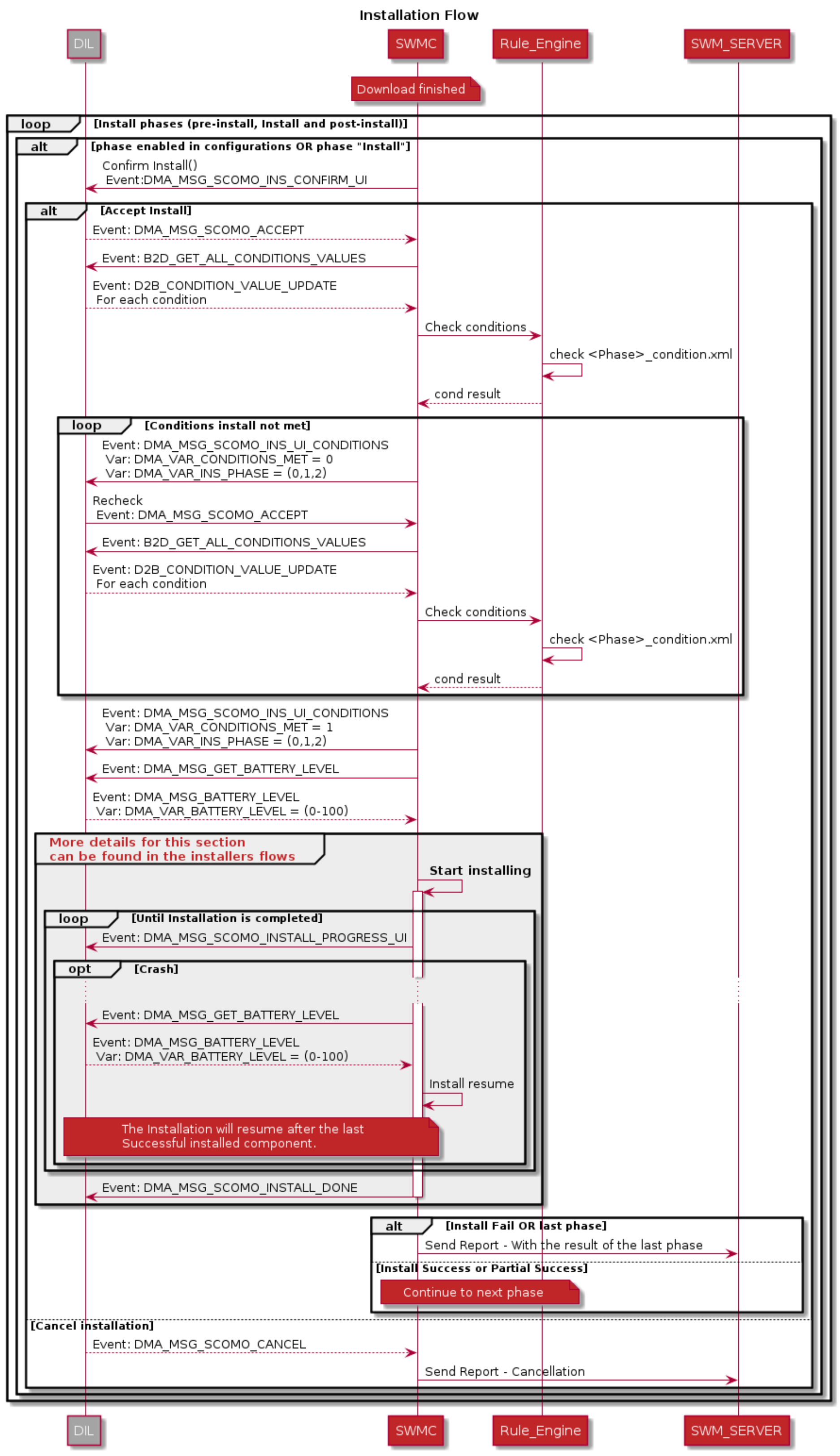

Installation Flow
DIL
SWMC
Rule_Engine
SWM_SERVER
Download finished
loop [Install phases (pre-install, Install and post-install)]
alt [phase enabled in configurations OR phase "Install"]
Confirm Install()
Event:DMA_MSG_SCOMO_INS_CONFIRM_UI
alt [Accept Install]
Event: DMA_MSG_SCOMO_ACCEPT
Event: B2D_GET_ALL_CONDITIONS_VALUES
Event: D2B_CONDITION_VALUE_UPDATE
For each condition
Check conditions
check <Phase>_condition.xml
cond result
loop [Conditions install not met]
Event: DMA_MSG_SCOMO_INS_UI_CONDITIONS
Var: DMA_VAR_CONDITIONS_MET = 0
Var: DMA_VAR_INS_PHASE = (0,1,2)
Recheck
Event: DMA_MSG_SCOMO_ACCEPT
Event: B2D_GET_ALL_CONDITIONS_VALUES
Event: D2B_CONDITION_VALUE_UPDATE
For each condition
Check conditions
check <Phase>_condition.xml
cond result
Event: DMA_MSG_SCOMO_INS_UI_CONDITIONS
Var: DMA_VAR_CONDITIONS_MET = 1
Var: DMA_VAR_INS_PHASE = (0,1,2)
Event: DMA_MSG_GET_BATTERY_LEVEL
Event: DMA_MSG_BATTERY_LEVEL
Var: DMA_VAR_BATTERY_LEVEL = (0-100)
More details for this section
can be found in the installers flows
Start installing
loop [Until Installation is completed]
Event: DMA_MSG_SCOMO_INSTALL_PROGRESS_UI
opt [Crash]
Event: DMA_MSG_GET_BATTERY_LEVEL
Event: DMA_MSG_BATTERY_LEVEL
Var: DMA_VAR_BATTERY_LEVEL = (0-100)
Install resume
The Installation will resume after the last
Successful installed component.
Event: DMA_MSG_SCOMO_INSTALL_DONE
alt [Install Fail OR last phase]
Send Report - With the result of the last phase
[Install Success or Partial Success]
Continue to next phase
[Cancel installation]
Event: DMA_MSG_SCOMO_CANCEL
Send Report - Cancellation
DIL
SWMC
Rule_Engine
SWM_SERVER

Figure B-5: Installation Flow

B.6 Update Agent Handoff Flow

This flow describes how the Software Management DM Client hands off to, and reads results from, the Update Agent. The process requires you to implement the two Porting Layer functions described in UA Porting Layer Functions.

In the diagram, SWMC is the Software Management DM Client, which contains the DIL and the *engine* (the Business Logic Layer and vDM Extended Framework).

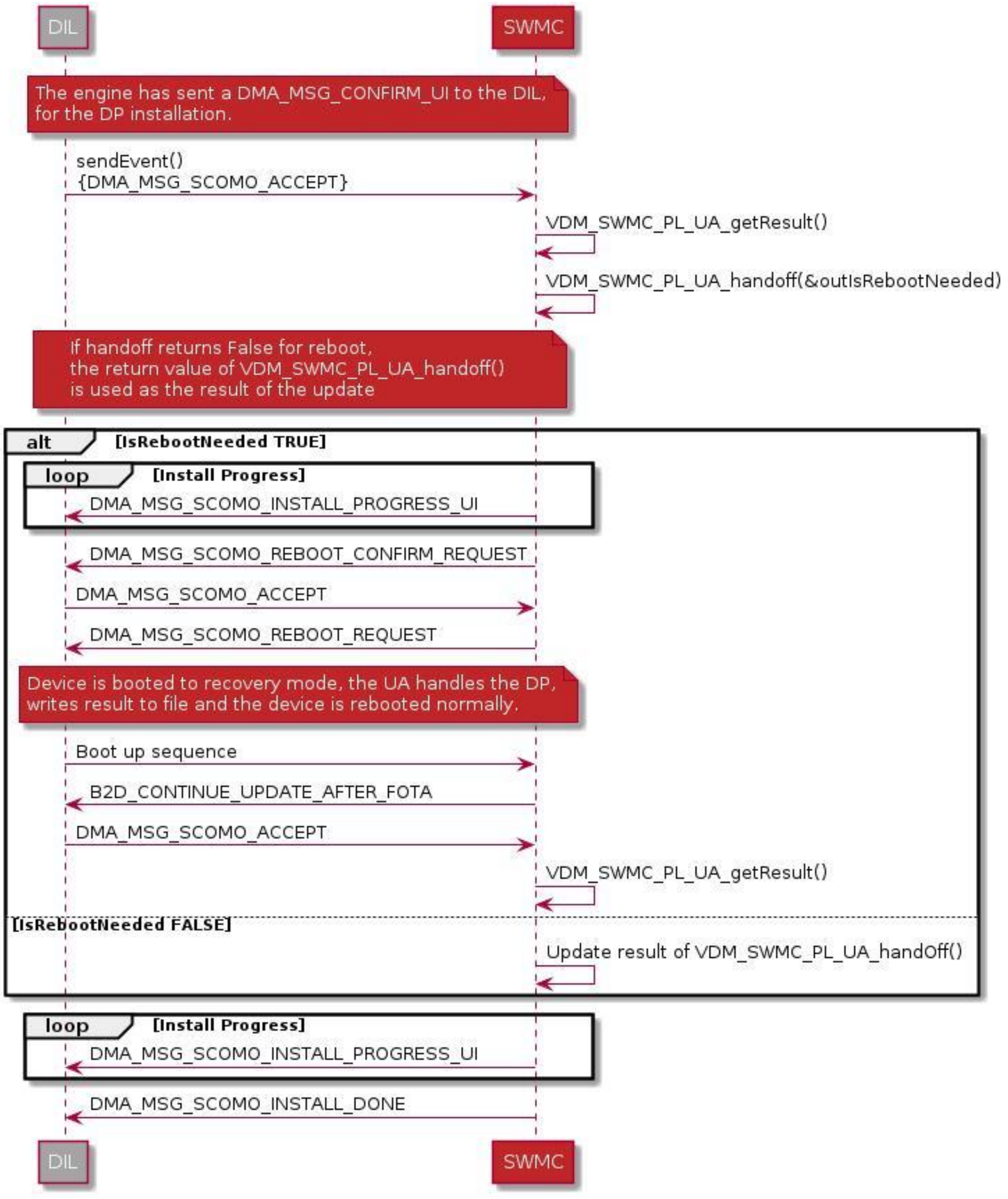

DIL
SWMC
The engine has sent a DMA_MSG_CONFIRM_UI to the DIL, for the DP installation.
sendEvent() {DMA_MSG_SCOMO_ACCEPT}
VDM_SWMC_PL_UA_getResult()
VDM_SWMC_PL_UA_handoff(&outIsRebootNeeded)
If handoff returns False for reboot, the return value of VDM_SWMC_PL_UA_handoff() is used as the result of the update
alt [IsRebootNeeded TRUE]
loop [Install Progress]
DMA_MSG_SCOMO_INSTALL_PROGRESS_UI
DMA_MSG_SCOMO_REBOOT_CONFIRM_REQUEST
DMA_MSG_SCOMO_ACCEPT
DMA_MSG_SCOMO_REBOOT_REQUEST
Device is booted to recovery mode, the UA handles the DP, writes result to file and the device is rebooted normally.
Boot up sequence
B2D_CONTINUE_UPDATE_AFTER_FOTA
DMA_MSG_SCOMO_ACCEPT
VDM_SWMC_PL_UA_getResult()
[IsRebootNeeded FALSE]
Update result of VDM_SWMC_PL_UA_handoff()
loop [Install Progress]
DMA_MSG_SCOMO_INSTALL_PROGRESS_UI
DMA_MSG_SCOMO_INSTALL_DONE
DIL
SWMC

Figure B-6: Update Agent Handoff Flow

The flow is as follows:

1 After the end-user confirms the installation, the DIL sends the Business Logic event `DMA_MSG_SCOMO_ACCEPT` to the Business Logic Layer.

2 The engine checks if the installation was already performed by looking for a result file (`VDM_SWMC_PL_UA_getResult`).

3 If a result file is not found, the engine writes information to a handoff directory, so that the Update Agent can read the information, and invokes the Update Agent (`VDM_SWMC_PL_UA_handoff`). The information includes the path to the downloaded package to install.

4 While the installation proceeds, the engine provides progress information to the DIL by repeatedly sending the DIL event `DMA_MSG_SCOMO_INSTALL_PROGRESS_UI`. Since at least one software instance that requires handoff is in the DP, only one `DMA_MSG_SCOMO_INSTALL_PROGRESS_UI` is transmitted with a progress value of zero.

5 The engine sends the DIL event `DMA_MSG_SCOMO_REBOOT_CONFIRM_REQUEST` to the DIL asking the end-user to confirm the reboot to recovery mode to complete the installation (in silent mode the DIL accepts immediately).

6 The DIL accepts the reboot by sending `DMA_MSG_SCOMO_ACCEPT` (or rejects the installation session and sends an installation failure report by sending `DMA_MSG_SCOMO_CANCEL`).

7 The engine sends the DIL event `DMA_MSG_SCOMO_REBOOT_REQUEST` to the DIL asking it to reboot to recovery mode to complete the installation.

8 After completing the installation, the device reboots again into standard mode and performs the boot flow (see Boot Flow).

   a The Business Logic Layer sends the DIL event `B2D_CONTINUE_UPDATE_AFTER_FOTA` to the DIL. The Business Logic Layer continues with the installation after it receives `DMA_MSG_SCOMO_ACCEPT` from the DIL.

   b The engine knows that the device is in the middle of an installation, so the engine checks for the results of the installation (`VDM_SWMC_PL_UA_getResult`).

   c If there is additional installation to complete, the engine provides progress information to the DIL by repeatedly sending the DIL event `DMA_MSG_SCOMO_INSTALL_PROGRESS_UI`.

   d When the installation is complete, the engine sends the DIL event `DMA_MSG_SCOMO_INSTALL_DONE` to the DIL.

B.7 DP Flow

These flows describe the handling of large and small DPs. For more information, see Handling Large DPs.

B.7.1 DP Download Flow

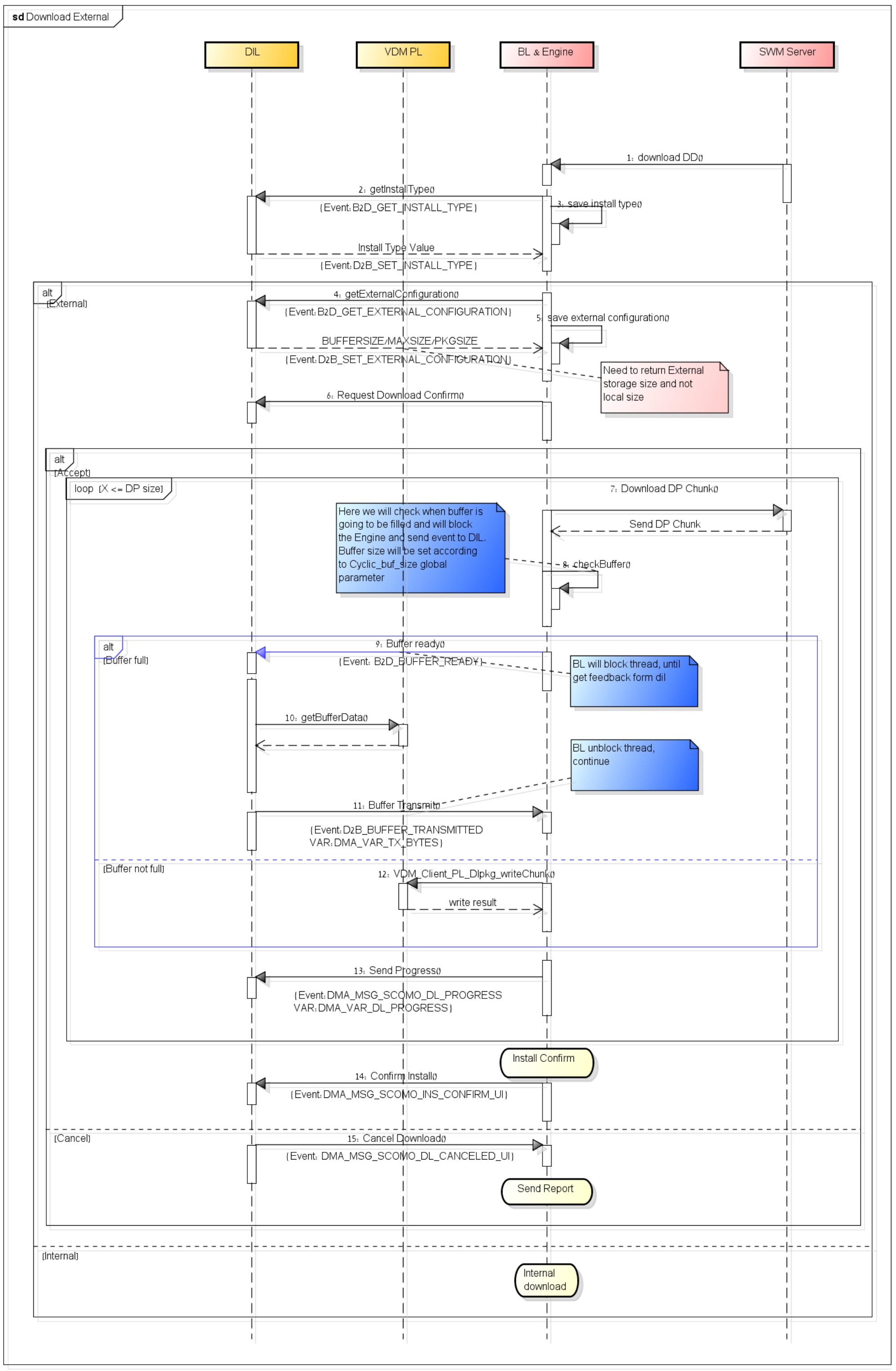

sd Download External
DIL
VDM PL
BL & Engine
SWM Server
1: download DD()
2: getInstallType()
{Event: B2D_GET_INSTALL_TYPE}
3: save install type()
Install Type Value
{Event: D2B_SET_INSTALL_TYPE}
alt
[External]
4: getExternalConfiguration()
{Event: B2D_GET_EXTERNAL_CONFIGURATION}
5: save external configuration()
BUFFERSIZE/MAXSIZE/PKGSIZE
{Event: D2B_SET_EXTERNAL_CONFIGURATION}
Need to return External storage size and not local size
6: Request Download Confirm()
alt
[Accept]
loop [X <= DP size]
7: Download DP Chunk()
Send DP Chunk
Here we will check when buffer is going to be filled and will block the Engine and send event to DIL. Buffer size will be set according to Cyclic_buf_size global parameter
8: checkBuffer()
alt
[Buffer full]
9: Buffer ready()
{Event: B2D_BUFFER_READY}
BL will block thread, until get feedback form dil
10: getBufferData()
BL unblock thread, continue
11: Buffer Transmit()
{Event: D2B_BUFFER_TRANSMITTED VAR: DMA_VAR_TX_BYTES}
[Buffer not full]
12: VDM_Client_PL_Dlpkg_writeChunk()
write result
13: Send Progress()
{Event: DMA_MSG_SCOMO_DL_PROGRESS VAR: DMA_VAR_DL_PROGRESS}
Install Confirm
14: Confirm Install()
{Event: DMA_MSG_SCOMO_INS_CONFIRM_UI}
[Cancel]
15: Cancel Download()
{Event: DMA_MSG_SCOMO_DL_CANCELED_UI}
Send Report
[Internal]
Internal download

Figure B-7: Large DP Download Flow

B.7.2 DP Installation Flow

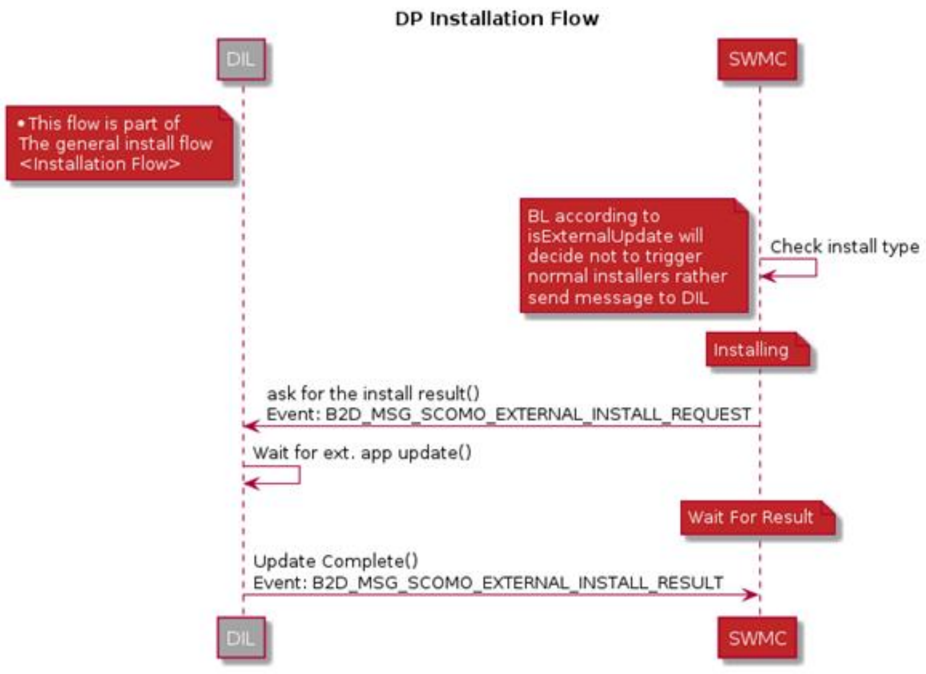

DP Installation Flow
DIL
SWMC
• This flow is part of The general install flow <Installation Flow>
BL according to isExternalUpdate will decide not to trigger normal installers rather send message to DIL
Check install type
Installing
ask for the install result()
Event: B2D_MSG_SCOMO_EXTERNAL_INSTALL_REQUEST
Wait for ext. app update()
Wait For Result
Update Complete()
Event: B2D_MSG_SCOMO_EXTERNAL_INSTALL_RESULT
DIL
SWMC

Figure B-8: DP Installation Flow

B.8 Boot Flow

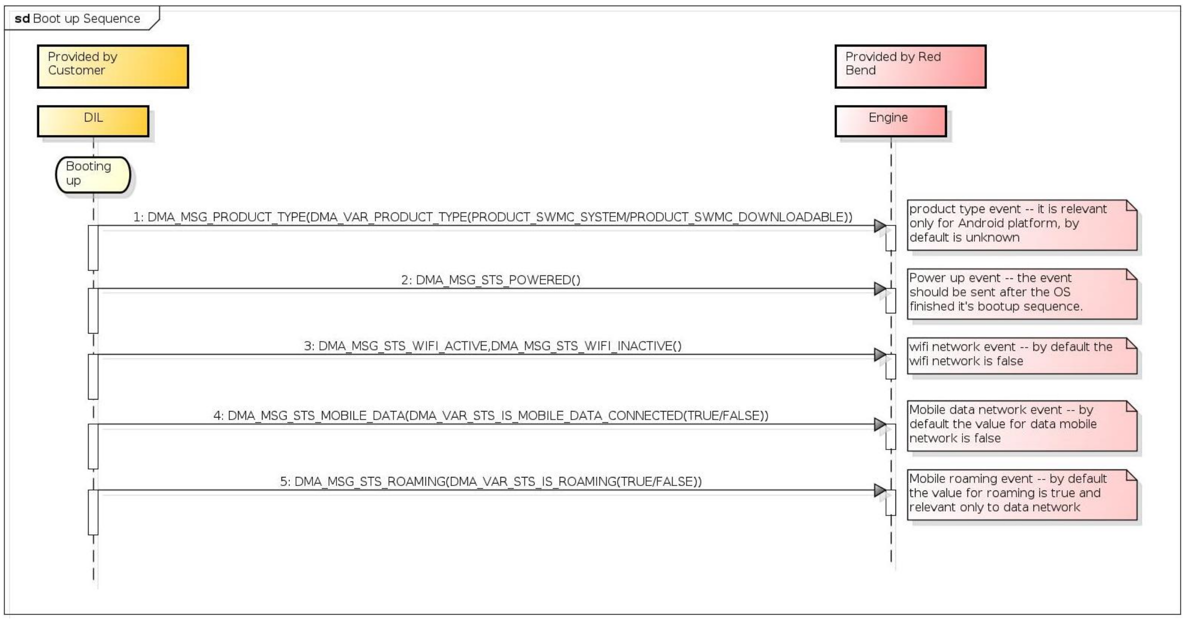

sd Boot up Sequence
Provided by Customer
Provided by Red Bend
DIL
Engine
Booting up
1: DMA_MSG_PRODUCT_TYPE(DMA_VAR_PRODUCT_TYPE(PRODUCT_SWMC_SYSTEM/PRODUCT_SWMC_DOWNLOADABLE))
product type event -- it is relevant only for Android platform, by default is unknown
2: DMA_MSG_STS_POWERED()
Power up event -- the event should be sent after the OS finished it's bootup sequence.
3: DMA_MSG_STS_WIFI_ACTIVE,DMA_MSG_STS_WIFI_INACTIVE()
wifi network event -- by default the wifi network is false
4: DMA_MSG_STS_MOBILE_DATA(DMA_VAR_STS_IS_MOBILE_DATA_CONNECTED(TRUE/FALSE))
Mobile data network event -- by default the value for data mobile network is false
5: DMA_MSG_STS_ROAMING(DMA_VAR_STS_IS_ROAMING(TRUE/FALSE))
Mobile roaming event -- by default the value for roaming is true and relevant only to data network

Figure B-9: Boot Flow

During boot the DIL must send events to update the business logic with the current status:

- A product type event: Describes if the SWM Client is installed in the data or integrated into the system partition. (This is relevant for Android only.)
- A power event: Sent when the boot of the OS is finished.

- Wi-Fi event: Updates the business logic if Wi-Fi is available or not available.
- Mobile data event: Sent whether available or not.
- Roaming event: Sent if data is roaming.

Appendix C Session Tracking

Session Tracking enables System Administrators to the Software Management Server UI to track the current status of each update for specific devices and to see the current stage of each update.

C.1 Status Notification

The Client notifies the Server of the condition of the update session by sending its current status.

- **Current state** of the update session
- **Timestamp** & **Track ID** are included for any state sent to the server.
  Note: Timestamp is sent as UTC, in milliseconds.
- For some statuses, additional information is sent as part of the reported status. See `status.tracking.param` in the Status for more information.

C.1.1 Precondition

In order to use the tracking feature, **dma_config** must include the following parameters:

- enable_concurrent_dm_dl=TRUE
- ststrk.active_statuses=<"none" or "basic" or "full" ±additional statuses>
- ststrk.report_interval_sec=<seconds>
- By default, status update sessions occur every 30 seconds (only the most recent status is reported).

Note: If the leaf `MinReportInterval` exists in tree.xml, the **dma_config** parameter `ststrk.report_interval_sec` is not used; the value set in leaf `MinReportInterval` will be used instead.

C.1.2 Sending Statuses in Bulk

The Client can send a report/alert containing statuses in bulk (a list of items in the generic alert). The Client by default can aggregate up to 100 items in a single report/alert. When more than 100 statuses must be sent in bulk, separate reports, each containing 100 statuses are sent.

Example:
```
<Alert>
<Alert>
    <CmdID>3</CmdID>
    <Data>1226</Data>
    <Item>
      <Meta><Format xmlns='syncml:metinf'>text/plain</Format>
      <Type>xmlns='syncml:metinf'>com.redbend.StatusTracking</Type>
      <Mark>informational</Mark></Meta>
      <Data>{
         "statusName": "status.tracking.download.started",
         "0": "status.tracking.param.bearer.type.Wi-Fi",
         "timeStamp": "1434547213078",
         "timeZone": \"GMT+03:00\",
         "trackId": "345673467345634 56"
      }
      </Data>
   </Item>
   <Item>
      <Meta><Format xmlns='syncml:metinf'>text/plain</Format>
      <Type>xmlns='syncml:metinf'>com.redbend.StatusTracking</Type>
      <Mark>informational</Mark></Meta>
      <Data>{
         "statusName": "status.tracking.download.completed",
         "0": "status.tracking.param.bearer.type.Wi-Fi",
         "timeStamp": "1434547213078",
         "timeZone": \"GMT+03:00\",
         "trackId": "345673467345634 56"
      }
      </Data>
   </Item>
 </Alert>
```

C.1.3 Configuring Report Size

The default alert/report size is 32K. To change the size (and hence the number of statuses that the alert/report can hold), change the value (in bytes) of the parameter `maxmsgsize` in **dma_config.txt**.

C.1.4 Configuring a different time interval

Use either ststrk.report_interval_sec in **dma_config** —or— the leaf `MinReportInterval` in tree.xml to change the time interval.

dma_config

Change the value of the parameter `ststrk.report_interval_sec=<seconds>` in the file **dma_config.txt**, where `<seconds>` is the number of seconds required. The default number of seconds is 30.

Note: If the leaf `MinReportInterval` exists in tree.xml, the **dma_config** parameter ststrk.report_interval_sec is not used; the value set in leaf `MinReportInterval` will be used instead.

Tree.xml

To set the report interval using **tree.xml**, use the leaf `MinReportInterval, as` shown below.

```
      <node>
        <name>StatusTracking</name>
        <leaf>
```

```
          <name>MinReportInterval</name>
          <copy/><delete/><exec/><get/><replace/>
          <format>int</format>
          <value>30</value>
        </leaf>
        <leaf>
          <name>TrackID</name>
          <get/><replace/>
          <format>chr</format>
          <value>123456789</value>
        </leaf>
      </node>
```

C.1.5 Reporting all Statuses

To report all statuses immediately:

- Set the parameter **MinReportInterval** to 0 seconds in the **tree.xml**.

  or

- Set the parameter `ststrk.report_interval_sec=0` in dma_config.txt.

NOTE: if the node parameter `./Ext/RedBend/`**StatusTracking/MinReportInterval** exists in the tree, the value in the tree will be used, **not** the value (`ststrk.report_interval_sec`) set in **dma_config.txt.**

C.2 Statuses and Details

The client notifies the server of its update stage using the statuses listed in this section.

Note: DL related statuses refer to the download of the package (not the download of a DD).

DL Statuses

a **DL Canceled (Rejected)**

Message key = status.tracking.download.cancelled
Message parameter key = one of the following:

Parameter Key	Description
• status.tracking.param.dl.cancel.cancelled.by.user	Download cancelled by user.
• status.tracking.param.dl.cancel.cancelled.by.server	Download cancelled by server.
• status.tracking.param.dl.cancel.restarted.by.user	Download cancelled due to user's request to restart it.
• status.tracking.param.dl.cancel.user.interaction.timeout	Download cancelled due to user interaction timeout while waiting for download confirmation.
• status.tracking.param.dl.cancel.external.download.timeout	Timeout has occurred while waiting for external download completion
• status.tracking.param.dl.cancel.external.download.timeout	External download timeout (third party download timeout).

b **DL Completed**

Message key = status.tracking.download.deferred
Includes Timestamp, Timezone and Track ID (see Full List of Statuses – Common Parameters).

c **DL Deferred**

Message key = status.tracking.download.completed

d **DL Failed**

Message key = status.tracking.param.dl.failed.max.retry.reached
Includes Timestamp, Timezone and Track ID (see Full List of Statuses – Common Parameters)).

e **DL Interrupted**

Message key = status.tracking.download.interrupted
Message parameter key = one of the following:

Parameter Key	Description
• status.tracking.param.error.type.connectivity.error	Connectivity error.
• status.tracking.param.error.type.paused.by.user	Paused by user.
• status.tracking.param.error.type.download.limitation.reached	Concurrent download limit (Clients) reached.
• status.tracking.param.error.type.time.window.limitation	Reached download limit for timeframe.
• status.tracking.param.error.type.unspecific.error	Unspecific error.

f **DL Progress**

Message key = one of the following

Parameter Key	Description
• status.tracking.download.progress "0": "25"	25% downloaded.
• status.tracking.download.progress "0": "50"	50% downloaded.
• status.tracking.download.progress "0": "75"	75% downloaded.

g **DL Resumed**

Message key = status.tracking.download.resumed
session.tracking.param.bearer.type.<bearer value – free string>

h **DL Started**

Message key = status.tracking.download.started
Message parameter key = one of the following:

Parameter Key	Description
• status.tracking.param.bearer.type.2G	2G
• status.tracking.param.bearer.type.3G	3G
• status.tracking.param.bearer.type.Wi-Fi	Wi-Fi
• status.tracking.param.bearer.type.Bluetooth	Bluetooth
• status.tracking.param.bearer.type.NFC	NFC
• status.tracking.param.bearer.type.defaultBearer	Default

i **Waiting for DL Confirm**

Message key = status.tracking.wait.for.download.confirm

DM Statuses

j **DM Started**

Message key = status.tracking.dm.started
Note: This status is sent for every DM (Server, Device, User initiated) except the DM used for status reporting and report sending.

k **Push Notification Received (nia received)**

Parameter Key	Description
• status.tracking.nia.received	Server initiated DM session.

Install Statuses

l **Install Cancelled (Rejected)**

Message key = status.tracking.install.cancelled
Note: Includes the installation *phase*.

m **Install Completed**

Message key = status.tracking.install.completed
Note: Includes the installation *phase*.

n **Install Component X Started**

Message key = status.tracking.install.component.started

Message parameter key = one of the following:

Parameter Key	Description
• param 0: Install phase: status.tracking.param.install.component.started • param 1: <component name>	During install phase.

o **Install Component X Completed (+Result)**

Message key = status.tracking.install.component.completed
Message parameter key = one of the following:

Parameter Key	Description
• param 0: Install phase: status.tracking.param.install.component.complete d • param 1: <component name> • Param 2: <installation result>	During install phase.

p **Install Component X Completed (+ result)**

Message key = status.tracking.install.component.completed

q **Install Deferred**

Message key = status.tracking.install.deferred

r **Install Started**

Message key = status.tracking.install.started
Message parameter key = one of the following:

Parameter Key	Description
• status.tracking.param.install.phase.pre.install.phas e	During pre-install phase.
• status.tracking.param.install.phase.install.phase	During install phase.
• status.tracking.param.install.phase.post.install.pha se	During post-install phase.
• status.tracking.param.install.phase.not.initialized	When installation phase is not initialized.

s **Waiting for User Install Confirm**

Message key = status.tracking.wait.for.install.confirm

C.2.1 Full List of Statuses

The full list of statuses is given below.

Note: The names of the presets and all individual statuses are NOT case sensitive.

DL_Cancelled

DL_Completed	Basic
DL_Deferred	
DL_Failed	Basic
DL_Interrupted	
DL_Progress	
DL_Resumed	
DL_Started	Basic
DM Push Notification Received	
DM_Started	Basic
INST_Cancelled	
INST_Completed	Basic
INST_Completed (+result)	Basic
INST_Comp_Completed (+result)	
INST_Deferred	
INST_Started	Basic
PUSH_Notif_Received	
Wait_For_INST_Confirm	
Wait_For_DL_Confirm	

C.2.2 Common Parameters

These parameters are common to more than one status.

Status	Description
timeStamp	The time, in milliseconds, from 1 January 1980.
timeZone	The GMT time and the offset to the current location.
trackId	A unique ID, determined by the Server, to identify each process, device, etc.

C.3 Typical Generic Alert Data Structure

Status **reports** are transmitted as generic alerts with multiple items. There is one item for each status. In addition, we can send several items in a single generic alert (to a maximum of 100 items per alert/session). The general alert data structure is given here.

```
<Alert>
    <CmdID>3</CmdID>
        <Data>1226</Data>
    <Item>
        <Meta>
          <Format xmlns='syncml:metinf'>text/plain</Format>
          <Type xmlns='syncml:metinf'com.redbend.StatusTracking
          </Type>
          <Mark>informational</Mark>
        </Meta>
        <Data>“See data element format below”</Data>
    </Item>
    <Item>
        <Meta>
          <Format xmlns='syncml:metinf'>text/plain</Format>
          <Type xmlns='syncml:metinf'com.redbend.StatusTracking
          </Type>
          <Mark>informational</Mark>
        </Meta>
        <Data>“See data element format below”</Data>
    </Item>
</Alert>
```

C.3.1 <data> Element Format

Each <data> element in the alert contains the status data formatted as a JSON object.

Example 1:

```
{
    "statusName": "status.tracking.download.started",
    "0":“status.tracking.param.bearer.type.3G”,
    "timeStamp": "1434547212800",
    "timeZone": \"GMT+03:00\",
    "trackId": "5848384756573482561111"
}
```

Example 2:

```
      "statusName": "status.tracking.download.progress",
        "0": "25",
      "timeStamp": "1434547212900",
      "timeZone": \"GMT+03:00\",
      "trackId": "5848384756573482561111"
```

Example 3:

```
{
    "statusName": "status.tracking.download.completed",
    "0": "status.tracking.param.bearer.type.3G",
    "timeStamp": "1434547213078",
    "timeZone": \"GMT+03:00\",
    "trackId": "5848384756573482561111"
 }
```

Appendix D Get Client Logs

Client log uploads can be initiated by the user via the UI.

Appendix E Software Management Client as a Library

You can build the Software Management Client as a library in order to include the Software Management Client in your own Android application. To do this, you must perform the following additional installation and configuration tasks. For more information about the Android Library Project, see http://developer.android.com/tools/projects/index.html#LibraryProjects.

E.1 Copying the DM Tree

Copy the DM Tree (**tree.xml**) from `assets/` in the delivery package to `assets/` in your application.

E.2 Copying AndroidManifest.xml Items

Copy the relevant items from **AndroidManifest.xml** in the delivery package to the library project **AndroidManifest.xml**.

E.3 Managing Events Outside the Software Management Client

The Software Management Client DIL handles all events from the business logic (DIL events) and sends all required events to the business logic (Business Logic events).

This section presents how to override the Software Management Client DIL if you would like your application to handle some or all of these events.

E.3.1 Handling DIL Events Outside the Software Management Client

To handle DIL events in your application:

1 Create an XML file containing the list of DIL events for your application to handle.

The name of this file must be **externalized_events_list.xml**. Add one line with an `<item>` tag for each DIL event.

```
<?xml version="1.0" encoding="utf-8" ?>
<resources>
   <string-array name="eventsForExternalize">
      <item>DIL_EVENT_1</item>
      <item>DIL_EVENT_2</item>
      <item>...</item>
   </string-array>
</resources>
```

Events listed in this file are sent to your application instead of being handled by the Software Management Client DIL. All other events continue to be handled internally by the Software Management Client DIL.

2 Place this file in `/res/values/`.

3 Create a receiver in your application for `SwmcClient.SwmcDilEvent` intents.

The following code provides an example for registering the receiver in **AndroidManifest.xml**. This code example is in the reference application (see Software Management Client as a Library Reference Application).

```
<receiver android:name="com.example.test.SwmcDilEventReciever"
      android:enabled="true">
   <intent-filter>
      <action android:name="SwmcClient.SwmcDilEvent" />
      <category android:name="com.redbend.client" />
   </intent-filter>
</receiver>
```

E.3.2 Sending BL Events from Outside the Software Management Client

To send Business Logic events use `SmmClient.sendEventToSmmService`. For example:

```
Event event = null;
event = new Event("DMA_MSG_SCOMO_NOTIFY_DL");
sendEventToSmmService(this, ClientService.class, event);
```

SwmcConstants.java contains an example of the Business Logic events that can be sent to the business logic.

E.4 Software Management Client as a Library Reference Application

The delivery package includes the directory `swmc_container_demo/`, which contains a reference Android application that demonstrates how to use the Software Management Client as a library.

The reference application handles the DIL events `DMA_MSG_SCOMO_DL_CONFIRM_UI` and `DMA_MSG_GET_BATTERY_LEVEL`.

For `DMA_MSG_SCOMO_DL_CONFIRM_UI`, if the installation is critical, the reference application responds with `DMA_MSG_SCOMO_ACCEPT`. Otherwise, the reference application responds with `DMA_MSG_SCOMO_CANCEL`.

For `DMA_MSG_GET_BATTERY_LEVEL`, the reference application responds with the battery level using `DMA_MSG_BATTERY_LEVEL`.

The reference application uses **SwmcDilResponse.java** to send Business Logic events by invoking `SmmClient.sendEventToSmmService`.

E.4.1 Running the Reference Application

To run the application:

1 Create **externalized_events_list.xml** as follows (see Handling DIL Events Outside the Software Management Client):

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<resources>
   <string-array name="eventsForExternalize">
      <item>DMA_MSG_SCOMO_DL_CONFIRM_UI</item>
      <item>DMA_MSG_GET_BATTERY_LEVEL</item>
   </string-array>
</resources>
```

2 Rebuild and install the application (see Building the Reference Application).

Critical campaign:

1 On the Software Management Server, create an Android app installation critical campaign.

2 Run the application.

An application screen with one button appears.

3 Click **Get Update**.

The application starts a DM session with the DM Server. The Software Management Client handles all responses with the Server, and sends `DMA_MSG_SCOMO_ACCEPT` when asked to install the Android app. The application sends `DMA_MSG_BATTERY_LEVEL` when prompted for the battery level. The reference application exits when the installation is complete.

Regular campaign:

1 On the Software Management Server, create an Android app installation regular campaign.

2 Run the application.

An application screen with one button appears.

3 Click **Get Update**.

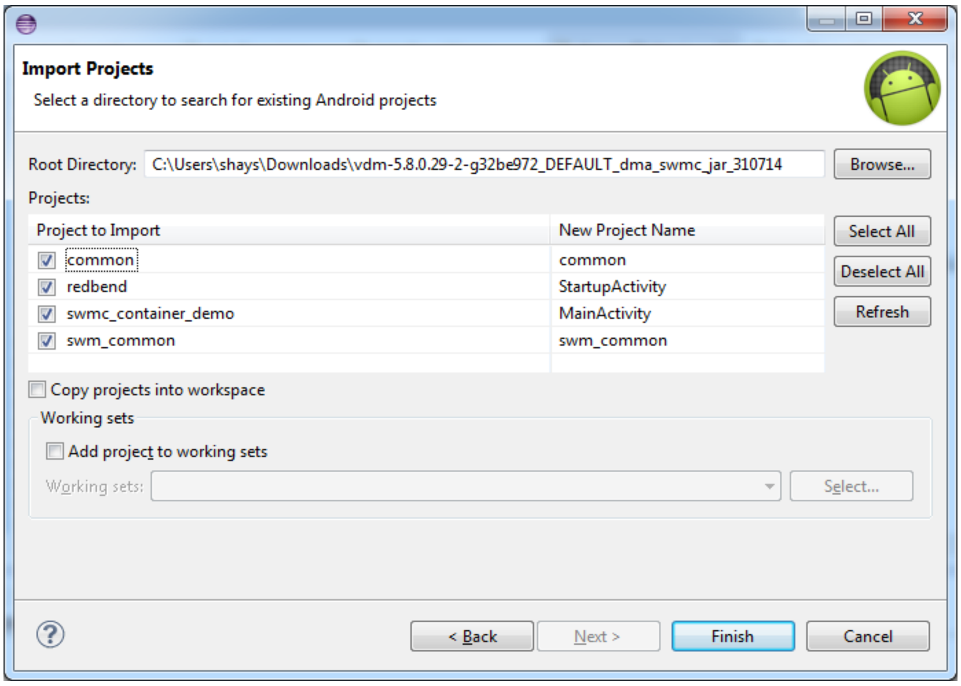
The application starts a DM session with the DM Server. The Software Management Client handles all responses with the Server, and sends `DMA_MSG_SCOMO_CANCEL` when asked to install the Android app. The download does not proceed and the reference application exits.

E.4.2 Building the Reference Application

To build the reference application in Eclipse (Kepler):

1 In Eclipse, select **File** → **Import** → **Android** → **Existing Android Code Into Workspace**.

The **Import Projects** page appears.



Import Projects
Select a directory to search for existing Android projects

Root Directory: C:\Users\shays\Downloads\vdm-5.8.0.29-2-g32be972_DEFAULT_dma_swmc_jar_310714

Projects:

Project to Import	New Project Name
☑ common	common
☑ redbend	StartupActivity
☑ swmc_container_demo	MainActivity
☑ swm_common	swm_common

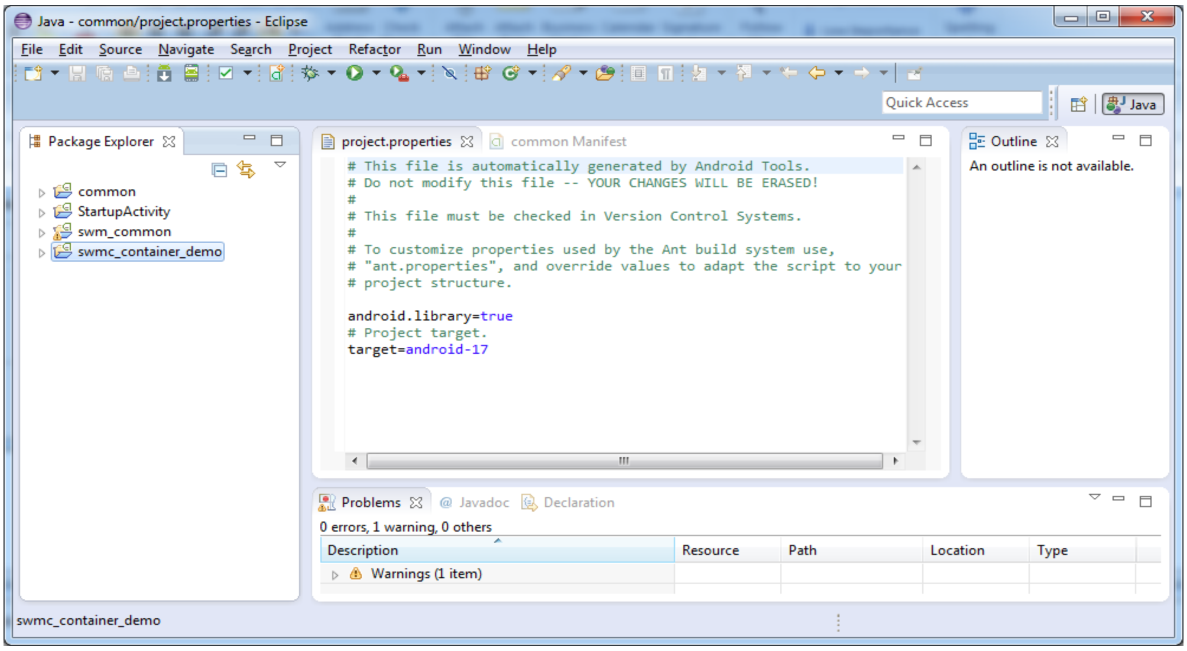
☐ Copy projects into workspace

Working sets
☐ Add project to working sets
Working sets:

2 Browse to the delivery root directory and import the four Android projects to your workspace.

The projects window appears.

3 Edit the code of the Android projects, as required.



Java - common/project.properties - Eclipse

```
# This file is automatically generated by Android Tools.
# Do not modify this file -- YOUR CHANGES WILL BE ERASED!
#
# This file must be checked in Version Control Systems.
#
# To customize properties used by the Ant build system use,
# "ant.properties", and override values to adapt the script to your
# project structure.

android.library=true
# Project target.
target=android-17
```

An outline is not available.

0 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Warnings (1 item)				

swmc_container_demo

4 Run the application: right-click **swmc_container_demo** → **Run As** → **Android Application**.

The application runs.

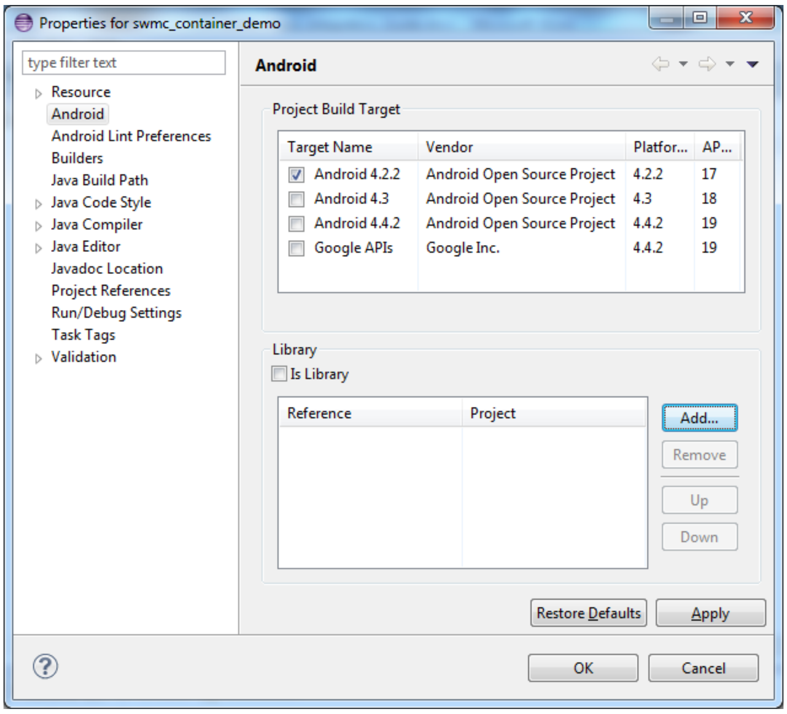
E.4.3 Including the Software Management Client in Your Own Android Application

To include the Software Management Client library into your existing Android application:

1 Import the projects as described in Building the Reference Application.

2 In the Eclipse Packages Explorer menu, right-click your Android project and select **Properties**.

The **Properties** screen appears.



Target Name	Vendor	Platfor...	AP...
✓ Android 4.2.2	Android Open Source Project	4.2.2	17
Android 4.3	Android Open Source Project	4.3	18
Android 4.4.2	Android Open Source Project	4.4.2	19
Google APIs	Google Inc.	4.4.2	19

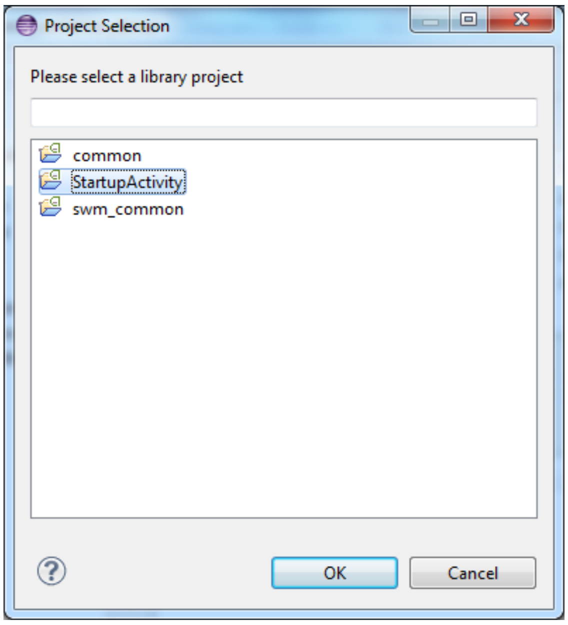
Is Library

Reference	Project

3 Select **Android** in the left menu.

4 Click **Add** in the **Library** pane.

The **Project Selection** screen appears.



5 Select **StartupActivity** and click **OK**.

The Software Management Client library is added to your application project.

Appendix F Authenticating a Self-Certified Server

To authenticate a self-certified server:

1 Get the server's CA (use the following **openssl** command on the device) and save it to a file (for example, save it to: `/tmp/server.ca`).

```
openssl s_client -showcerts -connect www.openssl.org:443  >
/tmp/server.ca
```

2 Edit the file so as to leave only the server's *self-certificate*:

a A certificate is preceded by a line with a string *identifier* used to identify the issuer (preceded by "i:"), and a line with a string identifier used to identify the subject (preceded by "s:"):

(i:/{issuer information})

(s:/{subject information})

b In a server *self-certificate*, the issuer and subject identifiers are the same:

(i:/{specific information})

(s:/{specific information})

c The certificate itself is the block of hashed data between, and including, the lines "Begin Certificate" and "End Certificate". Only the block of text (from the self-certificate) is to remain in the file.

```
-----BEGIN CERTIFICATE-----
HdE4OOz2bUlTIENBMB4XDTE1MDQxNjE0M
DQzNFoXDTE4MDQxNTE0MDQzNFOFCzAJBg
NVBATAkRFMRswGQYDVQQIExJCYWRlbiBX
dWVydHRlbWJlcmcxEjAQBCxMGSVRDL1RP
MRwwGgYDVQQDDBMqLmR2Yi5jb3JwaW50Z
XIubmV0MIIBIjANF62IJitg8NR3sbQ1Dz
-----END CERTIFICATE-----
```

3 Determine the subject hash, e.g., :

```
openssl x509 -hash -in /tmp/server.ca
```

4 The resulting format generally has the form:

```
24ad0b63
```

5 Put the file into the correct location with the correct name:

```
mv /tmp/server.ca/ etc/ssl/certs/24ad0b63
```

Explanation

Object	**Example Used**
Example file location	`/tmp/server.ca`
Example hash format	`24ad0b63`

Shell commands	`openssl s_client -showcerts -connect`
	`openssl x509 -hash -in`

Actual default path	`/etc/ssl/certs`

Appendix G Verifying DP Authenticity

For integration with security functions, verification and authentication of DPs is supported using NDS™ or Escrypt™ security library APIs.

The Software Management Client can verify the integrity and authenticity of DPs downloaded from the server, using the security library APIs located on the Management Unit.

To verify the integrity and authenticity of downloaded DPs:

1. The DP validation check should be done in the pre-install phase.
2. The downloaded file includes the DP itself and a signed SHA-256 hash of the DP. The SWM Client should authenticate the signature of the above SHA-256 hash by using NDS™ or Escrypt™ APIs and the secure *public key* that are stored on the Management Unit.
3. The SWM Client must then calculate the SHA-256 hash value of the downloaded DP and compare it to the signed hash value that was received with the DP in order to verify DP integrity.
4. If either the signature or the hash is not valid, display a message on the HMI and abort the update. In addition, report failure to server.
5. If both the signature and the hash value are verified, a message should be displayed on HMI and the SWM client should proceed to deploy the updates in the DP.

Note: When using Escrypt, signature verification should be performed using Escrypt security library, RSA-2000 functions.

Appendix H Installer Types

An installer type is a number that determines the executable to activate when trying to update, install, or remove software. Each software version is associated with an installer type. Each software version in a DP can be associated with a different installer.

You can create or assign your own installers (called *generic installers*). The Software Management Client will hand off processing software to the associated installer. The complete list of installer types is defined for the Software Management Server in the Customer Features configuration file. The list of valid installer types for each domain is a subset of the complete list. Some values are reserved for future use. Native installer types are indicated as such, where native implies that the OS can independently install the update.

Table H-2: Installer Types

Num	Name	UI Text / Description
0	IT_VRM	Firmware (FUMO) Update using FOTA, requires a reboot. Delta generation is not supported.
1	IT_EXTERN	External
2	IT_SYMBIAN	Symbian
3	IT_JAVA	Java
4	IT_BREW	Brew MP. Native installer type.
5	IT_LINUX	Linux
6	IT_BP	Baseband Processor
7	IT_BOOTLESS	Bootless
8	IT_CAB_RECREATION	Windows
9	IT_FOTA	Firmware (SCOMO) Firmware update using the Update Agent, requires a reboot. Delta generation is not supported.
10	IT_APK	Android Android user application. Native installer type, .apk files only.
11	IT_SYSAPK	Embedded Android Uses the Update Agent, requires a reboot. Supports .zip or .apk files only.
12	IT_WIDGET	WAC Widget. Native installer type.

Num	Name	UI Text / Description
13	IT_SED	Secure Enterprise Domain firmware for dual-persona s, requires a reboot. Delta generation is not supported.
14	IT_VLM	Firmware (TRUE).
15	IT_POLICY	Security policy
16	IT_IOS	iOS. Native installer type, .ipa only. Delta generation is not supported.
200-220	IT_GENERIC_200 … IT_GENERIC_220	Generic 200 … Generic 220 Generic installers 200 … 220. Native installer types enabling any file type. These are customer-specific installer types.
221-249	IT_GENERIC_221 … IT_GENERIC_249	Generic 221 … Generic 249 Generic installers 221 … 249. Native installer types enabling any file type.
250-251	IT_GENERIC_250 … IT_GENERIC_251	Generic 250 … Generic 251 Non-native installer types enabling .zip files only. These are customer-specific installer types.
252-299	IT_GENERIC_252 … IT_GENERIC_299	Generic 252 … Generic 299 Non-native installer types enabling .zip files only.

Appendix I Features to Implement When Using the HTML5 DIL

The HTML5 DIL does not implement all the features presented in this document. When using only the HTML DIL, you must implement the following features.

I.1 Move to Background

You must implement `FinishUI()` in **native_dil_actions.js** to close the UI and move the Software Management Client to the background after receiving certain events. For example, when the user presses **OK** on the **Installation Done** screen or presses the home button or its equivalent.

I.2 Background State Persistence

The Software Management Client must maintain the current state when it is moved to the background or closed. For example, if the Software Management Client is displaying a screen that asks for input from the end-user, and the screen is closed without end-user input, the same screen must be displayed when the Software Management Client returns to the foreground.

The exceptions to this are screens that indicate errors or the completion of a process. If the Software Management Client is moved to background while one of these screens is displayed, and then the Software Management Client returns to the foreground, the Software Management Client can resume on the next screen (as if the screens had been closed normally).

I.3 Move to Foreground

The DIL must listen for DIL events while running in the background. Some of these events require the Software Management Client to move to the foreground and display a message or receive end-user input.

The DIL events that might cause the Software Management Client to move to the foreground include:

- `DMA_MSG_DL_INST_ERROR_UI`
- `DMA_MSG_DM_ERROR_UI`
- `DMA_MSG_DNLD_FAILURE`
- `DMA_MSG_SCOMO_DL_CANCELED_UI`
- `DMA_MSG_SCOMO_DL_CONFIRM_UI`
- `DMA_MSG_SCOMO_DL_PROGRESS`
- `DMA_MSG_SCOMO_DL_SUSPEND_UI_FROM_ICON`
- `DMA_MSG_SCOMO_INS_CHARGE_BATTERY_UI`
- `DMA_MSG_SCOMO_INS_CONFIRM_UI`
- `DMA_MSG_SCOMO_INS_UI_CONDITIONS`
- `DMA_MSG_SCOMO_INSTALL_PROGRESS_UI`
- `DMA_MSG_SCOMO_NOTIFY_DL_UI`
- `DMA_MSG_SCOMO_POSTPONE_STATUS_UI`
- `DMA_MSG_SCOMO_REBOOT_CONFIRM_REQUEST`
- `DMA_MSG_SET_DL_COMPLETED_ICON`

I.4 Background Notification Alerts

You can implement a notification framework to display some messages as notifications. Notifications are often preferable to moving the Software Management Client to the foreground to provide a less invasive end-user experience.

I.5 Status Change Events

The DIL must send the following BL events when the device status changes, without any prompting from the business logic.

Table I-1: State Change Events

Change	BL Event
Wi-Fi connectivity	DMA_MSG_STS_WIFI
Cellular network connectivity	DMA_MSG_STS_MOBILE_DATA
Power on	DMA_MSG_STS_POWERED
Roaming	DMA_MSG_STS_ROAMING
Voice call start/end	DMA_MSG_STS_VOICE_CALL_START DMA_MSG_STS_VOICE_CALL_STOP

I.6 Server Initiated Sessions

To support server-initiated sessions, the Software Management Client must receive WAP push notifications by SMS or GCM. The DIL must receive the WAP push notification and extract the message.

The message is an NIA if the Content Type field in the WAP header is `C4`. For more information, see Appendix C, Example of Trigger Message from Server in http://vallejo.cc/proyectos/envio%20sms_files/OMA-TS-DM_Notification-V1_2-20060602-C.pdf [PDF].

For an NIA, the DIL must send the BL event `DMA_MSG_NET_NIA`, where `DMA_VAR_NIA_MSG` contains the extracted data and `DMA_VAR_NIA_ENCODED` is set to `1` if the data is ASCII hex encoded, or `0` if the data is binary.

Example code:

```
event.name = "DMA_MSG_NET_NIA";
event.values = VDM_SMM_Value_createBin("DMA_VAR_NIA_MSG", nia_msg,
sizeof nia_msg - 1);
event.values->next =
VDM_SMM_Value_createUint("DMA_VAR_NIA_ENCODED", 1);
```

I.7 Battery Level Check

You must implement a mechanism that checks the battery level. In response to the DIL event `DMA_MSG_GET_BATTERY_LEVEL`, the DIL must send the BL event `DMA_MSG_BATTERY_LEVEL` where `DMA_VAR_BATTERY_LEVEL` is set to the current battery level.

I.8 Porting Layer Callbacks

You must implement the callbacks required to retrieve device information. For more information, see Native Porting Layer Functions.

I.9 Lock and Wipe (LAWMO)

To support LAWMO, the DIL must handle the following DIL events:

- `DMA_MSG_LAWMO_LOCK_LAUNCH`
    - On success: Send `DMA_MSG_LAWMO_LOCK_ENDED_SUCCESS`.
    - On failure: Send `DMA_MSG_LAWMO_LOCK_ENDED_FAILURE`.
- `DMA_MSG_LAWMO_UNLOCK_LAUNCH`
    - On success: Send `DMA_MSG_LAWMO_UNLOCK_ENDED_SUCCESS`.
    - On failure: Send `DMA_MSG_LAWMO_UNLOCK_ENDED_FAILURE`.
- `DMA_MSG_LAWMO_WIPE_AGENT_LAUNCH`
    - On failure: Send `DMA_MSG_LAWMO_WIPE_AGENT_ENDED_FAILURE`

Appendix J Terms and Abbreviations

Term	Description
Application Core Layer	An Application Layer sub-layer that contains the SMM
Application Layer	The part of the Software Management Client containing the business logic and managing communication external to the device, such as networking and the UI The Application Layer is built on the Framework and contains the Application Core Layer, Business Logic Layer, and DIL sub-layers.
Business Logic Event	An event typically generated by the end-user or another external source, such as an incoming call or a message received from a DM Server The DIL passes Business Logic events to the Business Logic Layer. Business Logic events are queued before they are processed by the business logic. Certain Business Logic events are generated by the Framework internally or by the DIL in response to requests (DIL events) from the Business Logic Layer.
boot.img	The image of the device main system
Business Logic	One or more state machines within the Business Logic Layer The business logic responds to Business Logic events, instructs the Engine to perform actions, and generates internal and DIL events.
BLL	Business Logic Layer A mostly platform-independent Application Layer sub-layer that contains the business logic and one side of the Event Streamer
DD	Download Descriptor, a small file that contains information (size, location, etc.) about a file to download
DIL	Device Integration Layer A platform-dependent Application Layer sub-layer that contains the UI and platform traps
DIL event	An event that typically results in a UI screen presented to the end-user The Business Logic Layer generates DIL events and sends them to the DIL. Some DIL events are handled directly by the DIL without any change in the UI.
DL	Download
DM	Device Management
DP	Deployment Package

Term	Description
Engine	Redbend’s OMA-DM Protocol Engine The Engine contains the core library of the Framework
Event Streamer	A process of communication between the DIL and Business Logic Layer
Extended Framework	Redbend’s OMA-DM Protocol Engine Framework with the Application Core Layer
FOTA	Firmware Over-the-Air
Framework	Redbend’s OMA-DM Protocol Engine Framework The Framework contains the OMA-DM Core and Porting Layers The Application Layer communicates with the Framework using a comprehensive list of API and callback functions.
FUMO	Firmware Update Management Object
GCM	Google Cloud Messaging
IPC	Inter-Process Communication
IPL	Integration Point Layer
LAWMO	Lock And Wipe Management Object
Main System	The device main system, loaded from **boot.img**
MMI	Man Machine Interface
OMA	Open Mobile Alliance
OTA	Over-the-Air
Platform Traps	DIL functionality that handle external and device generated events, such as incoming phone calls and messages from the DM Server These events are passed as Business Logic events to the Business Logic Layer.
Porting Layer	A platform-dependent Framework layer that contains functions used by the Application Layer and the Engine
Recovery System	The device recovery system, loaded from **recovery.img**
recovery.img	The image of the device recovery system
SCOMO	Software Component Management Object

Term	Description
SMM	State Machine Manager Libraries for selecting and implementing the business logic; part of the Application Core Layer
Update Agent	Device-resident software that calls the Update Installer to perform updates
Update Generator	A Redbend product that generates updates of device firmware
Update Installer	Redbend device-resident software that installs updates on a device
UI	User Interface A DIL component that passes screens to the end-user, and passes information entered by the end-user to the Business Logic Layer as Business Logic events