**Medtronic MITG**

**Research & Development**

**Messaging Theory of Operations and Interface Specs**

**Gateway Device Management Portal v4.0**

[**RE00062811**](javascript:noaction();)  **Rev A**

|  |  |  |  |
| --- | --- | --- | --- |
| Issue Date: Refer to Agile | | | Page 1 of 126 |
|  | | | |
|  | | | |
| Revision History: | | | |
| Revision | SDA | Description | |
| A | RC088973 | Initial Release | |
|  |  | | |
|  |  | | |
| Author: Refer to Agile for approvals | | | |
| Paul Ourada, R&D , HI&M | | | |
|  | | | |
|  | | | |
| **Approvals:** Refer to Agile for approvals | | | |
| Ashwini Pandey – R&D, HI&M | | | |
| Hari Damineni – R&D, Ventilation | | | |
| Jesse Denson – R&D, Compression | | | |
| Casey Ladtkow, R&D, Ablation | | | |
|  | | | |
|  | | | |

**Table of Contents**

[1 Introduction 5](#_Toc471812951)

[1.1 Purpose and Scope 5](#_Toc471812952)

[1.2 Background System Description 5](#_Toc471812953)

[1.3 Communication Model 6](#_Toc471812954)

[1.4 Client Classes 6](#_Toc471812955)

[2 Common Communication Definitions and Enumerations 6](#_Toc471812956)

[3 Legacy Client Communication 7](#_Toc471812957)

[3.1 Notifications 7](#_Toc471812958)

[3.1.1 Notification Types 7](#_Toc471812959)

[3.1.2 Notification Actions 7](#_Toc471812960)

[3.2 XML Messages 7](#_Toc471812961)

[3.2.1 Schema 7](#_Toc471812962)

[3.2.2 Client Implementations 8](#_Toc471812963)

[3.2.3 Request Message 8](#_Toc471812964)

[3.2.4 Response Message 8](#_Toc471812965)

[3.2.5 Message Header 9](#_Toc471812966)

[3.2.6 Message Body 9](#_Toc471812967)

[3.2.7 Timestamp in the messages 9](#_Toc471812968)

[3.2.8 GUIDs 10](#_Toc471812969)

[3.2.9 Pertinent Types 13](#_Toc471812970)

[3.2.10 Device URIs 13](#_Toc471812971)

[3.2.11 Object IDs 14](#_Toc471812972)

[3.2.12 Specific Commands 14](#_Toc471812973)

[3.2.13 Message Part Constructs 14](#_Toc471812974)

[3.2.14 Error messages 18](#_Toc471812975)

[3.3 Connection and Communications Flows 19](#_Toc471812976)

[3.3.1 Basic Messaging API Communication Flow 19](#_Toc471812977)

[3.3.2 Session Management 19](#_Toc471812978)

[3.3.3 Device Management 21](#_Toc471812979)

[3.3.4 Overall Client/Agent Workflow 25](#_Toc471812980)

[3.3.5 Basic Connectivity and Session Management 25](#_Toc471812981)

[3.3.6 Device Information Synchronization 26](#_Toc471812982)

[3.3.7 Upload Logs 32](#_Toc471812983)

[3.3.8 Update Software 33](#_Toc471812984)

[3.3.9 View Document 37](#_Toc471812985)

[3.3.10 Serial Number Reprogram 38](#_Toc471812986)

[3.3.11 Prep Step 39](#_Toc471812987)

[3.4 Messages and Responses 41](#_Toc471812988)

[3.4.1 Session Management 41](#_Toc471812989)

[3.4.2 Authentication 46](#_Toc471812990)

[3.4.3 Agent Status 53](#_Toc471812991)

[3.4.4 Device Management 57](#_Toc471812992)

[3.4.5 Notification Management 67](#_Toc471812993)

[3.4.6 Post Notifications 80](#_Toc471812994)

[3.4.7 Prep Step 111](#_Toc471812995)

[3.4.8 GetSysConfigs 112](#_Toc471812996)

[3.4.9 Self-Registration 117](#_Toc471812997)

[4 REST Client Communication 119](#_Toc471812998)

[4.1 DownloadFile 119](#_Toc471812999)

[4.2 Service Definitions 119](#_Toc471813000)

[4.2.1 DownloadFile 119](#_Toc471813001)

[4.3 HwSw Config 119](#_Toc471813002)

[4.4 ClientApp 120](#_Toc471813003)

[4.4.1 Call 120](#_Toc471813004)

[4.4.2 Return 120](#_Toc471813005)

# Introduction

The Medtronic Gateway Device Management Portal capability is intended to enable “remote” interaction to medical devices performing such service oriented tasks as getting log files from devices or loading new software onto the device. This document describes two aspects of the system’s communications – between the service application and the remote service agent, and between the remote service agent and the Application Server.

## Purpose and Scope

The purpose of this document is to describe the messaging interface between the Service Client(e.g., VLEX, Enhanced Service Software, SCD Updater, Common Client, etc.) and the Remote Service Agent (RSA).

This document describes the “as built” communications interface. The interface between the RSA and Service Clients is managed by this communications contract.

# Common Terms and Definitions

**GDMP** - Gateway Device Management Portal. The combination of the servers in the cloud and RSA which provide device management functionality to Service Clients.

**RSA** – Remote Service Agent. Software component which is used by Service Clients to communicate with the GDMP Server in the cloud.

**Service Client –** Software component which communicates directly with the device being serviced using the device communication protocol, and the GDMP Server via the RSA.

**Application Server** - aka App Server. The GDMP server with which the RSA communicates at the behest of the Service Client.

**API** - Application Programming Interface. A predefined set of functions or services which are used to communicate with an application.

**GMT** - Greenwich Mean Time. The previously globally agreed imaginary line indicating the beginning of the day. All other time zones are defined in relation to this line.

**UTC** - Coordinated Universal Time. – The time standard presently used against which all time zones are determined. Essentially identical to GMT, but GMT is now merely a time zone.

**GUID** - Globally Unique Identifier. A standard for generating a random 128 bit number which is always unique on the globe. Similar to UUID which claims to be unique in the universe.

**URI** - Universal Resource Identifier. A schema to define the name and location of compute data, engines, etc. Most common schema is the URL for locating websites. But other schemas may be developed.

# Background System Description

The complete system is composed of:

* **Hardware device** (e.g., Ventilator, Pulse Oximeter, etc.)
* **Service Application** (e.g., Common Client)
* **Remote Service Agent** (RSA)
* **Application Server**
* **Data feeds** (e.g., manufacturing information, service contracts, etc.)

The hardware device is the terminal object with which we intend to interact. This is the source of device revisions, details, or logs. It is also the target for downloaded software.

The Application Server is populated with information and business rules to execute so that a Service Client can be provided with the proper information to service each device to which it is connected, based on the class of trade of the device, logged-in user’s rights, and other criteria.

The data feeds are enterprise back end systems and manufacturing feeds to the server providing knowledge about initial device configurations by device type and serial number, service contracts, software release information, purchasing customer, etc. These provide the basis for activity at the device management level, but are not significant to the remainder of this document.

The RSA is intended to support multiple device types, and multiple simultaneous device connections from a single Service Client.

The RSA is a service, acting as a proxy to the Application Server. It isolates the Service Client from the potential connectivity issues which exist with the server itself, and providing a common API for all Service Clients. For instance, most facilities do not allow external personnel to connect to their network, and often do not have guest networks. Cell cards often don’t work as the device to be updated is often in a Biomedical Lab, which is deep in the facility. The RSA must provide the same functionality as the Application Server in these circumstances. Due to this situation, the RSA must provide a means to access the information it needs from the Application Server while it is online, and present that information to the Service Client upon demand when it is offline.

## Communication Model

Communications are synchronous, with messages being sent from one system to another (Client 🡨🡪Agent 🡨🡪App Server) with expectation of a more or less immediate response. Communications are text based. Where appropriate, binary data will likely be base64 encoded (and appropriately identified as such).

An objective for the Remote Service Agent was for it to be “invisible”, such that the Service Client would act as if it was connected to the server itself, but without having to know that. However, due to the very nature of the connectivity, the Service Client does sometimes need to interact with the RSA knowing that RSA does exist. For instance, the Service Client needs to know when the RSA is done performing some function before the Service Client can continue.

## Client Classes

The RSA is designed to accommodate two classes of Clients. The first class is known as the Legacy Client class. These Clients communicate over TCP via a specific socket (localhost:9998), and pass XML messages and receive XML responses. The messaging interface for this class is detailed in section 2 Common Communication Definitions and Enumerations and section 3 Legacy Client Communication.

The second class, known as the REST Client class, communicates with the Agent via industry standard REST service URL (http://localhost/Agent) interfaces using JSON to encapsulate message and response data. The messaging interface for this class is detailed in section 2 Common Communication Definitions and Enumerations and section 4 REST Client Communication.

# Common Communication Definitions and Enumerations

This section describes those communication items which are common between both Legacy Clients and REST Clients.

# Legacy Client Communication

Legacy clients are defined as Valleylab Exchange (VLEX – used by Surgical Innovations), Enhanced Service Software (ESS – used by Patient Monitoring and Recovery for the PB980 Ventilator), and SCD Updater (SCDU – used by Patient Monitoring and Recovery for the SCD 700 Compression device).

## Notifications

The legacy Client communications system uses a model where messages would contain, in their header content, a high-level description of what the message is intended to accomplish. Two key pieces of information are provided: **notification type** and **notification action**.

### Notification Types

There are three notification types:

* Event – a “one-time” occurrence being reported
* Command – a command / demand / instruction to a partnering component
* Response – the response to a prior command

### Notification Actions

There are six notification actions:

* Create – the purpose of the notification is to create something.
* Delete – the purpose of the notification is to delete something.
* Modify – the purpose of the notification is to modify something.
* Replace – the purpose of the notification is to replace something.
* Get – the purpose of the notification is to make a request to get something.
* Set – the purpose of the notification is to set something.

The ”something” referred to above is clearly indicated by the specific message. At this point, not all actions are in use.

## XML Messages

### Schema

The basic message structure is composed of the standard XML header tag, followed by a message tag and the “**schema\_version**” attribute which it encodes.

<?xml version=”1.0” encoding=”utf-8”?>

<message schema\_version=”3644767c-2632-411a-9416-44f8a7dee08e”>

--- content ---

</message>

The message may be composed of XML elements of **head**, **body** or neither, which vary with need and content. Further detail is provided in the subsections below.

### Client Implementations

The DMP project promises to not change the message schema in a way that breaks existing Client implementations. However that does not mean that there will be no changes to the message schema. What this means in practical terms is that additional messages or optional tags in existing messages may be added, which existing Clients know nothing about.

Because of this possible change to the message schema, Customers must design their Client XML message handlers to ignore messages and tags which they do not recognize or understand. They must not fail to continue working in this condition. However, if a tag which is in the Client’s known repertoire has changed in its meaning or required content, then the Client should detect this and handle it through the normal application error handling mechanisms.

### Request Message

Most specific commands from the Service Client to the RSA are identified as **requests**. The general structure is an xml element request followed by a transaction GUID and, once known, a session GUID. The following shows an example.

<?xml version=”1.0” encoding=”utf-8”?>

<message schema\_version=”3644767c-2632-411a-9416-44f8a7dee08e”>

<request type=’createsession’ xaction\_guid=’0f16a851-a043-498d-a857-77ee5c59499c’ />

</message>

### Response Message

Requests made are expected to receive a **response**, usually indicating that the request was **ok** or **bad** and often receiving associated parameters / results. The general structure is an xml element response followed by a transaction GUID to indicate to which request the response is associated. The following shows an example (the response to the createsession request above).

<?xml version=”1.0” encoding=”utf-8”?>

<message schema\_version=”3644767c-2632-411a-9416-44f8a7dee08e” session\_guid=’bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3’>

<response type=’param’ xaction\_guid=’0f16a851-a043-498d-a857-77ee5c59499c’ >

<params>

<param name=’session’>bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3</param>

</params>

</response>

</message>

### Message Header

When used, this might read

<header>

<session>bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3</session>

<example>sample content</example>

</header>

### Message Body

When used, this might read

<body>

<is\_accessible>true</is\_accessible>

<last\_update>08/17/2012 12:00:00</last\_update>

<components>

-- component list here –

</components>

</body>

### Timestamp in the messages

Each message should have an element called <timestamp> which indicates the message creation time. This element is contained in the <params> element, and is formatted as a 32-bit UNIX timestamp. The value contained in this element is GMT/UTC .

This element is required for all messages emanating from the Agent or Server. In particular, the Agent and Server shall add the timestamp to responses to the Client.

This element is optional for the request message by the Client. However the Client should not provide this tag, as the Agent will always replace it with the Unix timestamp when it receives the message.

NOTE: Not all messages below show this required tag.

### GUIDs

To provide a level of flexibility and obscurity, GUIDs (global unique identifiers) are used to provide key tags used when identifying message types, destinations, etc. The GUIDs defined below were created by the development team and are considered “well known” (as they are documented here). Currently these are used for transactions and session ids as well as grouped to “define” specific object instances, such as mailbox classifications, device classifications, and message type classifications. As the need for additional common identifiers are recognized, the development team will create such and extend this document accordingly.

#### Schema GUID

To manage message revisions, a “**schema\_version**” GUID is defined and passed in every message element as an attribute, as follows:

<?xml version=”1.0” encoding=”utf-8”?>

<message schema\_version=”3644767c-2632-411a-9416-44f8a7dee08e” session\_guid=’bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3’ >

--- content ---

</message>

In the future, this GUID will serve as a guide to interpreting a message whose format may have evolved over time.

#### Session GUID

To manage connections and authentication, a session GUID will be created (as a result of a createsession message) which will always be passed with the messages thereafter. Once the session has been created or opened, the message element will contain the “**session\_guid**” attribute, as follows:

<?xml version=”1.0” encoding=”utf-8”?>

<message schema\_version=”3644767c-2632-411a-9416-44f8a7dee08e” session\_guid=’bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3’ >

--- content ---

</message>

Creation of the session GUID is the responsibility of the RSA. It persists for the duration of the session.

#### Transaction GUID

Because the system follows the pub / sub asynchronous communication model, there is no inherent mechanism by which to associate a reply with the originating message. To solve this, a transaction GUID, “**xaction\_guid**” attribute will be supplied with each request or response message (element). This should be, but is not required to be, unique. It might appear as follows:

<?xml version=”1.0” encoding=”utf-8”?>

<message schema\_version=”3644767c-2632-411a-9416-44f8a7dee08e”>

<request type=’createsession’ xaction\_guid=’0f16a851-a043-498d-a857-77ee5c59499c’ />

</message>

Because the message originator creates this GUID, the originator has the ability to recognize which response goes with which request.

Transaction GUID is generated by the originator of the message.

#### Mailbox Classifications

These GUIDs are used to represent different mailboxes (making it more difficult for someone snooping on the line to understand what is actually happening). These GUIDS remain static for the life of the system.

|  |  |
| --- | --- |
| Description (debug string) | GUID |
| DEVICE\_INBOX | 34CAA93B-6ED3-4A93-9BE7-4E31EB4B22D5 |
| DEVICE\_OUTBOX | 3BD775AD-7250-42FF-ADF4-188E0B691BFF |
| SYSTEM\_INBOX | B20AF193-6353-4CF5-95CC-E79B84D88907 |
| SYSTEM\_OUTBOX | 5B5FD529-4244-46BE-88CD-1A1304C179DF |

The system outbox is included for completeness, but is not expected to be used.

#### Device Classifications

Each device is associated with a GUID. These GUIDS remain static for the life of the system.

NOTE: The description can be used as a debug string or readable shortcut. Use of this shortcut in production is not allowed. The only exception is for the Client Info/Update message in which the “device” name is used as the Client “serial number” or instance identifier.

|  |  |
| --- | --- |
| Description (debug string) | GUID |
| PB980\_VENTILATOR | 7A85F0C9-531E-4754-AD68-04C77ED63657 |
| SCD-700 | 61E08B77-DF3C-4735-9F3B-0B42EFB7BDCF |
| Valleylab\_LS10 | 3B682913-6D1E-4355-9E48-208EB7061A3D |
| Valleylab\_FT10 | CEFC1E07-CFF6-4F27-AB05-4577A33A1BA8 |
| Emprint Procedure Planning Application | 61BF648E-3181-41e4-9EF2-222F8DF8B538 |
| VLEX Client | B0DC2BE4-D744-45c6-AEF6-EBEF319A336B |
| Emprint Client | 3695F650-E602-47E6-A3DF-5525BD41CCA3 |
| Signia | 5768C9CB-8ACE-4421-B1C6-071D131B935F |
| Situate 200E | 134089E4-AEFB-47D7-9E6F-2B64F19E1846 |
| Situate 200X | ABA45524-1C87-44FF-9B5C-D588D6016040 |
| Situate 200LD-V | D818C69E-DB2D-44E6-BF4C-47684F2680CA |
| VLFX8GEN | ADA07262-C1B1-4568-8F5D-938B0ABDF987 |

GUIDs for other devices to be added as implemented.

#### Message/Object Type Classifications

Each actionable message or message content descriptor has its own GUID.

|  |  |
| --- | --- |
| Description (debug string) | GUID |
| DEVICE\_INFO | 532FFDDA-6F38-41DA-9A40-CB0801E6695F |
| DEVICE\_SOFTWARE\_UPGRADE\_PACKAGE / DEVICE\_SOFTWARE | 24637560-EDB3-4961-B17D-14E74F74457C |
| DEVICE\_SOFTWARE\_DOWNLOAD\_ACK | 91DEC715-F256-421A-8560-E6B015DAE5F9 |
| DEVICE\_FIRMWARE\_UPGRADE\_PACKAGE / DEVICE\_FIRMWARE | A582E8A9-282A-4F49-93D1-F11547826A9 |
| DEVICE\_FIRMWARE\_DOWNLOAD\_ACK | 2C0D006F-2282-4417-BF20-2096B56A9FF6 |
| DEVICE\_HARDWARE | FC1F3C2C-16DF-4B52-A9EA-99409B131D31 |
| DEVICE\_LIST | 06949C6D-5893-4579-B587-736373685683 |
| DEVICE\_STAT | 4BEE9772-ADCB-4EC2-B944-3191492B3436 |
| DEVICE\_RAW\_LOG | 2F62D564-A162-440A-A5F6-ED16E7E632D5 |
| DEVICE\_DECODED\_LOG | 8E74DB99-F0A3-4DE4-AEED-F17AFB6896FC |
| RELEASE\_NOTES | 02D7AA7B-25E0-423F-8F98-C15612E6A7C0 |
| SERVICE\_MANUAL | C250C8DB-B532-4D18-8956-420C3D637A41 |
| USER\_GUIDE | F520B9CE-C641-5E29-9167-531D4E748B52 |
| OTHER (refers to document type) | 5789F84E-7F582-6E30-8289-692C57E59D63 |
| SYSTEMWIDE\_NOTIFICATION | 44972A33-8FC4-4742-A1FC-940F2CC1CC92 |
| FEATURE\_ENTITLEMENT | 832FFDDA-6F38-A1DC-9AF0-CB0801E6695D |

GUIDs for other specific identifiers to be added as implemented.

### Pertinent Types

Pub / sub messages are conceptually at a high level and further detail is often necessary to provide context. The pertinent type is a composition of the device classification and the message classification, as follows (without the excess blank spaces):

device classification GUID / message classification GUID

For example: the device information for a PB980 would read:

7a85f0c9-531e-4754-ad68-04c77ed63657/532ffdda-6f38-41da-9a40-cb0801e6695f

And, within the XML structure of the message, would read:

<pertinent\_type>7a85f0c9-531e-4754-ad68-04c77ed63657/532ffdda-6f38-41da-9a40-cb0801e6695f</pertinent\_type>

As can be seen, the guids defined in the section above are used.

### Device URIs

The specific device is referenced by its device classification GUID and relatively unique serial number. Because we cannot control the serial number, a choice was made to wrap each part of the data in curly braces (i.e., { and }). The construct is as follows (without the excess blank spaces):

{ device classification GUID } / { device serial number }

For example: the device URI for a PB980 might read:

{7a85f0c9-531e-4754-ad68-04c77ed63657}/{980 abc1234567}

### Object IDs

Communication objects, notifications specifically, are managed by an “object id” (OID). This identifier is provided by the RSA to the Service Client enabling the Service Client the ability to uniquely reference any object (e.g., message notification) of which it has knowledge.

### Specific Commands

Despite the original intent to follow the publish / subscribe model and use the pertinent type for further define the content of a message, several specific commands were specifically identified as follows:

|  |  |
| --- | --- |
| Description | Purpose |
| Createsession | Create a new session |
| Opensession | Open an existing session (session id handed off from an authenticated process) |
| Closesession | Close a session |
| Login | Login a given user (with password) for authentication |
| Logout | Logout the user |
| GetDeviceList | Get the list of devices of which the current user (representing a current facility) is allowed to have knowledge. |
| DeviceStat | Get the device status / statistics of a specific device |
| GetHeaders | Get the message headers that are available for the given user |
| GetNotification | Get the notification associated with a given header |
| PostNotification | Post the notification to the RSA |
| DeleteNotification | Delete the notification from the RSA |
| UpdateNotification | Update the notification at the RSA |
| ExpungeNotification | Remove completely the notification from the RSA |

The messages in the above list would be created from the service application and sent to the RSA. Where appropriate, and if connected, the RSA will message the server (which discussion is out of scope of this document).

### Message Part Constructs

Certain messages require meaningful construct(s) that may themselves be composed when appropriate. These are defined below.

#### Params

A “params” content provides a list of key-value-pairs, where the name attribute in the XML is the name of the param(eter) and is used as the key. The associate text of the XML is then used as the value.

**Tag Descriptions**

timestamp:optional from client:required from Agent/Server – a 32-bit Unix timestamp representing GMT time at time of message creation. Tag type attribute is “network” when the Agent is connected to the server. It is sent as “local” otherwise.

session:required – a GUID representing the current session

<params>

<timestamp type=network>UNIX timestamp</timestamp>

<session>bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3</session>

<example>sample content</example>

</params>

The following types of messages return params

|  |  |  |
| --- | --- | --- |
| Description | Direction |  |
| Createsession | response |  |
| Opensession | response |  |

#### Components

A “components” content provides a list of component details, where each component is defined as having a type and then a collection of internal elements further describing the component.

<component type=”HARDWARE”>

<name>INSPIRATORY</name>

<serial\_number>sample content</serial\_number>

<part\_number>sample content</part\_number>

<revision>sample content</revision>

<timestamp>time stamp</timestamp>

<status>installed</status>

<rights>

-- rights list here

</rights>

</component>

Naturally, the elements of a component may contain some, none, all, or more than exemplified above.

The following types of messages pass or return **params**.

|  |  |
| --- | --- |
| Description | Purpose |
| CreateSession | The resulting “session” or else a “reason” for failure |
| OpenSession | The resulting “session” or else a “reason” for failure |
| CloseSession | A “reason” for failure |
| Login | Request: “username” and “password” Response: a “reason” for failure |
| Logoff | A “reason” for failure |
| CreateDevice | Request: “device\_uri” Response: a “reason” for failure |
| DeleteDevice | Request: “device\_uri” Response: a “reason” for failure |
| StatDevice | Request: “device\_uri” Response: a “reason” for failure |
| GetHeaders | Request: “mailbox” and “device\_uri” Response: a “reason” for failure |
| GetNotification | A “reason” for failure |
| PostNotification | Request: “mailbox” |
| DeleteNotification | Request: “notification\_oid”  Response: a “reason” for failure |
| UpdateNotification | Request: “notification\_oid” (and the full notification?)  Response: a “reason” for failure |
| Expunge | Request: “notification\_oid” (or none if expunging all) Response: a “reason” for failure |

#### Rights

A rights list may accompany a component, indicating that the user has, for example, the right to install software.

<rights>

<right>install</right>

<right>remove</right>

</rights>

The list of available rights is not well defined at this point, and ultimately is a contract between the backend service management server and a given Service Client. However, here we need only detail the transport mechanism.

#### Logs

Log messages, whether for raw or decoded logs, require a meaningful construct for their body as defined below.

<logs>

<log name=’LogName’>

<transfer\_encoding>gzip</transfer\_encoding>

<uri\_inc>full path</uri\_inc>

</log>

</logs>

The “LogName” would be replaced with the actual name for the log.The “gzip” example shows how to indicate any external file formatting.

The “full path” is a path to the file wherein the log is stored.

### Error messages

|  |  |  |
| --- | --- | --- |
| Error Message | Occurrence | Resolution |
| Invalid session ID | Session closed or never opened | OpenSession message |
| Session Request Error | Close without open;  Logout with out login | Check state of the session |
| Login Session Not available | Session is open but not authenticated | Login  message |
| Login Failed | Bad credentials;  Locked account;  Disconnected mode without ever connecting to server with these credentials | Check credentials;  Unlock account through admin user on the web site;  Connect to the server via the agent; |
| Request not valid | Poorly formed message | Check the message documentation and correct syntax |
| Device Not Found | Agent/Server does not know of the device | createDevice message |
| Device already exists | Agent/Server knows the device | Sent a createDevice message but the device exists, send statDevice message first |
| Device information not found | Request for statDevice before sending or uploading device configuration info | Send deviceInfo message;  Upload configuration information to the server |
| No message in inbox | Message does not exist | Using expired notification\_Oid; |
| No notification found | Trying to expunge a message that no longer exist | Check state of the request |
| Bad notification\_Oid | Bad mailbox item identifier;  Trying to delete a previously delete message | Notification\_Oid is passed in the getHeaders message |
| No message in outbox | No messages in outbox | No action |
| Request was not appropriate | General Syntax error | Check message syntax and correct message |

## Connection and Communications Flows

The RSA is a service that will be running on the laptop, listening to socket port number 9999. The Service Client (also known as the Client) will connect to the server at the designated port and initiate communications. The following communications flows indicate interaction between the Service Client and the RSA.

### Basic Messaging API Communication Flow

The Device Management Portal messaging API has just a few types of messages. They fall into two categories: session management and device management.

### Session Management

Session management messages are initiated by Clients. The Agent is the respondent. As described above, these messages create, open, and close sessions as well as authenticate user login, and check Agent and Server status. The diagrams below show the typical session and login interaction between Clients and the Agent.





### Device Management

Device management transport messages are symmetrical – both Client and Agent may send and receive these messages. These messages take the form of notifications to the receiving party. The receiver polls to see if there are any notifications available, and then requests the messages from of their associated mailbox. The Agent controls this interaction by owning the mailbox and processing the post and receive messages. The diagram below shows the typical notification interaction between a Client and Agent.



Clients and Agents are treated as devices themselves. They have a special workflow to receive updates to Business Rules, or the Agent/Client itself. This message should only be issued after a user has logged in.



### Overall Client/Agent Workflow

In general, the Client must first contact the Agent and identify itself and its user. After that, several activities may be performed, such as uploading logs, updating a device’s configuration on the server, updating the device software, and so forth.



### Basic Connectivity and Session Management

The basic session setup and user login is shown above in Section 11.1.1, Session Management

### Device Information Synchronization

Device information synchronization refers to the process of sending the current device configuration information to the Agent and Server. This consists of three steps involving three different messages, StatDevice, CreateDevice and DeviceInfo.



StatDevice is used to query the Server to know whether the device exists. If it does exist, then the Server responds with the last known configuration.



CreateDevice is used to add an instance of the device record tables to the Server database, with the current device configuration. Since this message contains the current device configuration, a separate DeviceInfo message is not required.



DeviceInfo is used to upload the current device configuration to the Server for those Clients and devices which use Catalog Configuration. Catalog Configuration is a simple mapping of hardware catalog items to software catalog items using an OR relationship. The Agent or Server uses this information, plus other additional information to determine which software packages are available for software update.



Those Clients which Service Clients using Named Configurations use the SyncDeviceCfg and GetMatchedCfg messages.

Named Configuration maps groups of Hardware Catalog Items to a Named Hardware Configuration; groups of Software Catalog Items to a Named Hardware Configuration; and groups of Firmware Catalog Items to a Named Firmware Configuration. A Named System Configuration consists of one Named Hardware Configuration, one Named Software Configuration and zero or one Firmware Configuration.



In the case of a country change on the device, the following workflow can be used.



### Upload Logs

After the user has logged in and the current device configuration has been uploaded, then multiple operations are possible. It is suggested that the Client next upload log information before attempting to modify the device in any way.



### Update Software

To upgrade software on a device, first look for the download time to be zero in the response to GetStatus. This indicates that all files have been downloaded from the server.

NOTE: If the Agent has just requested files from the Server, then the download time has not yet fully been determined, and the result reported in GetStatus will be MAX\_FLOAT. Do not use this value to determine when the Client will next check the download time, as this will cause unnecessary delays. Instead, check the download time with a delay of perhaps one second until the download time is less than MAX\_FLOAT. That time will be more indicative of the actual time the Agent expects the file download will take.

After that, then a GetHeaders or GetMatchedConfigs message, depending on the configuration scheme to be used, retrieves the URIs of the files that are available. Finally, after the download is complete, the Client tests to see whether there are any files available. If there are files which are appropriate to this device, the Client next requests those files from the Agent, and updates the device SW with them. Once complete, the Client notifies the success of the operation to the server.





### View Document

The sequence for viewing a document is very similar to software update, except that no software is sent to the device.



### Serial Number Reprogram

This is the sequence of events for communicating a serial number reprogram event. Any time the device serial number has been reprogrammed, the Client should send this message. The DMP keeps track of the before and after serial number so that the DMP can detect the situation where a board swap occurred, and the other device may still have the wrong serial number.



### Prep Step

This is the sequence of steps that a Client should use if it is going to implement Prep Steps functionality. Prep Steps is used to focus the downloading of packages on those device types which a particular Client is responsible. The Prep Steps message itself can be used by the Client to further focus the packages to be downloaded to a subset of those devices for which it is responsible. Typically, this option is given to the end user of the Client.

The primary actions that a Client must take are these: At login, announce to the Agent which Client it is (GUID), and whether Prep Steps should be enabled. After login, the Client should issue the Prep Steps message to define which device packages should be downloaded.



## Messages and Responses

Following are definitions of messages and their responses. Note that one type of message, PostNotification, actually defines several additional interactions between the Client and the DMP. These interactions are typically specific to a particular device instance.

### Session Management

#### Create Session

A session\_guid id is generated by the laptop agent when the client requests a session. It is used by the laptop agent to distinguish between multiple application sessions and to assigns state to an application

Call:

<?xml version='1.0' encoding='utf-8'?>

<message schema\_version='3644767c-2632-411a-9416-44f8a7dee08e'>

<request type='createsession' xaction\_guid='3ca142fe-7d59-4969-9f92-e89d95b4b036' />

<params>

<timestamp type=network>UNIX timestamp</timestamp>

<sessionID>740306bb-a7e4-4a1c-be40-ffa0b720e284</sessionID>

</params>

</response>

</message>

Response:

On success, a new session GUID is returned.

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<message session\_guid="740306bb-a7e4-4a1c-be40-ffa0b720e284" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<response xaction\_guid="3ca142fe-7d59-4969-9f92-e89d95b4b036" type="ok">

<params>

<timestamp type=network>UNIX timestamp</timestamp>

<sessionID>740306bb-a7e4-4a1c-be40-ffa0b720e284</sessionID>

</params>

</response>

</message>

On failure, an error response is returned with an indication of reason.

**Reasons**

“Session Request Error” – problem with session request.

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<message session\_guid="" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<response xaction\_guid="dac312d1-49c9-48ce-aed2-3c9c8ad3a315" type="bad">

<params>

<timestamp type=network>UNIX timestamp</timestamp>

<reason>Error ?????</reason>

</params>

</response>

</message>

#### Open Session

Call:

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<request type='opensession' xaction\_guid='9650e555-810d-487b-94bf-da8d2f0e334e'>

<params>

<timestamp type=network>UNIX timestamp</timestamp>

<sessionID>bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3</sessionID>

</params>

</request>

</message>

Response:

On success, a confirmation message indicating that a session has been opened.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='ok' xaction\_guid='9650e555-810d-487b-94bf-da8d2f0e334e'>

<params>

<timestamp type=network>UNIX timestamp</timestamp>

</params>

</response>

</message>

On failure, an error response is returned with an indication of reason.

**Reasons**

“Invalid session ID” – Session can’t be opened for the requested session ID.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='0f16a851-a043-498d-a857-77ee5c59499c'>

<params>

<timestamp type=local>UNIX timestamp</timestamp>

<reason>Error ?????</reason>

</params>

</response>

</message>

#### Close Session

Call:

<?xml version='1.0' encoding='utf-8'?>

<message schema\_version='3644767c-2632-411a-9416-44f8a7dee08e' session\_guid='8a0c3837-0f4c-44ab-944f-154529be78f6'>

<request type='closesession' xaction\_guid='2e67dbaa-d81c-4873-b07c-8bb651ab2e8c' />

</message>

Response:

On success, a confirmation message indicating that the session is closed.

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

<message session\_guid="3dbe120e-846c-42cf-8cf2-bd6501cca1da" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<response xaction\_guid="fdd03294-6485-475a-b5b8-f69cf48d049a" type="ok">

<params>

<timestamp type=local>UNIX timestamp</timestamp>

<sessionID>3dbe120e-846c-42cf-8cf2-bd6501cca1da</sessionID>

</params>

</response>

</message>

On failure, an error response is returned with an indication of reason.

**Reasons**

“Invalid session ID” – Session for the requested session ID can’t be found.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='0f16a851-a043-498d-a857-77ee5c59499c'>

<params>

<timestamp type=local>UNIX timestamp</timestamp>

<reason>Error ?????</reason>

</params>

</response>

</message>

#### Session Sequence

The correct message sequence is as follows:

1. Call CreateSession
   1. Remember ID” – Session for the requested session GUID for future reference
2. Call Login (section 12.2.1)
3. For any helper process:
   1. Call OpenSession
4. Perform other Agent/Server interactions
5. Call Logout (section 12.2.2)
6. For each CreateSession and OpenSession
   1. Call CloseSession
7. Go to Step 1 to re-initiate a login.

NOTE: A properly behaving client application will properly close out the sessions associated with the client upon exit or crash. Failure to do so may cause the socket associated with the session to notID can’t be released and the Agent may not properly function, such as attempting to reconnect to the Server after the user logs out.

### Authentication

#### Login

The Login command is used to identify and authenticate a user, associating him/her to a session. If there is no Client session, then the Client must open a new session prior to issuing this request. If this request fails, it is best practice for the Client to close the current session and then create and open a new session prior to allowing the user to log in again.

The login command is also where the Client enables the Prep Step feature (see Section 12.7, Prep Step). If this occurs, then retrieval of the Named System Configuration details for that user is postponed to the handling of the Prep Step message.

Call:

<?xml version='1.0' encoding='utf-8'?><?>

<message schema\_version='3644767c-2632-411a-9416-44f8a7dee08e' session\_guid='8a0c3837-0f4c-44ab-944f-154529be78f6'>

<request type='login' xaction\_guid='6ca7eec2-e6e2-40ab-97fa-2f293c8a140b'>

<params>

<timestamp type=network>UNIX timestamp</timestamp>

<prepstep\_enabled>true|false</prepstep\_enabled>

<client\_guid>Client\_GUID/CLIENT\_PC\_MAC\_ADDRESS</client\_guid>

<username>some.user@covidien.com</username>

<password>pass1234\*</password>

</params>

</request>

</message>

**Tag Descriptions**

<prepstep\_enabled> : optional - true|false – causes the Agent to accept the Prep Step message. If true, causes the Agent to pause downloading until the Prep Step message is received.

<client\_guid> : optional – GUID of the client which is logging in. It is **required** for any Client which intends to use Prep Step functionality. The second half of this tag, CLIENT\_PC\_MAC\_ADDRESS is the octet MAC address of the PC ethernet port. Report it in numeric format – no ‘-‘s.

<username> : required – the user who is logging in

<password> : required – the user’s password

Response:

On success, a confirmation message indicating that the user has successfully logged in and is in the authenticated state.

<user\_access> is one of {none=0, latest only=1, all production versions=2}

<country> is one of ISO 3166-1 alpha-2 code see, <https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2>

<covidien\_user> Yes/No identifies the user as a Covidien Employee

<testing\_ software>true/false gives permission to download and install software that is not yet in production status.

<training> DeviceType name allow the user to train and enroll other users for that DEVICE

<passExpiration> Date that the user’s password will expire

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<message session\_guid="SESSION\_ID" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<response xaction\_guid="e1c38f30-50a0-4ccf-a74a-c7ec11b54544" type="ok">

<params>

<timestamp type=local>UNIX timestamp</timestamp>

</params><userlogin>

<country>US</country>

<passExpiration>2014-11-05 01:13:49.0</passExpiration>

<permissions>

<permissoin>

<device\_type>DEVICETYPE1</device\_type>

<user\_access>ACCESSLEVEL 0/1/2</user\_access>

<testing\_software>false</testing\_software>

</permissoin>

<permissoin>

<device\_type>DEVICETYPE2</device\_type>

<user\_access>ACCESSLEVEL 0/1/2</user\_access>

<testing\_software>true</testing\_software>

</permissoin>

</permissions>

<covidien\_user>Yes</covidien\_user>

<training>

<device\_type>DEVICETYPE1</device\_type >

</training>

</userlogin>

</response>

</message>

On failure, an error response is returned with an indication of reason.

**Reasons**

"Login Failed - password expired” – Use password must be changed first

“No client session. You may need to restart the Client.” – Disconnected mode - session used to log in is not found

“Invalid credentials. Connect to the internet and try again.” – Disconnected mode – user is unknown or not in the Agent login cache. May be first time login for user, which must be done while in connected mode.

“Invalid credentials. Session invalidated. You may need to restart the Client.” – Disconnected mode – password is invalid. Session is closed for security reasons. Client must reopen session, or user must restart Client.

“change proxy failed.” – Proxy credentials not found, or invalid

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='0f16a851-a043-498d-a857-77ee5c59499c'>

<params>

<timestamp type=local>UNIX timestamp</timestamp>

<reason>Error ?????</reason>

</params>

</response>

</message>

#### Logoff

The Logoff command informs the server that a user is no longer associated with a given session. Best practice is for the Client is to close the current session after sending this request.

Call:

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid="3dbe120e-846c-42cf-8cf2-bd6501cca1da">

<request type="logoff" xaction\_guid="ad21c55c-5360-488c-be06-01e6707bcfc3">

<params>

<timestamp type=network>UNIX timestamp</timestamp>

<username>SCD.admin@covidien.com</username>

</params>

</request>

</message>

Response:

On success, a confirmation that logoff completed, user is no longer associated.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3'>

<response xaction\_guid='23b2320b-6378-4013-80fb-a05ffe199e9c' type='ok'>

<params>

<timestamp type=local>UNIX timestamp</timestamp>

</params>

</response>

</message>

On failure, an error response is returned with an indication of reason.

**Reasons**

“Invalid session ID” – User has already logged off, or session ID is stale

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='0f16a851-a043-498d-a857-77ee5c59499c'>

<params>

<timestamp type=local>UNIX timestamp</timestamp>

<reason>Error ?????</reason>

</params>

</response>

</message>

Note: See Section 12.1.4, Session Sequence, for an explanation of how the session and login messages should be implemented by the Client.

#### Forgot password

The forgotpassword command informs the server that a user forgot the password and needs a new password.

Call:

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

<message schema\_version=’3644767c-2632-411a-9416-44f8a7dee08e’ session\_guid='8a0c3837-0f4c-44ab-944f-154529be78f6'>

<request type='forgetpassword' xaction\_guid='6ca7eec2-e6e2-40ab-97fa-2f293c8a140b'>

<params>

<username>user@mail.com</username>

</params>

</request>

</message>

Response:

On success, a mail contains url link which redirects to a page to reset password is sent to user’s mailbox..

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3'>

<response xaction\_guid='23b2320b-6378-4013-80fb-a05ffe199e9c' type='ok'/>

</message>

On failure, an error response is returned with an indication of reason.

**Reasons**

“Login Failed” – The Agent user could not log in

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='0f16a851-a043-498d-a857-77ee5c59499c'>

<params>

<timestamp type=local>UNIX timestamp</timestamp>

<reason>Error ?????</reason>

</params>

</response>

</message>

#### Change password

The changepassword command informs the server that a user need to change password.

Call:

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

<message schema\_version=’3644767c-2632-411a-9416-44f8a7dee08e’ session\_guid='8a0c3837-0f4c-44ab-944f-154529be78f6'>

<request type=’changepassword’ xaction\_guid='6ca7eec2-e6e2-40ab-97fa-2f293c8a140b'>

<params>

<username>[user@mail.com</username](mailto:user@mail.com%3c/username)>

<password>oldPassword</password>s

<new\_password>newPassword</new\_password>

</params>

</request>

</message>

Response:

On success, a notification mail is sent to user’s mailbox..

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3'>

<response xaction\_guid='23b2320b-6378-4013-80fb-a05ffe199e9c' type='ok'>

<params>

<timestamp type=local>UNIX timestamp</timestamp>

<reason>Error ?????</reason>

</params>

</response>

</message>

On failure, an error response is returned with an indication of reason.

**Reasons**

“Invalid request” – Improperly formed change password request

“Logging Session Not Available” – Login session which made the request does not exist

“use name is not available” – User name for requesting session does not exist

“Invalid Username” – user name did not return valid login info

“old password is not correct” – the current password specification in message does not match the value currently associated with user

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='0f16a851-a043-498d-a857-77ee5c59499c'>

<params>

<timestamp type=local>UNIX timestamp</timestamp>

<reason>Error ?????</reason>

</params>

</response>

</message>

### Agent Status

#### Get status

This will tell the client applications about the current status of the RSA. It contains a wealth of information about the current agent status. This information is continuously updated and can be quarried regularly to update the status information of the client application.

**Tag Descriptions**

<CoT\_name>:optional – Name of a Class of Trade. Used to tell the Agent to report on all devices for this Class of Trade. Defaults to the current Class of Trade(s) serviced by the Client which currently has an open session with the Agent.

<DeviceType\_GUID>: optional – The GUID for a specific device type. Used to tell the Agent to report on this device only.

Call:

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3'>

<request type='getstatus' xaction\_guid='9650e555-810d-487b-94bf-da8d2f0e334e'/>

<cot\_name>Vessel Sealing</cot\_name>

<devicetype\_guid>DeviceTypeGUID1</devicetype\_guid>

</message>

Response:

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

<message session\_guid="3dbe120e-846c-42cf-8cf2-bd6501cca1da" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<response xaction\_guid="c418263b-cb77-4c32-92a2-c3a7ae8f713c" type="param">

<params>

<timestamp type=local>UNIX timestamp</timestamp>

<agent\_status>idle</agent\_status>

<cache\_timeout>Jun 25, 2013 12:28:46</cache\_timeout>

<data\_ready>true</data\_ready>

<datetime>Jun 24, 2013 11:51:40</datetime>

<job\_status>complete</job\_status>

<last\_update>Jun 24, 2013 11:51:40</last\_update>

<server\_status>connected</server\_status>

<server\_available>true</server\_available>

<upload\_time>0.0</upload\_time>

<download\_time>234589

<devicetype guid= “DT1\_GUID”>remained\_time\_For\_DT1</devicetype>

<devicetype guid= “DT2\_GUID”>remained\_time\_For\_DT2</devicetype>

<devicetype guid= “DT3\_GUID”>remained\_time\_For\_DT3</devicetype>

</download\_time>

<Used\_Space>10.01M

<devicetype guid= “DT1\_GUID”>used space\_For\_DT1</devicetype>

<devicetype guid= “DT2\_GUID”>used space For\_DT2</devicetype>

<devicetype guid= “DT3\_GUID”>used space For\_DT3</devicetype>

</Used\_Space>

<Free\_Space>10.12M</Free\_Space>

</params>

</response>

</message>

**Tag Descriptions**

NOTE: All tags in this section are required unless specified otherwise.

<agent\_status> The activity status of the agent idle, downloading, etc.

< cache\_timeout > The datetime when the oldest files in the cache will be removed – because they have not been used.

<data\_ready> Data is ready to be uploaded to the server, so if you are disconnected you should find a network connection to allow the agent to upload the logs.

<datetime> current date and time from the RSA

<job\_status> The status of the most active job running on the RSA

<last\_update> The last time the RSA was connected to the server, allowing updates to the software cache and the uploading of logs.

< server\_status > state of the server connection either connected or disconnected

<server\_available> true/false – reflects the ability of the agent to see the server.

< upload\_time > time requirement estimate for uploading logs to the server

<download\_time> time requirement estimate for the downloading of software to the RSA  
NOTE: The Client should always update any Client-side progress calculation upon receiving this download time. This value may be very large at the start. Calculating an update time based upon this could unwarranted delays as the Agent finishes downloading before the Client next displays or calculates progress.

<devicetype> Optional. Sub tag of <download\_time> which specifies the time to complete download for a particular device.   
Optional. Sub tag of <Used\_space> which specifies how much disk space a particular device is occupying.

<Used\_space> Optional. The total disk space used by all devices.

<Free\_space> Optional. The available disk space.

#### Disconnect

The disconnect command informs the Agent to change status to disconnect.

Call:

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

<message schema\_version=’3644767c-2632-411a-9416-44f8a7dee08e’ session\_guid='8a0c3837-0f4c-44ab-944f-154529be78f6'>

<request type='disconnect' xaction\_guid='6ca7eec2-e6e2-40ab-97fa-2f293c8a140b'>

</request>

</message>

Response:

On success, Agent changes status to disconnected..

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3'>

<response xaction\_guid='23b2320b-6378-4013-80fb-a05ffe199e9c' type='ok'/>

<params>

<timestamp type=local>UNIX timestamp</timestamp>

<reason>Error ?????</reason>

</params>

</response>

</message>

On failure, an error response is returned with an indication of reason.

Reasons:

“Loggin Session Not available” – Client issuing the command does not have a valid session.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='0f16a851-a043-498d-a857-77ee5c59499c'>

<params>

<timestamp type=local>UNIX timestamp</timestamp>

<reason>Error ?????</reason>

</params>

</response>

</message>

### Device Management

#### Create Device

The Create Device command is used to create a device node, allowing the agent to know about the existence of a given device. It is used when a device is connected to the Service Application but the RSA does not recognize the device. When this message is invoked the RSA will create an entry for the unknown device. The user only need be registered with the Gateway, but does not need to have a training record for the requested device in order to create a device. The create device must include the country code. This is the country location the device will be installed in long term.

Call:

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2">

<request type="createdevice" xaction\_guid="9cf90928-fabc-4a9f-a22f-c830a90fc009">

<params>

<device\_uri>{61e08b77-df3c-4735-9f3b-0b42efb7bdcf}/{V7777777BX}</device\_uri>

<country>US</country>

<region>United States</region>

</params>

</request>

</message>

**Tag Descriptions**

device\_uri:required – Unique device instance identifier which consists of the device GUID and the device serial number

country:required – ISO 3166 compliant country name

region:optional – Used by Valleylab FT10 and Valleylab LS10 devices. DMP only stores this tag in the device record. – NOTE: Implemented in Sprint D

Response:

On success, a confirmation that the device was created.

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

<message session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<response xaction\_guid="9cf90928-fabc-4a9f-a22f-c830a90fc009" type="ok" />

<params>

<timestamp type=local>UNIX timestamp</timestamp>

</params>

</response>

</message>

On failure, an error response is returned with an indication of reason.

**Reasons**

“Loggin Session Not Available” – Client session is invalid

“Invalid Device Type” – Type of device requested to be created is not known

“User doesn’t have permission to create device” – User permission is missing

“Device already exists” – Server already has a device of this type, configuration and serial number in its database

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<message session\_guid="82f773f9-002a-4053-bcff-664d6939e6a3" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<response xaction\_guid="387f24ca-78c6-47cc-83e0-102397f514ff" type="bad">

<params>

<timestamp type=local>UNIX timestamp</timestamp>

<device\_uri>{7a85f0c9-531e-4754-ad68-04c77ed63657}/{35B12P3001}</device\_uri>

<reason>Generic Create Device Failure Reason</reason>

</params>

</response>

</message>

#### Delete Device – NOT CURRENTLY IMPLEMENTED

The Delete Device command is used to remove a device node, allowing the agent to forget about the existence of a given device. This only causes the Agent to forget about the device, it doesn’t go to the server. Device will still be a registered device on the Server.

Call:

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3'>

<request type='deletedevice' xaction\_guid='55c26878-90fb-4a4a-aa1f-3a689310771e'>

<params>

<device\_uri>{7a85f0c9-531e-4754-ad68-04c77ed63657}/{840 34567453}</device\_uri>

</params>

</request>

</message>

Response:

On success, a confirmation that the device was deleted.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3'>

<response type='ok' xaction\_guid='55c26878-90fb-4a4a-aa1f-3a689310771e'/>

</message>

On failure, an error response is returned with an indication of reason.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='0f16a851-a043-498d-a857-77ee5c59499c'>

<params>

<reason>Error ?????</reason>

</params>

</response>

</message>

#### Get Device List – not implemented

The intent of this message is to get the list of all devices that the agent knows about that are relevant for the logged-in user.

#### Stat Device

The Stat Device command is used to test the assertion that a device is known by the server. If that fails it goes to the RSA. Although known configuration details are returned, perhaps the most critical information provided are the rights for the logged-in user.

Note you must include the country code in the statdevice message, there are software download limitations enforced based on the country. This ensures that the system will comply with local medical device regulations.

Call:

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2">

<request type="statdevice" xaction\_guid="48e9d531-38ea-47f2-a352-27e769594d81">

<params>

<device\_uri>{61e08b77-df3c-4735-9f3b-0b42efb7bdcf}/{V7777777BX}</device\_uri>

<country>US</country>

<format>named\_configuration</format>

</params>

</request>

</message>

**Tag Descriptions**

1. <country>:required – used to help determine what SW packages to send to the Client
2. <format>:optional – value:named\_configuration. Used to request the named configuration format of the last known device configuration

Response:

On success, confirmation that a device is known to system, the last known configuration, and user access rights.

The following response is for standard catalog-based configurations.

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<message session\_guid="9bb8b389-6591-4f66-b97b-c0616126d3a0" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<response xaction\_guid="7d79bad2-9368-489d-9496-99a80bf22016" type="statdevice">

<params>

<device\_uri>{7a85f0c9-531e-4754-ad68-04c77ed63657}/{35B12P3001}</device\_uri>

<country>US</country>

</params>

<notifications>

<notification oid="809fa71a-60d6-4c62-a121-6ac2bfe8bbf9">

<header notification\_type="response" notification\_action="get">

<pertinent\_identifier>35B12P3001</pertinent\_identifier>

<pertinent\_type>7a85f0c9-531e-4754-ad68-04c77ed63657/4bee9772-adcb-4ec2-b944-3191492b3436</pertinent\_type>

</header>

<body>

<is\_Accessible>false</is\_Accessible>

<last\_update>Tue Nov 13 17:04:17 PST 2012</last\_update>

<components>

<component type="hardware">

<name>MixAirFlowSensor</name>

<part\_number>840610</part\_number>

<rights>

<right>install</right>

<right>remove</right>

</rights>

<serial\_number>6101233040</serial\_number>

<revision>55</revision>

</component>

</components>

<features>

<feature type="availablefeature">

<feature\_id>10</feature\_id>

<feature\_name>f1</feature\_name>

<feature\_desc> desc </feature\_desc>

</feature>

<feature type="entitledfeature">

<feature\_id>10</feature\_id>

<feature\_name>f1</feature\_name>

<start\_date> timestamp </start\_date>

<end\_date> timestamp </end\_date>

<feature\_duration>30</feature\_duration>

<feature\_term>[0|1|2]</feature\_term>

<license\_key></license\_key>

<feature\_desc> desc </feature\_desc>

<sw\_upgrade\_required>[true|false]</sw\_upgrade\_required>

</feature>

<feature type='enabledfeature'>

<feature\_id>1</feature\_id>

<feature\_name>gzip</feature\_name>

<start\_date> timestamp </start\_date>

<end\_date> timestamp </end\_date>

<feature\_duration>30</feature\_duration>

<feature\_term>0</feature\_term>

<license\_key></license\_key>

<feature\_desc>desc</feature\_desc>

</feature>

</features>

</body>

</notification>

</notifications>

</response>

</message>

**Tag Description**

See Section 12.6.1.3, Sync Device Config for tag descriptions.

Named configuration response:

<?xml version="1.0" encoding="UTF-8"?>

<message session\_guid="26a1d0f5-3996-49de-a133-b9d1e7ae4f61" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<request xaction\_guid="c67a7a12-f291-4f02-a2ea-63f5bc83b5a8" type="ok">

<params>

<componentChecksum>64401269460f0d0763486d4944f59a38</componentChecksum>

<country>US</country>

<device\_uri>{a7a65225-ef2d-48e9-89c8-6975dd7dc054}/{35B1234567}</device\_uri>

</params>

<notifications>

<notification>

<body>

<system\_config type="running\_config">

<config type="SOFTWARE">

<component>

<name>pb980sw1</name>

<revision>v1</revision>

<part\_number>s1</part\_number>

</component>

<component>

<name>ttts</name>

<revision>v3</revision>

<part\_number>yy</part\_number>

<crc>cedf</crc>

</component>

</config>

<config type="HARDWARE">

<component>

<name>pb980hw1</name>

<revision>v1</revision>

<part\_number>p1</part\_number>

<serial\_number>35B1234567</serial\_number>

</component>

<component>

<name>WINTEL</name>

<revision>4.0</revision>

<part\_number>ABC</part\_number>

<serial\_number>abcdefg</serial\_number>

</component>

</config>

<config type="FIRMWARE">

<component>

<name>Amplifier</name>

<revision>4.0</revision>

<part\_number>ABC</part\_number>

</component>

</config>

</system\_config>

<features>

<feature type="availablefeature">

<feature\_id>10</feature\_id>

<feature\_name>f1</feature\_name>

<feature\_desc> desc</feature\_desc>

</feature>

<feature type="entitledfeature">

<feature\_id>10</feature\_id>

<feature\_name>f1</feature\_name>

<start\_date> timestamp</start\_date>

<end\_date> timestamp</end\_date>

<feature\_duration>30</feature\_duration>

<feature\_term>[0|1|2]</feature\_term>

<license\_key></license\_key>

<feature\_desc> desc</feature\_desc>

<sw\_upgrade\_required>[true|false]</sw\_upgrade\_required>

</feature>

<feature type='enabledfeature'>

<feature\_id>1</feature\_id>

<feature\_name>gzip</feature\_name>

<start\_date> timestamp</start\_date>

<end\_date> timestamp</end\_date>

<feature\_duration>30</feature\_duration>

<feature\_term>0</feature\_term>

<license\_key></license\_key>

<feature\_desc>desc</feature\_desc>

</feature>

</features>

<body/>

</notification>

<notifications>

</request>

</message>

On failure, an error response is returned with an indication of reason.

**Reasons**

“Loggin Sesstion Not Available” – no user is logged in.

“Invalid Device Type” – Device type in stat request is not recognized.

“” – Device configuration data is missing (TODO: StatDevice:statDeviceInfo, line 93)

“Device not found” – Device not found in server database

“Failed - Device has no trade embargo exception” – Device has reported a trade embargoed country, but it does not have a trade embargo exception, allowing the DMP to update it.

“User is unknown” – User does not exist on Agent or on Server

“Request not valid” – Some tag or tags do not have valid information

“User Session not valid” – There is no login session for this user

“Invalid Device type” – This system does not recognize the device type sent to it. Could be Client mismatch.

“Device information not found” – DMP server does not have a last known configuration for this device

“User has no permission to view current Device Type” – User has no device access on Server, nor on Agent

“Failed - Device not found” –  **–** The device instance specified in the request was not found in the database.

“no last known configuration” –

“System Error: Try re-docking the device or contact DMP service.” – An internal DMP error occurred while servicing this message.

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

<message session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<response xaction\_guid="48e9d531-38ea-47f2-a352-27e769594d81" type="bad">

<params>

<device\_uri>{61e08b77-df3c-4735-9f3b-0b42efb7bdcf}/{V7777777BX}</device\_uri>

<reason>Failed - Device not found</reason>

</params>

</response>

</message>

### Notification Management

This section describes the general notification management messages and their responses. These messages manage the mailboxes which PostNotification messages are maintained per device instance. The messages described below are generic in nature, and are provided to give a sense of the various tags used per message type.

#### Get (Headers) Notification

The GetHeaders message is used by clients which Service Clients whose configuration is defined on the Server using DMP 1.1+ Catalog Configuration scheme. It allows the client application to ask the RSA if it has any new notifications for the Client regarding a particular device. This is very much like asking if the device has mail. So naturally the Client identifies the device serial number (device name) in the getheaders message. The RSA will respond with the list of notifications that are specific to the device with the given serial number.

NOTE: This message must not be issued directly after sending DeviceInfo message. DeviceInfo message may kick off package downloads as a result of examining the device running configuration. Therefore, the Client should first set the GetStatus message first to ensure that the Agent is in the idle state prior to sending GetHeaders.

Call:

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2">

<request type="getheaders" xaction\_guid="470a858b-6d8c-45e6-be99-76b9d082c074">

<params>

<device\_uri>{61e08b77-df3c-4735-9f3b-0b42efb7bdcf}/{V7777777BX}</device\_uri>

</params>

</request>

</message>

Response:

On success, a list of notifications that is available for a given device serial number is returned. This list points to the items in the inbox for this specific device. It contains descriptions of all packages that the RSA/RSS found for the device. To produce this list, the RSA/RSS processed the device configuration details contained in the deviceinfo message which the Client sent earlier.

NOTE: The <components> tag and sub-tags are optional. Clients may parse this information in order to know what is actually contained in the OIDs and more efficiently populate package selection lists. Legacy clients may ignore these tags and retrieve the OIDs directly to determine what is available.

NOTE: The component type, document, contains not only the information about which OID to use to retrieve the document, it also contains the MD5 checksum of the software file with which it is associated. By re-searching the get headers response again for this MD5 checksum, the Client can discover how to associate what software is associated with the document. There may be 0..N software packages associated with a document.

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

<message session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<response xaction\_guid="470a858b-6d8c-45e6-be99-76b9d082c074" type="headers">

<notes>

<note oid="3fde9c7583d8e2ac">

<header notification\_type="response" notification\_action="get">

<pertinent\_identifier>V7777777BX</pertinent\_identifier>

<pertinent\_type>61e08b77-df3c-4735-9f3b-0b42efb7bdcf/24637560-edb3-4961-b17d-14e74f74457c</pertinent\_type>

</header>

<components>

<component type="software">

<name>BdSoftware</name>

<revision>B</revision>

<md5>14bc6e8db8b7b94cf7a0ec5080d007f9</md5>

<part\_number>10130385</part\_number>

<language>English</language>

<features>

<feature>

<feature\_id>10</feature\_id>

<feature\_name>f1</feature\_name>

<feature\_desc> desc</feature\_desc>

</feature>

</features>

</component>

</components>

</note>

<note oid="3fb5854e947673b0">

<header notification\_type="response" notification\_action="get">

<pertinent\_identifier>V7777777BX</pertinent\_identifier>

<pertinent\_type>61e08b77-df3c-4735-9f3b-0b42efb7bdcf/24637560-edb3-4961-b17d-14e74f74457c</pertinent\_type>

</header>

<components>

<component type="software">

<name>BdSoftware</name>

<revision>G</revision>

<md5>9bb1f8ba908e32f0dbb4811aa4da0729</md5>

<part\_number>10130385</part\_number>

<language>English</language>

</component>

</components>

</note>

<note oid="3f9bede9974c2a60">

<header notification\_type="response" notification\_action="get">

<pertinent\_identifier>V2214314AX</pertinent\_identifier>

<pertinent\_type>61e08b77-df3c-4735-9f3b-0b42efb7bdcf/c250c8db-b532-4d18-8956-420c3d637a41</pertinent\_type>

</header>

<body>

<is\_Accessible>false</is\_Accessible>

<components>

<component type="document">

<name>Operator and Service Manual SW Rev 1.11.02</name>

<md5>84dbc6043d706c629317ca7d4cb1adbd</md5>

<softwares>

<software>

<md5>9bb1f8ba908e32f0dbb4811aa4da0729</md5>

</software><software>

<md5>14bc6e8db8b7b94cf7a0ec5080d007f9</md5>

</software>

</softwares>

</component>

</components>

</body>

</note>

</notes>

</response>

</message>

On failure, an error response is returned with an indication of reason.

**Reasons**

“Downloading in Progress” – rule for ESS which disallows asking for OID headers if the system is still downloading packages.

“No message in inbox” – there are no message headers to return

“Device not found” – the device to which the message refers does not exist, or Agent is not aware of it.

“User doesn’t have permission to get notifications” – User does not have permission to download packages

“User does not have device privilege” – User does not have access rights to the device which headers are requested

“Vessel Sealing Country not available” – Device is vessel sealing or ablation CoT, and device country is not provided

“Country code violation” – device country is set to a trade embargoed country.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='0f16a851-a043-498d-a857-77ee5c59499c'>

<params>

<reason>Error ?????</reason>

</params>

</response>

</message>

#### Get Notification

All legacy Clients will use this message, regardless of DMP version. This message is equivalent to the Client asking for a specific item in the inbox for a specific device instance. The Client sends the notification\_oid listed in the GetHeaders response to identify the notification it wants to read.

Call:

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2">

<request type="getnotification" xaction\_guid="bfc33531-7d9a-4d30-9245-4fa001f28385">

<params>

<notification\_oid>3fde9c7583d8e2ac</notification\_oid>

</params>

</request>

</message>

Response:

On success, a notification with both body and header is sent. The message includes the notification\_oid for easy cross checking of the resulting notification message with the request. The body of the message includes information about the software or document available.

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

<message session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<response xaction\_guid="bfc33531-7d9a-4d30-9245-4fa001f28385" type="notifications">

<notifications>

<notification oid="3fde9c7583d8e2ac">

<header notification\_type="response" notification\_action="get">

<pertinent\_identifier>V7777777BX</pertinent\_identifier>

<pertinent\_type>

61e08b77-df3c-4735-9f3b-0b42efb7bdcf/24637560-edb3-4961-b17d-14e74f74457c</pertinent\_type>

</header>

<body>

<is\_Accessible>false</is\_Accessible>

<components>

<component type="software">

<name>Control</name>

<revision>01.10.en</revision>

<uri><https://rssqa-app01.covidien.com/software/1367566802_raven_01-10-en.bin.zip></uri>

<location>

C:\Program Files\Covidien\Device Management Agent\Cache\raven\_01-10-en.bin

</location>

<file\_size>4830250</file\_size>

<md5>cdac5e53145c2f45f2943a8a686845e700b8901f</md5>

<part\_number>2222222</part\_number>

<language>US English</language>

<status>update</status>

<nid>676</nid>

<softwareStatus>In Production</softwareStatus>

</component>

</components>

</body>

</notification>

</notifications>

</response>

</message>

Below is a different response message that contains document information.

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

<message session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<response xaction\_guid="1bae7774-a30d-4f6c-8915-959b0777ff0b" type="notifications">

<notifications>

<notification oid="3fe77219c81f6257">

<header>

<pertinent\_identifier>V7777777BX</pertinent\_identifier>

<pertinent\_type>

61e08b77-df3c-4735-9f3b-0b42efb7bdcf/c250c8db-b532-4d18-8956-420c3d637a41

</pertinent\_type>

</header>

<body>

<is\_Accessible>false</is\_Accessible>

<components>

<component type="document">

<name>IntegratedTestCasesSCDUpdater.pdf</name>

<uri>https://rssqa-app01.covidien.com/document/servicemanual/1367941501\_IntegratedTestCasesSCDUpdater.pdf.zip</uri>

<location>C:\Program Files\Covidien\Gateway Agent\Cache\3fcf8bc9fdff39f0Cipher.zip</location>

<decrypt\_location>C:\Program Files\Covidien\Gateway Agent\Cache\1367941501\_IntegratedTestCasesSCDUpdater.pdf</decrypt\_location>

<file\_size>415658</file\_size>

<md5>1c1121dbf9db125709e1146a23026c204c08bf5e</md5>

<status>update</status>

<documentType>Other</documentType>

</component>

<component type="software">

<name>Control</name>

<part\_number>0</part\_number>

<revision>01.08.01</revision>

</component>

<component type="hardware">

<name>Control Board</name>

<part\_number>0</part\_number>

<revision>0</revision>

</component>

</components>

</body>

</notification>

</notifications>

</response>

</message>

On failure, an error response is returned with an indication of reason.

**Reasons**

“Request was not the correct type” – Client requested an OID type that was not supported

“No notification found” – the OID requested by the Client was not present in the inbox

“Bad notification\_Oid” – either the OID itself was bad, or the inbox file location specified by the OID is bad

“No message in outbox” – Couldn’t find outbox, or no message was found in the outbox

“Bad notification\_Oid” – Notification OID in params argument to getInboxMessage() or getOutboxMessage() is bad

“Bad notification\_Oid” – Notification OID sent into getMailMessageByOid() is bad

“Software not found” – could not find encrypted software file

“Downloaded file md5 not matched” – the encrypted software file MD5 checksum does not match the expected value

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='0f16a851-a043-498d-a857-77ee5c59499c'>

<params>

<reason>Error ?????</reason>

</params>

</response>

</message>

On failure due to a change in the device , an error response is returned indicating a change in state of the device.

The params will also contain a device\_status from the RSA indicating that a software install cannot be accomplished without a new connection to the server. (This message will only occur if an installation is attempted while disconnected from the server.)

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='0f16a851-a043-498d-a857-77ee5c59499c'>

<params>

<device\_status>conflicted</device\_status>

<reason>Error: The Device has changed configuration since software download and software installation. </reason>

</params>

</response>

</message>

#### Post Notification

See Section 12.6, Sample Messages / Post Notifications

#### Delete Notification

Mark a notification deleted.

Call:

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3'>

<request type='deletenotification' xaction\_guid='55c26878-90fb-4a4a-aa1f-3a689310771e' >

<params>

<notification\_oid>bc108d24-1cd7-403f-8dd4-e8e0ad816a20</notification\_oid>

</params>

</request>

</message>

Response:

On success, a confirmation that the notification was deleted.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3'>

<response type='ok' xaction\_guid='55c26878-90fb-4a4a-aa1f-3a689310771e'/>

</message>

On failure, an error response is returned with an indication of reason.

**Reasons**

"Request was not appropriate" – message fields may be null, message type is wrong, or session may be invalid

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='0f16a851-a043-498d-a857-77ee5c59499c'>

<params>

<reason>Error ?????</reason>

</params>

</response>

</message>

#### Update Notification – NOT Currently Used

The RSA design supports making changes to a notification that was previously sent. This is not currently used.

Call:

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<request type='updatenotification'

xaction\_guid='55c26878-90fb-4a4a-aa1f-3a689310771e'>

<params>

<notification\_oid='344aa359-f0ba-4f5a-b6d2-790bfd1ad95e'>

<header notification\_type='command' notification\_action='set'>

<pertinent\_type>ddf96625-cb25-49c2-a59b-07da0f48d46c/86fd6b83-219a-4482-8813-191c477ccd7e</pertinent\_type>

<pertinent\_identifier>840 351545678</pertinent\_identifier>

<notification\_id>344aa359-f0ba-4f5a-b6d2-790bfd1ad95e</notification\_id>

</header>

<body />

</notification>

</params>

</request>

</message>

Response:

On success, confirmation that a notification has been updated.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3'>

<response type='ok' xaction\_guid='55c26878-90fb-4a4a-aa1f-3a689310771e'/>

</message>

On failure, an error response is returned with an indication of reason.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='0f16a851-a043-498d-a857-77ee5c59499c'>

<params>

<reason>Error ?????</reason>

</params>

</response>

</message>

#### Expunge (Notification)

Call:

Remove a single notification marked for deletion.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3'>

<request type='expunge' xaction\_guid='55c26878-90fb-4a4a-aa1f-3a689310771e' >

<params>

<notification\_oid>bc108d24-1cd7-403f-8dd4-e8e0ad816a20</notification\_oid>

</params>

</request>

</message>

Remove notifications marked deletion from system.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3'>

<request type='expunge' xaction\_guid='55c26878-90fb-4a4a-aa1f-3a689310771e' />

</message>

Response:

On success, a confirmation that the notification was undeleted is returned.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3'>

<response type='ok' xaction\_guid='55c26878-90fb-4a4a-aa1f-3a689310771e'/>

</message>

On failure, an error response is returned with an indication of reason.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='0f16a851-a043-498d-a857-77ee5c59499c'>

<params>

<reason>Error ?????</reason>

</params>

</response>

</message>

### Post Notifications

Post Notification messages are all about informing the DMP system regarding some aspect of the docked device. There are several types of Post Notification. Each will be called out here along with their responses.

#### Device Configuration

##### Device Info

The Device Info Post Notification message is one of three messages which inform the server of the current state of the configuration of the docked device. The Device Info post notification informs the server as to the different hardware, software and firmware items which make up the device. This information includes name, part number, and revision of the item. These attributes must match those specified in the corresponding item catalog in the DMP server.

Call:

Here we are telling the RSA about the configuration of the medical device connected to the client application. This will cause the RSA/RSS to compare this configuration with the production software and documentation to determine the best match for a upgrade or if the system is already at the current version.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2">

<request type="postnotification" xaction\_guid="6e77fa94-67df-419c-82a1-2b64244d7f25">

<params>

<device\_uri>{61e08b77-df3c-4735-9f3b-0b42efb7bdcf}/{V7777777BX}</device\_uri>

<country>US</country>

<region>United States</region>

</params>

<notification oid="1e3af352-d3e4-4abc-a619-cbdbe128760c">

<header notification\_type="event" notification\_action="create">

<pertinent\_type>61e08b77-df3c-4735-9f3b-0b42efb7bdcf/532ffdda-6f38-41da-9a40-cb0801e6695f</pertinent\_type>

<pertinent\_identifier>V7777777BX</pertinent\_identifier>

</header>

<body>

<components>

<component type="HARDWARE">

<name>Control Board</name>

<part\_number>9999999</part\_number>

<revision>9</revision>

</component>

<component type="SOFTWARE">

<name>Control</name>

<part\_number>2222222</part\_number>

<revision>01.08.en</revision>

</component>

</components>

</body>

</notification>

</request>

</message>

<message schema\_version='3644767c-2632-411a-9416-44f8a7dee08e'

session\_guid='9860f123-cfe9-420f-884f-83348a95d240'>

<request type='postnotification' xaction\_guid='c67a7a12-f291-4f02-a2ea-63f5bc83b7e9'>

<params>

<device\_uri>{3B682913-6D1E-4355-9E48-208EB7061A3D}/{T1I24851EX }</device\_uri>

<country>US</country >

<region>North America</region>

</params>

<notification oid='dc308fde-bf9a-4c6f-9ef0-f41cb499c8e1'>

<header notification\_type='event' notification\_action='create'>

<pertinent\_type>3B682913-6D1E-4355-9E48-208EB7061A3D/532ffdda-6f38-41da-9a40-cb0801e6695f</pertinent\_type>

<pertinent\_identifier>3B682913-6D1E-4355-9E48-208EB7061A3D</pertinent\_identifier>

</header>

<body>

<components>

<component type="SOFTWARE ">

<name>Build</name>

<part\_number>0</part\_number >

<revision>XX.YY.ZZ</revision>

</component>

<component type="SOFTWARE">

<name>Host Bootloader</name>

<part\_number>XXXXXX</part\_number >

<revision>X.Y</revision >

</component>

<component type="SOFTWARE">

<name>Host App</name>

<part\_number>XXXXXXX</part\_number>

<revision>X.Y</revision>

</component>

<component type="SOFTWARE">

<name>DSP Bootloader</name>

<part\_number>XXXXXX</part\_number >

<revision>X.Y</revision >

</component>

<component type="SOFTWARE">

<name>DSP App</name>

<part\_number>XXXXXX0</part\_number >

<revision>X.Y8.3</revision >

</component>

<component type="SOFTWARE">

<name>VIBE Bootloader</name>

<part\_number>XXXXXX0</part\_number >

<revision>X.Y1.0024</revision >

</component>

<component type="SOFTWARE">

<name>VIBE App</name>

<part\_number>XXXXXX</part\_number >

<revision>X.Y</revision >

</component>

<component type="HARDWARE">

<name>Main PCBA</name>

<part\_number>XXXXXX</part\_number>

<revision>Z</revision>

<serial\_number>YYYYYYY</serial\_number>

</component>

<component type="HARDWARE">

<name>VIBE</name>

<part\_number>XXXXXX</part\_number>

<revision>X</revision>

<serial\_number>YYYYYYY</serial\_number>

</component>

</components>

</body>

</notification>

</request>

</message>

Response:

The xaction guid is your receipt that the system received the message in proper format and that it will process the client request.

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

<message session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<response xaction\_guid="6e77fa94-67df-419c-82a1-2b64244d7f25" type="ok" />

</message>

On failure, an error response is returned with an indication of reason. While failures typically mean that the message was not handled successfully and the reason expressed in the response relates directly to the error. However, in this case, the response to a device info message may well still result in SW packages being delivered to the Client. This is due to the Bad response reasons indicating that a configuration in whole may not be

**Reasons**

“Failed - Device not found” – Device instance not found in the database. Client should first use StatDevice and CreateDevice.

“Invalid user” – User/session is not found.

“Invalid Country” – Country is not found in DMP country list

“No components found” – The components sent in the request are not in the HW/SW catalogs

“User Session not valid” – Request is sent with invalid session

“Invalid Device type” – Device type is not currently being managed by the DMP. Check Client version.

“User doesn't have permission on device” – SW is available, but user is not trained on this device.

“Failed because of exception ...” – Request failed due to DMP internal logic error.

“Message format error” – Message is missing one or more required tags

“Not valid type” -

“User is unknown” – User is not registered with the DMP

“Request not valid” – One or more tags has incorrect information for this message request.

“User dosen't have permission to create device” – User does not have the correct device access permission

“Country code violation for the device” – Device reports a country which is in the Trade Embargo list. No SW is provided.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='0f16a851-a043-498d-a857-77ee5c59499c'>

<params>

<reason>Error ?????</reason>

</params>

</response>

</message>

##### Agent/Client Update Message

This message is used by the Agent (referred to as the “Client” below) or the Client to request a newer version of the Client, or newer versions of the business rules associated with the Client. The workflow is very similar to the DeviceInfo/GetHeaders workflow. As such, the Client Update workflow only uses Catalog Configuration, and not Named Configuration workflows.

The Client Update Message should be sent from the Client application to the Agent after the user has logged in.

If the response to the Client is “ok,” then the Client should issue GetStatus until the download\_time tag for overall package download status, or remaining\_time tag for the device type GUID specified in the Client Update Message is 0. Client should then issue GetHeaders to retrieve the available new packages for the Client. If GetHeaders returns packages/business rules, the Client should send a GetNotification message with the OID of the package required for each Client data file. The Agent will then unencrypt the package, and return its location and metadata in the GetNotification(OID) response.

As mentioned above, the Client Update message is similarly constructed as a “device info message” in that the Client identifies itself. It may do so in one of two ways, depending on the workflow/process used by the CoT for storing Client update installer and Client Business Rules. The message contents must match the CoT process for uploading Business Rules.

1. Store Data in Client SW Catalog
   1. Use the Client GUID as the “device type” identifier ***(PREFERRED)***
   2. NOTE: Client “GUID” for the Agent is the string, “ADMINISTRATIVE\_AGENT”
2. Store Client Business Rules in the Device SW Catalog (Legacy Client Support)
   1. Use the Device GUID as the “device type” identifier.

For this message, the body and components tags are no longer required. They can be left in for legacy Clients. Reason being that the point of this message is to always retrieve the latest version of every uniquely named Business Rule item in the “device type” SW Catalog.

1. Query all items in the Client SW Catalog (specified by GUID in the message)
2. Sort on Item name.
3. For each unique item name
   1. Add to the list the item with the latest version determined by the comparison order field of the items
4. Compare list from item 3 to those in the current cache for the Client.
   1. Download those items which are missing
5. On download complete
   1. Add those items downloaded to cache index.
   2. Remove those items from the cache index for which there was a new version downloaded.
6. Put list from item 3 into mailbox for Client for later retrieval by GetHeaders and GetNotification(OID)

The agent will forward this to the server if it is connected. The Client application should always send this message regardless of server connection status.

**Tag Descriptions**

1. pertinent\_type: required – this describes the entity for the “device type” which the message is intended.
2. body,components,component: optional – for backward compatibility. They have no effect on the handling of the message.

Request:

**Early Developed Client Request**

Uses Device GUID to retrieve data from the Device SW Catalog

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2">

<request type="postnotification" xaction\_guid="8073cfca-aa6d-4178-b9ce-5b4529bf8458">

<params />

<notification oid="235464f8-c3d3-4c41-886d-7acf70f7b90c">

<header notification\_type="event" notification\_action="create">

<pertinent\_type>61e08b77-df3c-4735-9f3b-0b42efb7bdcf/APPLICATION\_CLIENT\_INFO</pertinent\_type>

</header>

<body>

<components>

<component type="Software">

<name>SCDUInstaller</name>

<revision>0.6</revision>

<part\_number>12345688</part\_number>

</component>

<component type="Business Rules">

<name>SCDUBusinessRules</name>

<revision>1.0</revision>

<part\_number>77777</part\_number>

</component>

</components>

</body>

</notification>

</request>

</message>

**Preferred Client Request**

Uses Client GUID to retrieve data from the Client SW Catalog

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2">

<request type="postnotification" xaction\_guid="8073cfca-aa6d-4178-b9ce-5b4529bf8458">

<params />

<notification oid="235464f8-c3d3-4c41-886d-7acf70f7b90c">

<header notification\_type="event" notification\_action="create">

<pertinent\_type>61e08b77-df3c-4735-9f3b-0b42efb7bdcf/APPLICATION\_CLIENT\_INFO</pertinent\_type>

</header>

</notification>

</request>

</message>

If Client update data is available, the following message will be sent to the Agent from the Server. If there is no update available, then the <components> tag will be empty.

Response from Server to Agent:

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

<message session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<response xaction\_guid="8073cfca-aa6d-4178-b9ce-5b4529bf8458" type="notifications">

<notifications>

<notification oid="3fb17ec329b860a8">

<header notification\_type="response" notification\_action="get">

<pertinent\_identifier>SCD 700</pertinent\_identifier>

<pertinent\_type>61e08b77-df3c-4735-9f3b-0b42efb7bdcf/APPLICATION\_CLIENT\_INFO</pertinent\_type>

</header>

<body>

<is\_Accessible>false</is\_Accessible>

<components>

<component type="Software">

<name> SCDUInstaller </name>

<revision>2.0</revision>

<uri>https://rssqa-app01.covidien.com/software/1370981101\_SCD700\_Updater\_V2.config.zip</uri>

<location>C:\Program Files\Covidien\ Device Management Agent\Cache\SCD700\_Updater\_V2.config</location>

<file\_size>33632</file\_size>

<md5>K1349b673da93504f5c6f077f2aaa6d7052c2bcc</md5>

<part\_number>12345688</part\_number>

<status>update</status>

<nid>9817</nid>

<softwareStatus>In Production</softwareStatus>

</component>

<component type="Business Rules">

<name>SCDUBusinessRules</name>

<revision>2.0</revision>

<uri>https://rssqa-app01.covidien.com/software/1370981101\_SCD700\_Updater\_V2.config.zip</uri>

<location>C:\Program Files\Covidien\ Device Management Agent\Cache\SCD700\_Updater\_V2.config</location>

<file\_size>3632</file\_size>

<md5>d1349b673da93504f5c6f077f2aaa6d7052c2bcc</md5>

<part\_number>77777</part\_number>

<status>update</status>

<nid>3817</nid>

<softwareStatus>In Production</softwareStatus>

</component>

</components>

</body>

</notification>

</notifications>

</response>

</message>

If a successful response is received from the Server, then the Agent responds to the Client with a simple “ok” response.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='535cd3f1-84b7-42d9-b9f6-7ed4e78580e2' >

<response type='ok' xaction\_guid='8073cfca-aa6d-4178-b9ce-5b4529bf8458'>

</response>

</message>

On failure, an error response is returned to the Agent with an indication of reason. This message is forwarded to the Client.

**Reasons**

“No Software Updates Found” – no Client software updates, nor business rules found

“agent is running in disconnected mode” – Agent must be connected to the server to update itself.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='535cd3f1-84b7-42d9-b9f6-7ed4e78580e2' >

<response type='bad' xaction\_guid='8073cfca-aa6d-4178-b9ce-5b4529bf8458'>

<params>

<reason>Error ?????</reason>

</params>

</response>

</message>

##### Sync Device Config

Sync Device Config is the Named Configuration version of Device Info postnotification. This

<?xml version="1.0" encoding="UTF-8"?>

<message session\_guid="26a1d0f5-3996-49de-a133-b9d1e7ae4f61" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<request xaction\_guid="c67a7a12-f291-4f02-a2ea-63f5bc83b5a8" type="postnotification">

<params>

<country>US</country>

<region>United States</region>

<facility> Boulder Hospital </facility>

<address> san Francisco </address>

<applied\_config>namedSysCfg01</applied\_config>

</params>

<notification oid="dc308fde-bf9a-4c6f-9ef0-f41cb499c8e1">

<header notification\_type="event" notification\_action="syncDeviceCfg">

<pertinent\_type>@GUID@/532ffdda-6f38-41da-9a40-cb0801e67dfa</pertinent\_type>

<pertinent\_identifier>@SN@</pertinent\_identifier>

</header>

<body>

<system\_config type="running\_config">

<config type="SOFTWARE">

<component>

<name>pb980sw1</name>

<revision>v1</revision>

<part\_number>s1</part\_number>

</component>

<component>

<name>ttts</name>

<revision>v3</revision>

<part\_number>yy</part\_number>

<crc>cedf</crc>

</component>

</config>

<config type="HARDWARE">

<component>

<name>pb980hw1</name>

<revision>v1</revision>

<part\_number>p1</part\_number>

<serial\_number>35B1234567</serial\_number>

</component>

<component>

<name>WINTEL</name>

<revision>4.0</revision>

<part\_number>ABC</part\_number>

<serial\_number>abcdefg</serial\_number>

</component>

</config>

<config type="FIRMWARE">

<component>

<name>Amplifier</name>

<revision>4.0</revision>

<part\_number>ABC</part\_number>

</component>

</config>

</system\_config>

<body/>

</notification>

</request>

</message>

<component>

<name>WINTEL</name>

<revision>4.0</revision>

<part\_number>ABC</part\_number>

<serial\_number>abcdefg</serial\_number>

</component>

</config>

<config type="FIRMWARE">

<component>

<name>Amplifier</name>

<revision>4.0</revision>

<part\_number>ABC</part\_number>

</component>

</config>

</system\_config>

<body/>

</notification>

</request>

</message>

**Tag Descriptions**

country: required – country where device is installed

region:optional – Used by Valleylab FT10 and Valleylab LS10 devices. DMP only stores this tag in the device record.

facility:optional – facility name where device is installed

address:optional – street address where device is installed

applied\_config:optional – name of system configuration installed on device

name:required – name of configuration item

revision:required – revision of configuration item

part\_number:required – part number of configuration item

serial\_number:optional – serial number of HW item installed in device

warning:optional – an extended status which further explains the status of the configuration of the device.

system\_config:required – contains a list of HW, SW and FW configurations

config:required – contains a list of components or HW, SW, and FW items. Tag attribute, type, contains the values Hardware, Software or Firmware.

component:required – contains meta information for a HW, SW or FW item

If the request is successful, the following response is returned to the Client:

<?xml version="1.0" encoding="UTF-8"?>

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid="bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3">

<response type="ok" xaction\_guid="55c26878-90fb-4a4a-aa1f-3a689310771e" />

<warning>this running configuration is wrong-software </warning>

</message>

**Warning Messages**

"Device running software [ComponentName] v[SoftwareRevision] and device country are in conflict for country exclusion." – Device country is in the country exclusion list of the installed software.

"Device country is unknown. Country exclusion cannot be checked." – Device country is not in the list of known countries

"Failed to get running configuration. Country exclusion cannot be checked." -- Can’t find a valid running config

“It's a warning configuration because of invalid hardware combination. Please make hardware changes in order to get the device into a production status.” – Self-explanatory

“It's a warning configuration because of software incompatible with hardware. Please check and upgrade software packages to get the device into a production status.” – Self-explanatory

"It's a warning configuration because of invalid software combination. Please check and upgrade software packages to get the device into a production status." – Self-explanatory

“This device shall never be upgraded due to administrative purpose” – CoT Admin has marked this configuration as not to be upgraded.

**NOTE:** One and only one warning will be returned. Client should re-issue this message until user has updated the device such that there are no message responses with warning config status.

If the request was unsuccessful, the following response is returned to the Client:

<?xml version="1.0" encoding="UTF-8"?>

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid="bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3">

<response type="bad" xaction\_guid="55c26878-90fb-4a4a-aa1f-3a689310771e" />

<reason>Error Reason</reason>

</message>

**Reasons**

“Failed - Device not found” – The device instance specified in the request was not found in the database.

“Invalid user” – User is not a registered user, or has been deleted.

“Invalid Country” – Country specified in the request is not in the DMP country list

“System Error: Try re-docking the device or contact DMP service.” – An internal DMP error occurred while servicing this message.

“The request message is invalid.” – Some tag or tags in the request were not filled in properly.

“No running configuration in the request from client.” – The system could not find the tags which make up the running configuration.

“The running configuration from client is empty.” – The system could not find the tags which make up the running configuration.

##### Get Matched Configs

This message is the Named Configuration equivalent of GetHeaders. It returns the system configurations which match the device hardware. This message is handled only by the Agent. It is not sent to the Server. This means that it can be used to “test” a device configuration without updating the device record.

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<message session\_guid=*"26a1d0f5-3996-49de-a133-b9d1e7ae4f61"* schema\_version=*"3644767c-2632-411a-9416-44f8a7dee08e"*>

<request xaction\_guid=*"c67a7a12-f291-4f02-a2ea-63f5bc83b5a8"* type=*"postnotification"*>

<notification oid=*"dc308fde-bf9a-4c6f-9ef0-f41cb499c8e1"*>

<header notification\_type=*"event"* notification\_action=*"getMatchedCfg"*>

<pertinent\_type>@GUID@/532ffdda-6f38-41da-9a40-cb0801e67dfa</pertinent\_type>

<pertinent\_identifier>@SN@</pertinent\_identifier>

</header>

<body>

<system\_config type="running\_config">

<config type="SOFTWARE">

<component>

<name>pb980sw1</name>

<revision>v1</revision>

<part\_number>s1</part\_number>

</component>

<component>

<name>ttts</name>

<revision>v3</revision>

<part\_number>yy</part\_number>

<crc>cedf</crc>

</component>

</config>

<config type="HARDWARE">

<component>

<name>pb980hw1</name>

<revision>v1</revision>

<part\_number>p1</part\_number>

<serial\_number>35B1234567</serial\_number>

</component>

<component>

<name>WINTEL</name>

<revision>4.0</revision>

<part\_number>ABC</part\_number>

<serial\_number>abcdefg</serial\_number>

</component>

</config>

<config type="FIRMWARE">

<component>

<name>Amplifier</name>

<revision>4.0</revision>

<part\_number>ABC</part\_number>

</component>

</config>

</system\_config>

<body/>

</notification>

<params>

<country>US</country>

<region>United States</region>

</params>

</request>

</message>

**Tag Descriptions**

1. name:required – name of system configuration which matches the device running configuration
2. warning:optional – contains information about incompatibilities in the running configuration
3. status:required – contains the values “In Production”, “Limited Release”, “Archived” or “Obsolete”. This describes the operational state of a SW/FW package.

For the other tags, see section 12.7.1.3, Sync Device Config

If the request is successful, the following response is returned to the Client:

|  |
| --- |
| <? xml version="1.0" encoding="UTF-8"?>  <message session\_guid="4bcfca69-a819-42d8-b84b-56b9c520dd64" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">  <response xaction\_guid="c67a7a12-f291-4f02-a2ea-63f5bc83b7e9" type="ConfigurationNotification">  <notifications>  <notification>  <header notification\_type="response" notification\_action="get" /> |
| <devicetype\_guid> abcdefg </devicetype\_guid> |
| <system\_config type="normal"> |
| <name>sys\_config01</name> |
| <revision>v1</revision> |
| <description>this is a named system config</description> |
| <update\_time>01/10/2013</update\_time> |
| <checksum>34ef670sdflj7340hb</checksum> |
| <status>In Production</status> |
| <config type="SOFTWARE"> |
| <name>sw\_config</name> |
| <revision>v2</revision> |
| <description>ddd</description> |
| <component primary="yes"> |
| <name>pb980sw1</name> |
| <revision>v1</revision> |
| <part\_number>s1</part\_number> |
| <oid>Z@SDF2345SF!FDFJL</oid> |
| <description>de1</description> |
| <comparison\_order>99</comparison\_order> |
| </component> |
| <component> |
| <name>ttts</name> |
| <revision>v3</revision> |
| <part\_number>yy</part\_number> |
| <status>In Production</status> |
| <oid>SFSDFSF4556#$1DFGG#@G</uri> |
| <description>des</description> |
| <regulatory\_exception> |
| <country>CN</country> |
| <country>US</country> |
| </regulatory\_exception> |
| <crc>abcd</crc> |
| </component> |
| </config> |
| <config type="HARDWARE"> |
| <name>hw\_config1</name> |
| <revision>v1</revision> |
| <description>dd</description> |
| <update\_time>01/10/2013</update\_time> |
| <status>In Production</status> |
| <component> |
| <name>pb980hw1</name> |
| <revision>v1</revision> |
| <part\_number>p1</part\_number> |
| <description>de1</description> |
| <status>In Production</status> |
| <serial\_number>35B1234567</serial\_number> |
| </component> |
| <component required="false"> |
| <name>WINTEL</name> |
| <revision>4.0</revision> |
| <part\_number>ABC</part\_number> |
| <status>In Production</status> |
| </component> |
| </config> |
| <config type="FIRMWARE"> |
| <name>fw\_config1</name> |
| <revision>v1</revision> |
| <description>xyz</description> |
| <update\_time>01/10/2013</update\_time> |
| <status>In Production</status> |
| <component> |
| <name>Amplifier</name> |
| <revision>4.0</revision> |
| <part\_number>ABC</part\_number> |
| <status>In Production</status> |
| </component> |
| </config> |
| </system\_config> |
| <system\_config type="normal"> |
| <name>sys\_config02</name> |
| <revision>v1</revision> |
| <description>this is another named sys config</description> |
| <status>In Production</status> |
| <checksum>68684656</checksum> |
| <config type="HARDWARE"> |
| <name>hw\_config1</name> |
| <revision>v1</revision> |
| <description>dd</description> |
| <component required="true"> |
| <name>pb980hw9</name> |
| <revision>v1</revision> |
| <part\_number>p1</part\_number> |
| </component> |
| </config> |
| </system\_config> |
| </devicetype\_config>  </notification>  </notifications>  </response>  <warning>this running configuration is upgrade-forbidden </warning>  </message> |

**Match Algorithm**

This is the logic used to match system configurations, defined on the server and forwarded to the Agent, to the device running configuration.

1. Compare device running configuration and defined system configurations. If all HW in the device running configuration exists in a system configuration, add it to the candidate list.
2. Scan the candidate list for configurations which have a Regulatory Exclusion which matches the device country. Remove those system configurations which meet this criteria.
3. Sort the candidate list, latest first. according to the precedence order in the defined system configurations.
4. Determine whether any warning need be added to the response.

**Configurations Which Generate a Warning**

1. Incompatible hardware – requires hardware update
2. Incompatible software – software should be updated, if possible
3. Incompatible software and hardware – software or hardware should be updated
4. Never upgrade software – usually a device which is running special software
5. Country exclusion – device country collides with software exclusion list. Change software or device country

**Warning Messages**

"Device running software [ComponentName] v[SoftwareRevision] and device country are in conflict for country exclusion." – Device country is in the country exclusion list of the installed software.

"Device country is unknown. Country exclusion cannot be checked." – Device country is not in the list of known countries

"Failed to get running configuration. Country exclusion cannot be checked." -- Can’t find a valid running config

“It's a warning configuration because of invalid hardware combination. Please make hardware changes in order to get the device into a production status.” – Self-explanatory

“It's a warning configuration because of software incompatible with hardware. Please check and upgrade software packages to get the device into a production status.” – Self-explanatory

"It's a warning configuration because of invalid software combination. Please check and upgrade software packages to get the device into a production status." – Self-explanatory

“This device shall never be upgraded due to administrative purpose” – CoT Admin has marked this configuration as not to be upgraded.

**NOTE:** One and only one warning will be returned. Client should re-issue this message until user has updated the device such that there are no message responses with warning config status.

If the request was unsuccessful, the following response is returned to the Client:

<?xml version="1.0" encoding="UTF-8"?>

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid="bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3">

<response type="bad" xaction\_guid="55c26878-90fb-4a4a-aa1f-3a689310771e" />

<reason>Error Reason</reason>

</message>

**Reasons**

“There is not any RUNNING configuration could be found in the request.” – System could not find the running configuration part of the message.

#### Software Installation Notifications

##### Software Download Start (Notification)

Mark the status of the software as ‘start’ to indicate that the attempt is eminent. This allows the backend systems to track the state of the update. If the closing “software download acknowledgement” message is not sent the system will respond by sending an email to enrolled users. The message will notify them of the problem. The “software download acknowledgement” message follows the installation message.

<?xml version='1.0' encoding='utf-8'?>

<message schema\_version='3644767c-2632-411a-9416-44f8a7dee08e' session\_guid='7242ff7a-fbec-44d8-8c50-2b1fc0780617'>

<request type='postnotification' xaction\_guid='6d21cdca-bbd3-4e38-aa9e-bb83d2b8b68f'>

<params>

<mailbox>3bd775ad-7250-42ff-adf4-188e0b691bff</mailbox>

</params>

<notification oid='378534e1-2ec5-410f-a6e3-fcf238d77efb'>

<header notification\_type='event' notification\_action='create'>

<pertinent\_type>61e08b77-df3c-4735-9f3b-0b42efb7bdcf/91dec715-f256-421a-8560-e6b015dae5f9</pertinent\_type>

<pertinent\_identifier>980 112358</pertinent\_identifier>

</header>

<body>

<components>

<component type='software'>

<part\_number>10090951</part\_number>

<revision>X00</revision>

<timestamp>8/21/2012 1:37:36 PM</timestamp>

<status>start</status>

</component>

</components>

</body>

</notification>

</request>

</message>

##### Software Download Acknowledgement (Notification)

For the software installation, the component status should be one of: installed, failed, or not attempted.

NOTE: The Agent will replace the timestamp tag value supplied by the Client with a 32-bit Unix timestamp. This eliminates the issue of not having the Client timezone. Since the Agent cannot know which Client may provide a Unix timestamp and which may not, it will always replace this tag’s contents. This means that the time reported by the DMP will likely be a few seconds later (at most) than that reported by the Client.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2">

<request type="postnotification" xaction\_guid="791a4831-a42d-46fd-b193-5a07bcc62796">

<params>

<device\_uri>{61e08b77-df3c-4735-9f3b-0b42efb7bdcf}/{V7777777BX}</device\_uri>

<country>US</country>

</params>

<notification oid="c1f1810f-9682-40f8-8faa-f754209e548b">

<header notification\_type="event" notification\_action="create">

<pertinent\_type>61e08b77-df3c-4735-9f3b-0b42efb7bdcf/91dec715-f256-421a-8560-e6b015dae5f9</pertinent\_type>

<pertinent\_identifier>V7777777BX</pertinent\_identifier>

</header>

<body>

<components>

<component type="SOFTWARE">

<name>Control</name>

<part\_number>2222222</part\_number>

<revision>01.08.en</revision>

<timestamp>6/20/2013 9:49:08 AM</timestamp>

<status>not attempted</status>

</component>

</components>

</body>

</notification>

</request>

</message>

On failure, an error response is returned with an indication of reason.

**Reasons**

“User Session not valid” – there is no session associated with the request

“Device Info not found” – the device configuration database is not found

“Invalid request message” – request message could not be parsed successfully

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid="535cd3f1-84b7-42d9-b9f6-7ed4e78580e2" >

<response xaction\_guid="791a4831-a42d-46fd-b193-5a07bcc62796" type=”bad”>

</response>

<params>

<reason>Error ?????</reason>

</params>

</message>

##### Software Download Acknowledgement Sequence

The correct sequence for applying the Software Download Acknowledge messages goes as follows:

1. Issue GetStatus until download time is 0 and Agent status is idle
2. Post DeviceInfo notification w/current docked device’s configuration
   1. Agent puts file OIDs into device Inbox
3. Post GetHeaders notification to retrieve OIDs
4. Post GetOid notification for OID to be programmed into device
   1. Retrieve SW specified in response
5. Post SoftwareDownloadAck with SW update start status
6. Client Updates the device
7. Retrieve device config from device to verify device has updated correctly
8. Post SoftwareDownloadAck with SW update complete status
9. Post DeviceInfo notification to inform server of new device configuration

NOTE: Deviating from this sequence can cause unpredictable results on the server. Such results may include recording both a SW update success AND a device configuration discrepancy. An update should not result in a discrepancy report.

#### Log Info (Notification)

Log Info is used by the Client to ask the Agent to upload one or more device log files. The Client may specify individual files, or directories containing files. If a directory is specified, then all files in the directory will be transferred as a log file to the GDMP log repository and made available in the WebUI, and log retrieval service.

NOTE: Once the Client has issued this message, the Client may not modify the specified log entities, regardless whether file or directory. This means the specified log entities may not be added/removed/modified.

**Header Setup**

The pertinent\_type tag is takes the usual device type specifier, followed by the message type specifier. In this case the message type specifier indicates either a decoded (to text form) log or a raw binary log. The pertinent\_identifier tag takes the usual device serial number.

**Body Setup**

The tags are described below.

**Agent/Server Behavior**

When the GDMP Agent receives this message, it takes one of two actions. If the log file specification is just a normal file path format (c:\some\path\logfile.txt), it behaves in the legacy manner: it copies the file from the Client workspace to the Agent workspace and then deletes the Client copy of the file. However, if the log file specification uses the URI format ([FILE|DIR]://some/path/logfile.txt), it immediately moves the specified log file(s) (copy is ***\*not\**** used) to the Agent folder space. This assumes that the log file is in the same disk volume as the Agent workspace. If it is successful, then it responds to the Client with an “ok” status. Otherwise, it responds with a “bad” status and the associated reason. See bad response example below.

The Agent then compresses and encrypts each log file. The Agent next assigns the logs to various threads for uploading to the GDMP App server. The log files will be chunked up into 10 MB pieces before they are sent. The Agent performs the uploading in the background, even if there is no active Client connection. The Agent creates checksums for each chunk, which the Server checks, and responds to the Agent as appropriate. Any failures result in retrying a chunk until it is successful.

NOTE: GDMP will not manage split archives. If a Client sends split archives, GDMP will not assemble them in the repository. The end user will need to perform this function when they retrieve the log splits. If the end user is not prepared to perform this function, then all compressed files sent to the GDMP must be self-consistent and de-compressible on their own.

<?xml version='1.0' encoding='utf-8'?>

<message schema\_version='3644767c-2632-411a-9416-44f8a7dee08e' session\_guid=''>

<request type='postnotification' xaction\_guid='1d65b46d-d14d-4bcd-9c50-083102cc043c'>

<params>

  <mailbox>3bd775ad-7250-42ff-adf4-188e0b691bff</mailbox>

</params>

<notification oid='95db4769-701b-4aa4-8f6f-f47e124459bf'>

   <header notification\_type='event' notification\_action='create'>

    <pertinent\_type>7a85f0c9-531e-4754-ad68-04c77ed63657/2f62d564-a162-440a-a5f6-ed16e7e632d5</pertinent\_type>

     <pertinent\_identifier>980 112358</pertinent\_identifier>

  </header>

<body>

     <logs>

<log name='sysdiag'>

<meta http-equiv="transfer-encoding" content="gzip"/>

<uri\_inc> C:/Logs/PB980\_VENTILATOR-980 112358-sysdiag-raw-2\_21\_53 PM.log.xml.gz </uri\_inc>

     </log>

     </logs>

   </body>

  </notification>

</request>

</message>

**Tag Description**

1. logs – required: denotes the beginning of list of log files or directories to be delivered to the GDMP server and repository
2. log – required: denotes the beginning of metadata describing the log to be uploaded. The name parameter is required. It is used to describe the log in the GDMP WebUI.
3. uri\_inc – required: defines path to the log to be uploaded. Path must be accessible by the GDMP Agent. URI is preferred to begin with file:// or dir:// as appropriate to the entity being specified. If no type specifier is included, then file:// is assumed.
4. meta – optional: used to define how the log is comprised. Most typical is to leave it out, or define the compression type, if any.

If the log file was successfully transferred, an OK response message is returned.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='Ok' xaction\_guid='1d65b46d-d14d-4bcd-9c50-083102cc043c'>

</response>

</message>

On failure, an error response is returned with an indication of reason.

**Reasons**

"Loggin Session Not available" – there is no valid session for the operation.

"Log File Not Found" – The file specified in the message was not found.

“Log move failed” – the log move operation was not successful.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3' >

<response type='bad' xaction\_guid='1d65b46d-d14d-4bcd-9c50-083102cc043c'>

<params>

<reason>Error ?????</reason>

</params>

</response>

</message>

#### Serial Number Reprogram

Serial number reprogram is for those devices which keep their chassis serial number not only on a tag on the back of the machine, but also in non-volatile memory that is accessible by the device’s internal processor. In the case of the need to swap out the board on which the serial number is stored, then a means to reprogram it must be supplied by the device and the Client must be able to take advantage of this means. When it does so, then the Client sends this message to the Server in order to inform it that this operation has been completed.

Note that this message is designed to help track odd use cases where a service technician has swapped boards between two devices (for testing purposes, for example) but did not reprogram the serial numbers back to their proper values, and they no longer match what is listed on the back of the device. So, the Client must keep track of both the previous and the current device serial numbers.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid='SESSION\_ID'>

<request type='snreprogramlog' xaction\_guid='55c26878-90fb-4a4a-aa1f-3a689310771e'>

<params>

<device\_uri>{@GUID@}/{@SN@}</device\_uri>

<oldsn>@SN@</oldsn>

<newsn>@SN@</newsn>

<reason>@REASON@</reason>

</params>

</request>

</message>

**Tag Description**

1. oldsn: required, is old serial number before reprogram
2. newsn: required, is new serial number after reprogram
3. reason: required, the reason of executing serial number reprogramming – i.e. board replaced, mismatch detected, etc.

#### Document Request Message:

Refer to section 4.6, Message/Object Type Classifications, for the GUIDs which specify other types of documents.

This message is deprecated because it is no longer necessary to use this message for DMP v2.1 and higher. A document may now be retrieved by OID, just like any other package. The document OID is returned in GetHeaders or GetMatchedConfigs **(TBD)**.

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

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e"

session\_guid='0c1d5c90-4dd2-4e76-bf6e-286e024b1822'>

<request type='postnotification' xaction\_guid='e1c38f30-50a0-4ccf-a74a-c7ec11b54544'>

<notification oid='3fe214bce7fa9bea'>

<header notification\_type="event" notification\_action="create">

<pertinent\_type>7a85f0c9-531e-4754-ad68-04c77ed63657/c250c8db-b532-4d18-8956-420c3d637a41</pertinent\_type>

<pertinent\_identifier>docGUID</pertinent\_identifier>

</header>

<body>

<components>

<component type='document'>

<file\_size>40660</file\_size>

<md5>b641ecc77d014cb8e63964f315a108868c034b09</md5>

<location>C:\Program Files\Covidien\ Device Management Agent\cache\3fe883d4c3f469c9Cipher.zip</location>

</component>

</components>

</body>

</notification>

</request>

</message>

Response:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<message session\_guid="0c1d5c90-4dd2-4e76-bf6e-286e024b1822"

schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<request xaction\_guid="e1c38f30-50a0-4ccf-a74a-c7ec11b54544"

type="postnotification">

<notification oid="3fe214bce7fa9bea">

<header notification\_type="event" notification\_action="create">

<pertinent\_identifier>docGUID</pertinent\_identifier>

<pertinent\_type>7a85f0c9-531e-4754-ad68-04c77ed63657/c250c8db-b532-4d18-8956-420c3d637a41</pertinent\_type>

</header>

<body>

<is\_Accessible>false</is\_Accessible>

<components>

<component type="document">

<location>C:\Program Files\Covidien\Device Management

Agent\cache\Gateway-986-988.pdf</location>

<file\_size>40660</file\_size>

<md5>b641ecc77d014cb8e63964f315a108868c034b09</md5>

</component>

</components>

</body>

</notification>

</request>

</message>

Failure Response:

**Reasons**

”Doucment Not Found” – couldn’t find the document in the URL specified in the message.

“File is not found” - could not open the document file.

“File is not pdf format – file is not in pdf format.

“User Session not valid” – User login not found

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<message session\_guid="0c1d5c90-4dd2-4e76-bf6e-286e024b1822"

schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">

<request xaction\_guid="e1c38f30-50a0-4ccf-a74a-c7ec11b54544" type="bad">

<params>

<reason>Doucment Not Found</reason>

</params>

</request>

</message>

### Prep Step

This message allows the Client to inform the Agent which devices should have precedence regarding package download from the server. Those in the list will be enabled for package download. Those not in the list will be disabled. Furthermore, if multiple Clients from different Classes of Trade have logged into the system, those not specified in the list will also be disabled.

When this message is received by the Agent, and Named Configuration data files are paused, these files will be downloaded first.

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

<message schema\_version=*"3644767c-2632-411a-9416-44f8a7dee08e"* session\_guid=*'SESSION\_ID'*>

<request type=*'prepsteps'* xaction\_guid=*'e1c38f30-50a0-4ccf-a74a-c7ec11b54544'*>

<params>

<device\_types>

<device\_type> DeviceTypeGUID1 </device\_type>

<device\_type> DeviceTypeGUID2 </device\_type>

<device\_type> DeviceTypeGUID3 </device\_type>

</device\_types>

</params>

</request>

</message>

**Tag Descriptions**

1. device\_types:optional – wraps a list of device\_type tags. If it is not specified, the prep steps feature is turned off at the Agent.
2. device\_type:required – required if device\_types is present. Contains a single device GUID.

If the request is successful, the following response is returned to the Client:

<?xml version="1.0" encoding="UTF-8"?>

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid="bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3">

<response type="ok" xaction\_guid="55c26878-90fb-4a4a-aa1f-3a689310771e" />

<warning>this running configuration is wrong-software </warning>

</message>

If the request was unsuccessful, the following response is returned to the Client:

**Reasons**

“User does not have permission to prep-step for device type: DeviceTypeGUID” – Agent was unable to honor the request due to lack of user permission.

“Please enable prepsteps feature on login” – Before using the PrepStep message Client must enable the feature in the Agent using the <prepstep\_enabled> tag when the user logs in.

<?xml version="1.0" encoding="UTF-8"?>

<message schema\_version="3644767c-2632-411a-9416-44f8a7dee08e" session\_guid="bd86a4a2-a894-4be5-b0e4-8b9b24ba6ed3">

<response type="bad" xaction\_guid="55c26878-90fb-4a4a-aa1f-3a689310771e" />

<reason>Error Reason</reason>

</message>

### GetSysConfigs

GetSysConfigs is an Agent🡪Server request only. It is used to retrieve the Named System Configuration details for all the devices which the users which have logged into this particular Agent (via a Client) have permissions to install SW. This occurs typically during Agent startup, but is also checked during user Login. See Section 12.7, Prep Step and Section 12.2.1, Login. This data is used by the Agent to provide matched configurations in response to GetMatchedConfigs.

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* standalone=*"yes"*?>  <message session\_guid=*"df17f4a9-9324-42a5-8fce-06b8aa8b1a28"*  schema\_version=*"3644767c-2632-411a-9416-44f8a7dee08e"*>  <request xaction\_guid=*"c67a7a12-f291-4f02-a2ea-63f5bc83b9f2"* type=*"getsysconfigs"*>  <params>  <device\_type>@GUID@</device\_type>  </params>  </request>  </message> |

Response is the list of Named System Configurations for the device.

**Tag Descriptions**

system\_config:required – contains a list of HW, SW and FW configurations. Tag attribute, type, has values normal, running, and incompatible software.

component:required – contains a list of item metadata. Tag attribute, primary, indicates that it is used in conjunction with tag, comparison\_order, to determine the precedence order of a software item – e.g. to determine the “latest software.”

comparison\_order:required – contains an increasing integer value, which is used to determine the “latest software.” Larger values indicate software which was released later in time.

For other tag definitions, see section 12.7.1.2, Sync Device Config, and section 12.7.1.3, Get Matched Configs.

|  |
| --- |
| <? xml version="1.0" encoding="UTF-8"?>  <message session\_guid="4bcfca69-a819-42d8-b84b-56b9c520dd64" schema\_version="3644767c-2632-411a-9416-44f8a7dee08e">  <response xaction\_guid="c67a7a12-f291-4f02-a2ea-63f5bc83b7e9" type="ConfigurationNotification">  <notifications>  <notification>  <header notification\_type="response" notification\_action="get" /> |
| <devicetype\_guid> abcdefg </devicetype\_guid> |
| <system\_config type="normal"> |
| <name>sys\_config01</name> |
| <revision>v1</revision> |
| <description>this is a named system config</description> |
| <update\_time>01/10/2013</update\_time> |
| <checksum>34ef670sdflj7340hb</checksum> |
| <status>In Production</status> |
| <config type="SOFTWARE"> |
| <name>sw\_config</name> |
| <revision>v2</revision> |
| <description>ddd</description> |
| <component primary="yes"> |
| <name>pb980sw1</name> |
| <revision>v1</revision> |
| <part\_number>s1</part\_number> |
| <oid>Z@SDF2345SF!FDFJL</oid> |
| <description>de1</description> |
| <comparison\_order>99</comparison\_order> |
| </component> |
| <component> |
| <name>ttts</name> |
| <revision>v3</revision> |
| <part\_number>yy</part\_number> |
| <status>In Production</status> |
| <oid>SFSDFSF4556#$1DFGG#@G</uri> |
| <description>des</description> |
| <regulatory\_exception> |
| <country>CN</country> |
| <country>US</country> |
| </regulatory\_exception> |
| <crc>abcd</crc> |
| </component> |
| </config> |
| <config type="HARDWARE"> |
| <name>hw\_config1</name> |
| <revision>v1</revision> |
| <description>dd</description> |
| <update\_time>01/10/2013</update\_time> |
| <status>In Production</status> |
| <component> |
| <name>pb980hw1</name> |
| <revision>v1</revision> |
| <part\_number>p1</part\_number> |
| <description>de1</description> |
| <status>In Production</status> |
| <serial\_number>35B1234567</serial\_number> |
| </component> |
| <component required="false"> |
| <name>WINTEL</name> |
| <revision>4.0</revision> |
| <part\_number>ABC</part\_number> |
| <status>In Production</status> |
| </component> |
| </config> |
| <config type="FIRMWARE"> |
| <name>fw\_config1</name> |
| <revision>v1</revision> |
| <description>xyz</description> |
| <update\_time>01/10/2013</update\_time> |
| <status>In Production</status> |
| <component> |
| <name>Amplifier</name> |
| <revision>4.0</revision> |
| <part\_number>ABC</part\_number> |
| <status>In Production</status> |
| </component> |
| </config> |
| </system\_config> |
| <system\_config type="normal"> |
| <name>sys\_config02</name> |
| <revision>v1</revision> |
| <description>this is another named sys config</description> |
| <status>In Production</status> |
| <checksum>68684656</checksum> |
| <config type="HARDWARE"> |
| <name>hw\_config1</name> |
| <revision>v1</revision> |
| <description>dd</description> |
| <component required="true"> |
| <name>pb980hw9</name> |
| <revision>v1</revision> |
| <part\_number>p1</part\_number> |
| </component> |
| </config> |
| </system\_config> |
| <system\_config type="incompatible\_software"> |
| <name>blacklist\_cfg\_01</name> |
| <revision>v1</revision> |
| <description>SW is incompatible with HW</description> |
| <checksum>sdfgsfgsg</checksum> |
| <config type="HARDWARE"> |
| <name>hw\_cfg\_09</name> |
| <revision>v1</revision> |
| <description>dd</description> |
| <component required="true"> |
| <name>RF Board 1</name> |
| <revision>v1</revision> |
| <part\_number>p1</part\_number> |
| </component> |
| <component required="true"> |
| <name>Steering Relay Board</name> |
| <revision>v1</revision> |
| <part\_number>p1</part\_number> |
| </component> |
| <config type="SOFTWARE"> |
| <name>sw\_config</name> |
| <revision>v2</revision> |
| <description>ddd</description> |
| <component> |
| <name>pb980sw1</name> |
| <revision>v1</revision> |
| <part\_number>s1</part\_number> |
| </component> |
| <component> |
| <name>ttts</name> |
| <revision>v3</revision> |
| <part\_number>yy</part\_number> |
| </component> |
| </config> |
| </system\_config> |
| </devicetype\_config>  </notification>  </notifications>  </response>  <warning>this running configuration is upgrade-forbidden </warning>  </message> |

### Self-Registration

The DMP provides a means for the Client to obtain the URL of the Self-Registration page of the DMP WebUI. The purpose of this is so that the Client may render this page in its display to provide a means for the user to easily request a login to the DMP. The function is offered only during connected mode, and the Client should merely display the page(s) and allow the user to interact with it as they would normally in any browser.

Request

<?xml version=*'1.0'* encoding=*'utf-8'*?>

<message schema\_version=*'3644767c-2632-411a-9416-44f8a7dee08e'* session\_guid=*'{$sessionId}'*>

<request type=*'register'* xaction\_guid=*'{$randomVar}'*></request>

</message>

Variables:

*$sessionId*: the ID of the current session the Client opened before.

*$randomVar*: It’s decided by your Client to identify the unique action your Client did, this value will be in the response to this request.

Response

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

<message schema\_version=*"3644767c-2632-411a-9416-44f8a7dee08e"* session\_guid=*'{$sessionId}'* >

<response type=*'ok'* xaction\_guid=*'{$randomVar}'*>

<params>

<url>https://rss.covidien.com/register.php</url>

</params>

</response>

</message>

**Reasons**

“This function cannot work in disconnected mode” – the client wants to get the register url but the agent is running on offline mode

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

<message schema\_version=*"3644767c-2632-411a-9416-44f8a7dee08e"* session\_guid=*'{$sessionId}'* >

<response type=*'bad'* xaction\_guid=*'{$randomVar}'*>

<reason>Error Reason</reason>

</response>

</message>

**Tag Descriptions**

*<url>: the URL to the register page.*

Variables:

*$sessionId*: It is same as the *$sessionId* in the request.

*$randomVar*: It is same as the *$randomVar* in the request

# REST Client Communication

This section details the REST services which are provided by the Agent. Just as the Agent passes along XML messages to the App Server when the Agent is online, it also passes along REST service calls to the App Server when the Agent is online. The converse is also true. When the Agent is offline, it handles the REST service calls itself.

## REST Interface Details

### REST API

The Agent REST API is a Web Services interface using HTTP POST. The HTTP POST message is sent to a REST URL and contains a JSON data object

### URL

#### Client

The Agent REST service URL invoked by the Client is comprised of two components: Agent location and REST service path.

Agent location is specified in the following manner:

<http://[hostname|domain> name|IP address]

Since the Agent is typically accessed by the Service Client locally, the Agent location usually looks like this:

<http://localhost>

REST path is specified in the following manner:

Agent/{service name}

Where {service name} might look like “Login”

Putting it all together, a typical Agent REST URL could look like

<http://localhost/Agent/Login>

#### Agent

As mentioned above, the Agent is in between the Service Client and App Server for the purposes of providing offline operational capability. However, when the Agent is directly connected to the App Server, then, in most cases, it passes the service requests to the App Server instead handling them itself. In this case, the Agent must access the REST services provided by the App Server. The Agent will use a different URL than the Client. Typically, this URL looks like the “domain name” variant mentioned above:

<https://gdmp.covidien.com/REST-SERVICE-PATH>

Note that the Agent communicates with the App Server over HTTPS while the Client communicates with the Agent over HTTP. This is due to the fact that typically the Client is on the same machine as the Agent. However, the new v4.x architecture allows the Agent and the Client to exist on separate machines.

### REST JSON Data Object

The Agent REST JSON data object contains data relevant to the service being invoked. It follows the JSON “key”:”value” text specification and typically looks something like the request text below. Note that in this case the designation BASE64(MD5(password) specifies special operations that the Service Client must perform on the user-supplied password and which results in the “value” string associated with the “password” key.

{  
 "password": BASE64(MD5(password)),  
 "user": "jack"  
}

The REST service response JSON data object looks like the following:

{  
 "result": {  
 "errMsg": "",  
 "success": "true"  
 },  
 "downloaded": "true",  
 "fileUri": "web url for cc to download the file",  
 "percentage": "100%"  
}

All responses start with the “result” key, which encapsulates a summary of the operation. The “errMsg” key is blank or null if the “success” key is set to “true.” Otherwise “errMsg” will contain detailed error information for the requested service.

After the “result” key, comes the data response which is specific to the requested service.

NOTE: The explanations of the various services below will only call out keys which have not previously been defined, or which change meaning in the context of a particular service.

## Service Client 🡨🡪 Agent Service Definitions

NOTE: When the Agent is connected to the App Server, the Agent forwards most of these service calls to the App Server for handling.

### GetServerStatus

GetServerStatus service informs the Service Client as to whether the Agent is connected to the App Server. It lets the Client know whether to expect its messages will generally go directly to the App Server, or whether the Agent is providing the App Server functionality while offline.

#### Call

URL: http://{agentloc}/Agent/GetServerStatus

JSON:

{

"auth": {

"token": "token123"

}

}

**auth**: JSON key denoting a structure for authentication

**token:** a key which value maps onto an authenticated session on the App server or the Agent

#### Response

JSON:

{  
 "result": {  
 "errMsg": "",  
 "success": "true"  
 },  
 "connected": "true"  
}

**errMsg**: Contains error explanation text if “success” is “false.” Otherwise, it is set to the empty string, “”.

**success:** State of the operation/request. Set to true or false.

**connected**: State of the Agent connection to the App server.   
True if connected, otherwise false.

### StatDevice

StatDevice service is used to determine if the server already knows about a particular device. Typically, if the server does not know about the device, then the Client would issue a CreateDevice request.

#### Call

URL: http://{agentloc}/Agent/StatDevice

JSON:

{  
 "auth": {  
 "token": "token123"  
 },  
 "country": "US",  
 "deviceID": {  
 "deviceType": "61BF648E-3181-41e4-9EF2-222F8DF8B538",  
 "serialNumber": "SN123"  
 }  
 }

**country:** key which maps onto a two-letter ISO definition of a country

**deviceID:** key which defines the beginning of a structure of device identifiers

**deviceType:** a key which value is a GUID string which uniquely identifies the device type.

#### Response

JSON:

{  
 "result": {  
 "errMsg": "",  
 "success": "true"  
 },  
 "exists": "true"  
}

**exists:** True if the App server has the device in its DB, false otherwise.

### CreateDevice

CreateDevice service is used to instruct the server to add the device details to the devices data in the server DB. After this call, the Client should sync the device running configuration.

#### Call

URL: http://{agentloc}/Agent/CreateDevice

JSON:

{  
 "auth": {  
 "token": "token123"  
 },  
 "country": "US",  
 "deviceID": {  
 "deviceType": "61BF648E-3181-41e4-9EF2-222F8DF8B538",  
 "serialNumber": "SN123"

},

“facilityID”:”US-540614”  
}

**serialNumber:** key which value corresponds to the device unique identifier which is most usually stamped on the device itself, but in the case of software devices, is created using multiple pieces of host computer information.

**facilityID:** key which value is the MDT designator for the facility. In some cases, this is from the Salesforce.com system, and in some cases, is created on the GDMP server. It can also be “Unknown.”

#### Response

JSON:

{  
 "result": {  
 "errMsg": "",  
 "success": "true"  
 },  
 "created": "true"  
}

**created:** True if successfully created in App server DB, false otherwise.

### UpdateAcknowledge

The UpdateAcknowledge service is used to inform the server of the steps of installing software or a feature entitlement package on the device. Part number and revision parameters are not required for feature entitlement.

#### Call

URL: http://{agentloc}/Agent/UpdateAcknowledge

JSON:

{  
 "auth": {  
 "token": "token123"  
 },  
 "country": "US",  
 "deviceID": {  
 "deviceType": "61BF648E-3181-41e4-9EF2-222F8DF8B538",  
 "serialNumber": "SN123"  
 },

"updateAck": {  
 "ackType": "ScheduleUpdateSoftware",  
 "name": "software name",  
 "partNumber": "part number 123",  
 "revision": "rv 12",  
 "scheduleJobGUID": "test job guid",  
 "status": "start",  
 "timestamp": "6/20/2013 9:49:08 AM"  
 }  
}

**ackType**: key which value is the state of the operation which is being reported. Valid values follow below:  
UpdateSoftware  
UpdateSoftwareComplete  
UpdateLicense  
UpdateLicenseComplete  
ScheduleRetrieveLog  
ScheduleUpdateSoftware  
ScheduleUpdateLatestSoftware

**name**: Key which value is the textual name of the software/firmware which is being reported on.

**partNumber**: Key which value is the textual part number of the software/firmware which is being reported on.

**revision**: Key which value is the textual revision of the software/firmware which is being reported on.

**scheduleJobGUID**: Key which value is the textual GUID which specifically identifies the job on which the software/firmware activity will take place.

**status**: Key which value indicates progress of operation being reported

**timestamp**: Key which value is the textual date and time of the operation which status is being reported

#### Response

JSON:

{  
 "result": {  
 "errMsg": "",  
 "success": "true"  
 }

}

### GetCountryList

GetCountryList service is used to retrieve a list of all known device countries.

#### Call

URL: http://{agentloc}/Agent/GetCountryList

JSON: {  
 "auth": {  
 "token": "token123"  
 }

}

#### Response

JSON: {  
 "result": {  
 "errMsg": "",  
 "success": "true"  
 },  
 "countries": [{  
 "code": "CN",  
 "name": "China"  
 }, {  
 "code": "FR",  
 "name": "France"  
 }]  
 }

**countries:** Key which denotes the beginning of a list of structures defining countries

**code:** Key which value is the two character ISO country code

**name:** Key which value is the textual name of the country in English

### GetFacilityList

GetFacilityList service is used to retrieve a list of all facilities which are associated with a user.

#### Call

URL: http://{agentloc}/Agent/GetFacilityList

JSON: {

"auth": {

"token": "token123"

}

}

#### Response

JSON: {

"result": {

"errMsg": "",

"success": "true"

},

"facilities": [{

"address": "facilityAddress1",

"id": " US-540614",

"name": "facilityName1"

}, {

"address": "facilityAddress2",

"id": " US-540615",

"name": "facilityName2"

}]  
}

**facilities:** Key which denotes the beginning of a list of structures defining facilities.

**address:** Key which value is the textual address containing at a minimum, street number, street, city, state, and postal code

**id:** Key which value is the MDT designator for the facility. In some cases, this is from the Salesforce.com system, and in some cases, is created on the GDMP server. It can also be “Unknown.”  
  
NOTE: Same as facilityId in CreateDevice service

**name:** Key which value is the textual name of the facility

### GetDeviceList

GetDeviceList service retrieves all devices for a particular facility.

#### Call

URL: http://{agentloc}/Agent/GetDeviceList

JSON: {  
 "auth": {  
 "token": "token123"  
 },

“facilityId”:”US-540614”

}

#### Response

JSON: {

"result": {

"errMsg": "",

"success": "true"

},

"devices": [{

"coT": "class of trade1",

"deviceId": {

"deviceType": "device type1",

"serialNumber": "serial number1"

}

}, {

"coT": "class of trade2",

"deviceId": {

"deviceType": "device type2",

"serialNumber": "serial number2"

}

}]  
}

### GetDeviceTypes

GetDeviceTypes service retrieves all device types supported by GDMP. If the facilities list is not empty, get only those device types which are at the facilities in the list.

#### Call

URL: http://{agentloc}/Agent/GetDeviceTypes

JSON: {

"auth": {

"token": "token123"

},

"facilities": [{

"address": "facilityAddress1",

"name": "facilityName1",

"id": "id1"

}, {

"id": "id2",

"name": "facilityName2",

"address": "facilityAddress2"

}]  
}

NOTE: “facilities” is currently unused

#### Response

JSON: {

"deviceTypes": [{

"cleanLog": "true",

"coT": "class of trade1",

"type": "device type name1"

}, {

"cleanLog": "false",

"coT": "class of trade2",

"type": "device type name2"

}],

"result": {

"errMsg": "",

"success": "true"

}  
}

**cleanLog:** Key which value determines whether a particular device type’s logs should be cleared after they have been retrieved.

**type:** Key which value is the textual name of the device type.

### GetSoftwares

GetSoftwares service returns the metadata for all the software packages which are available for a particular device type.

#### Call

URL: http://{agentloc}/Agent/GetSoftwares

JSON: {  
 "auth": {  
 "token": "token123"  
 },  
 "deviceType": "device type1"  
 }

#### Response

JSON: {

"result": {

"errMsg": "",

"success": "true"

},

"softwares": [{

"CRC": "crc1",

"comparisonOrder": "1",

"fileSize": "168",

"language": "CN",

"name": "software1",

"partNumber": "partNumber1",

"revision": "rv1",

"status": "InProduction"

}, {

"CRC": "crc2",

"comparisonOrder": "2",

"fileSize": "169",

"language": "US",

"name": "software2",

"partNumber": "partNumber2",

"revision": "rv2",

"status": "LimitedRelease"

}]

}

**comparisonOrder:** Key which numeric value denotes which of several packages which may be similar in name, part number or revision, is actually the “latest” software. Largest number is the “latest software.”

**fileSize:** Key which value denotes the size of the device SW package in bytes.

**language:** Key which value denotes the language of the SW package.

**status:** Key which value denotes the production status of the package. It may have the values Production, and LimitedRelease.

### DownloadSoftware

URL: http://{agentloc}/Agent/DownloadSoftware

DownloadSoftware service requests the Agent to download software to the Agent cache by language, part number, revision and name. Repeated calls to this service will report the progress of ongoing downloads.

If a download is ongoing, it will report the following parameters:

Downloaded: false

Percentage: 0-99

If a download is complete, the service will report the following parameters:

Downloaded: false

Percentage: 0-99

FileUri: web url to download the file locally.

#### Call

JSON {

"auth": {

"token": "token123"

},

"language": "English",

"partNumber": "partNumber1",

"revision": "rv1",

"softwareName": "software1"

}

#### Response

JSON {

"result": {

"errMsg": "",

"success": "true"

},

"downloaded": "true",

"fileUri": "web url for cc to download the file",

"percentage": "100%"

}

**downloaded:** Key which values specifies whether the Agent has the file fully downloaded on disk. Downloaded should only be true when percentage has gone to 100%.

**fileUri:** Key which specifies the PC local URI of the file which has been requested.

**percentage:** Key which denotes how much of the file had been downloaded from the App server.

### DownloadSavedLogfile

DownloadSavedLogfile service requests the metadata for a log file for a specific device which was saved in the Agent cache. If the log file exists, then the file URI is returned in the response. The Client may then retrieve the file by its URI.

#### Call

URL: http://{agentloc}/Agent/DownloadSavedLogfile

JSON: {

"auth": {

"token": "token123"

},

"deviceId": {

"deviceType": "device type",

"serialNumber": "SN01"

}  
}

#### Response

JSON: {

"result": {

"errMsg": "",

"success": "true"

},

{

"logFileSaved": "true",

"logFiles": [{

"date": "20160418",

"fileUri": "url for cc to download the file",

"logFileType": "sub type1"

}, {

"date": "20160418",

"fileUri": "url for cc to download the file",

"logFileType": "sub type2"

}]

}  
}

**logFileSaved:** Key which value denotes whether there are any saved log files

**logFiles:** Key which value denotes the beginning of a list of log files saved on the Agent.

**date:** Key which value denotes the date the log file was saved

**logFileType:** Key which value denotes a device type specific log file type (event, error, etc)

### DownloadFeatureLicense

DownloadFeatureLicense service downloads device feature license files by facility, device type and serial number to the Agent cache. If device type and serial number are not provided, then the service returns all the feature licenses for the devices on which the user is trained.

The service does not return until all feature license files are downloaded. On completion, the Agent returns the local URIs to these files to the Client. If the Agent is offline, then it immediately returns the URIs of the license files already in the Agent cache.

NOTE: Facility is not used at this time.

#### Call

URL: http://{agentloc}/Agent/DownloadFeatureLicense

JSON: {

"auth": {

"token": "token123"

},

"deviceId": {

"deviceType": "device type",

"serialNumber": "SN01"

},

"facilities": [{

"address": "facilityAddress1",

"id": "id1",

"name": "facilityName1"

}, {

"address": "facilityAddress2",

"id": "id2",

"name": "facilityName2"

}]  
}

#### Response

JSON: {

"result": {

"errMsg": "",

"success": "true"

},

"fileURIs": [{

"fileURI": "C:\\Users\\gpan\\Git\\gdmp4-server\\gdmp-agent\\.\\cache\\Ablation\\Emprint Ablation Visualization Application\\3fad61c1efc56020Cipher",

"deviceTypeName": "Emprint Ablation Visualization Application",

"serialNumber": "0000-0002-7433-4237-8077-8021-88",

"fileName": null,

"fileId": 87,

"timestamp": 1489002129000,

"applied": true

},{

"fileURI": "C:\\Users\\gpan\\Git\\gdmp4-server\\gdmp-agent\\.\\cache\\Ablation\\Emprint Ablation Visualization Application\\3fad61c1efc56020Cipher",

"deviceTypeName": "Emprint Ablation Visualization Application",

"serialNumber": "0000-0002-7433-4237-8077-8021-88",

"fileName": null,

"fileId": 87,

"timestamp": 1489002129000,

"applied": true

}]  
}

**fileName:** Key which value is the text name of the file

**fileId:** Key which value is the unique numeric/GUID ID of the file

**timestamp:** Key which value is the numeric Unix timestamp of the file

**applied:** Key which value is boolean and indicates whether the license has previously been applied to the device.

### UploadFileInit

UploadFileInit service is used to inform the GDMP that a file with one or more chunks will be sent to the GDMP server. The file metadata is sent to the GDMP server in this request. No file chunks are sent in this request. The GDMP server returns with a task ID. File chunks are then sent using this task ID.

#### Call

URL: http://{agentloc}/Agent/UploadFileInit

JSON: {

"auth": {

"token": "token123"

},

"chunkCount": "100",

"chunkSize": "1024",

"date": "20160418",

"deviceID": {

"deviceType": "VLEX Client",

"serialNumber": "SN123"

},

"fileType": "LogFile",

"originalFileName": "xxx.log",

"scheduleJobGUID": "test job guid",

"subFileType": "sub log file type"  
 }

**chunkCount:** Key which value is the number of data blocks which will be transferred to the Server to complete the upload of a file.

**chunkSize:** Key which value denotes the number of bytes in a single upload chunk.

**fileType:** Key which value denotes the general file type, such as Log File, Report, etc

**subFileType:** Key which value denotes the specific type of file (error log, event log, use report, etc).

**scheduleJobGUID:** Key which denotes the schedule ID which is uploading the device file. This key may not be present as the log could be uploaded manually.

#### Response

JSON: {

"result": {

"errMsg": "",

"success": "true"

},

"taskID": "taskID1"  
}

**taskID:** Key which value denotes the task on which a particular file will be uploaded. It is used to reference the proper task when uploading chunks.

### UploadFileChunk

UploadFileChunk service is used to upload the file specified in UploadFileInit call, one chunk at a time, using the task ID returned by the UploadFileInit call.

#### Call

URL: http://{agentloc}/Agent/UploadFileChunk

JSON: {

"auth": {

"token": "token123"

},

"base64EncodeData": "AQIDBAUGBwgJCg==",

"CRC": "cJA+ebdXXj9Of/oVwmCKxw==",

"chunkID": "1",

"taskID": "taskID1"  
}

**base64EncodeData:** Key which value is a base 64 string of the file chunk data to upload.

**CRC:** Key which value is the MD5sum of the base 64 encoded data

**chunkID:** Key which value is the monotonically increasing value which identifies the chunk order.

#### Response

JSON: {

"result": {

"errMsg": "",

"success": "true"

}

}

### DownloadFileInit

DownloadFileInit service is used to start the retrieval of any file which other REST calls have returned the file URI and metadata. The Agent or Server will decrypt the file and send the chunks in clear text. However, the file should not exist in clear text at rest. This service returns the number of chunks that will be required given the file size (already known by the Agent) and the chunk size passed in. This means that the Agent will only return chunks of the size requested by the Client. The Client must invoke DownloadFileChunk REST API chunkCount times to retrieve the entire file.

#### Call

URL: http://{agentloc}/Agent/DownloadFileInit

JSON: {

"auth": {

"token": "token123"

},

"chunkSize": "1024",

"fileURI": "//D://test.zip"  
}

#### Response

JSON: {

"chunkCount": "123",

"result": {

"errMsg": "",

"success": "true"

}  
}

**chunkCount:** Key which value denotes the number of chunkSize chunks required to retrieve the requested file.

### DownloadFileChunk

The DownloadFileChunk service requests a chunk of the file specified at the specified chunk index. This allows the Client to re-request a specific chunk if it failed on a previous request.

#### Call

URL: http://{agentloc}/Agent/DownloadFileChunk

JSON: {

"auth": {

"token": "token123"

},

"chunkID": "1",

"chunkSize": "1024",

"fileURI": "//D://test.zip"  
 }

#### Response

JSON: {

“result": {

"errMsg": "",

"success": "true"

}  
 }

### UploadRunningCfg

UploadRunningCfg service is used to send the device running configuration to the Agent. The Agent will return a software list that can be applied to this device or a warning message if anything is wrong with the running cfg. This list may be used by the Client to populate a selection list from which the user can choose a software package.

#### Call

URL: http://{agentloc}/Agent/UploadRunningCfg

JSON: {

"auth": {

"token": "token123"

},

"components": [{

"CRC": "12345",

"name": "software name",

"partNumber": "part number 123",

"revision": "rv 12",

"type": "Software"

}, {

"name": "hardware name",

"partNumber": "part number 123",

"revision": "rv 12",

"serialNumber": "serial number",

"type": "Hardware"

}],

"country": "US",

"deviceID": {

"deviceType": "VLEX Client",

"serialNumber": "SN123"

},

"facilityID": "Facility1",

"features": [{

"id": "feature id 1",

"licenseTerm": {

"limit": "42",

"remaining": "1",

"type": "LimitedCount",

"used": "41"

},

"status": "feature status 1"

}, {

"id": "feature id 2",

"licenseTerm": {

"endDateTime": "2016-01-01 00:00:00",

"limit": "0",

"remaining": "0",

"startDateTime": "2015-12-01 00:00:00",

"type": "LimitedDuration",

"used": "0"

},

"status": "feature status 2"

}],

"region": "NorthAmerica"

}

**type:** Components: component type: hardware, software  
License term: term type: LimitedCount, LimitedDuration, Permanent

**status:** Key which value means whether the feature is in use or not on the device.

#### Response

JSON: {

"result": {

"errMsg": "",

"success": "true"

},

"softwares": [{

"CRC": "CRC111",

"comparisonOrder": "1",

"language": "CN",

"name": "software1",

"partNumber": "part number",

"revision": "rv1",

"status": "InProduction"

}, {

"CRC": "CRC222",

"comparisonOrder": "1",

"language": "CN",

"name": "software2",

"partNumber": "part number2",

"revision": "rv2",

"status": "InProduction"

}],

"warningMsg": "all good"

}

**warningMsg:** Key value which specifies whether the supplied running config is acceptable. If it is not acceptable, it may be due to HW-HW incompatibility, HW-SW incompatibility, HW-FW incompatibility, or SW-FW incompatibility.

### GetSoftwaresForDevice

GetSoftwaresForDevice service will return software that can be applied to the specific device based on running configuration of that device. This means that the device running configuration must have been sent to the Agent using UploadRunningConfig service prior to invoking this service. If latestOnly parameter is true, only return latest software

DMM should filter the software list by user access rights.

#### Call

URL: http://{agentloc}/Agent/GetSoftwaresForDevice

JSON: {

"auth": {

"token": "token123"

},

"deviceID": {

"deviceType": "111",

"serialNumber": "sn111"

},

"latestOnly": "false"  
}

**latestOnly:** Deprecated: Key which boolean value specifies whether to return only the software with the highest comparison order, or not.   
The latestOnly parameter is ignored. The Agent is solely responsible for checking this permission.

*Deprecated:*

#### Response

JSON: {  
 "result": {  
 "errMsg": "",  
 "success": "true"  
 },  
 "softwares": [{  
 "CRC": "crc1",  
 "comparisonOrder": "1",  
 "fileSize": "168",  
 "language": "CN",  
 "name": "software1",  
 "partNumber": "partNumber1",  
 "revision": "rv1",  
 "status": "InProduction"  
 }, {  
 "CRC": "crc2",  
 "comparisonOrder": "2",  
 "fileSize": "169",  
 "language": "US",  
 "name": "software2",  
 "partNumber": "partNumber2",  
 "revision": "rv2",  
 "status": "LimitedRelease"  
 }]  
}

### CCVersion

CCVersion service is used to report Common Client version to GDMP and check for updates. Latest version of Client software is returned.

*Deprecated: This service will be renamed to ClientVersion, as it is entirely possible for there to be other v4.0 native Clients other than Common Client.*

#### Call

URL: http://{agentloc}/Agent/CCVersion

JSON: {  
 "auth": {  
 "token": "token123"  
 },  
 "ccVersion": "2015.111.2"  
}

#### Response

JSON: {  
 "result": {  
 "errMsg": "",  
 "success": "true"  
 },  
 "latestCCVersion": "2015.111.1"  
 }

### Login

Login service is used to log user into GDMP Agent/Server. It supports two modes:

* Normal login with user name and textual password.
  + This is to support first time login of the user.
  + Before the user logs in the first time, the Agent has no record of the user’s credentials.
* Third Party login with user name and password hash.
  + This is to support user login from Vital Sync or another third party application.
  + In this case, the password will be MD5ed then encoded by Base64 before sent to Agent

The service returns the user’s authorizations and permissions.

#### Call

URL: http://{agentloc}/Agent/Login

JSON:

{  
 "password": BASE64(MD5(password)),  
 "user": "jack"  
}

#### Response

JSON:

{  
 "ExternalSystemUserPermissions": {  
 "permission": [{  
 "deviceType": "Valleylab LS10",  
 "testingSoftware": "false",  
 "userLicenseUpdate": "false",  
 "userLogCfgUpload": "false",  
 "userSoftwareAccess": "1"  
 }, {  
 "deviceType": "VLEX Client",  
 "testingSoftware": "true",  
 "userLicenseUpdate": "false",  
 "userLogCfgUpload": "true",  
 "userSoftwareAccess": "2"  
 }]  
 },  
 "result": {  
 "errMsg": "",  
 "success": "true"  
 },  
 "ExternalSystemSession": {  
 "SessionID": "100"  
 }  
}

### Logoff

Logoff service is used to notify the Agent/Server that the user has terminated their session.

#### Call

URL: http://{agentloc}/Agent/Logoff

JSON:

{  
 "auth": {  
 "token": "token123"  
 }

}

#### Response

JSON:

{  
 "result": {  
 "errMsg": "",  
 "success": "true"  
 }  
}

## Agent 🡨🡪 App Server Specific Service Definitions

### DownloadFile

#### Call

URL: /gdmp-server/agent/api/download

JSON: {

“id”:[0-9]+,

“type”:[“Software”|”Business Rules”|”Document”]

}

#### Response

HTTP response which contains either the binary data stream which is the file itself, or an HTTP error code.

## HwSw Config

#### Call

URL: /gdmp-server/agent/api/config/hwsw

#### Response

JSON: {

“version": -1,

"device\_type\_guids": ["B0DC2BE4-D744-45c6-AEF6-EBEF319A336B",

"DACD7FA1-9D67-4057-9952-C55F8EA6227B",

"D:\\COVIDIEN\\DMP\\DELIVERABLES\\GDMP\\SRC\\GDMP4-SERVER\\GDMP-AGENT\\RUNTIME",

"61e08b77-df3c-4735-9f3b-0b42efb7bdcf",

"61BF648E-3181-41e4-9EF2-222F8DF8B538",

"ADMINISTRATIVE\_AGENT",

"4DCC88DA-5D92-4219-AF13-162ABBCB853C",

"7a85f0c9-531e-4754-ad68-04c77ed63657",

"71C3D34A-BDB2-4B3E-9940-208E1383E47F",

"5768C9CB-8ACE-4421-B1C6-071D131B935F",

"a7a65225-ef2d-48e9-89c8-6975dd7dc054",

"AAE142EF-6047-43F8-B44E-3D0C723066E1",

"CEFC1E07-CFF6-4F27-AB05-4577A33A1BA8",

"BD5CE934-26AE-484A-8395-EB0FD29F6838",

"C1FF0EB8-1CD7-4448-BD06-406B2F93E725",

"74C44BFD-0677-4b3a-AE67-7070FCD5A720",

"fc1f3c2c-16df-4b52-a9ea-99409b131d31",

"APPLICATION\_CLIENT\_INFO",

"3695F650-E602-47e6-A3DF-5525BD41CCA3",

"CED7070B-0F36-43E0-BFD4-8282E3393A76",

"3B682913-6D1E-4355-9E48-208EB7061A3D",

"HW\_SW\_CONFIG"]

}

## ClientApp

### Call

URL: /gdmp-server/agent/api/clientapp

JSON:

{

"client\_type\_guid": "ADMINISTRATIVE\_AGENT",

"serial\_number": null,

"software\_list": [{

"name": null,

"part\_number": null,

"revision": ""

}],

"business\_rule\_list": [{

"name": "Prep step mapping",

"part\_number": null,

"revision": "1"

}]

}

Response

JSON: {

"timestamp": 1491333657657,

"ref": "13860556626992",

"server\_version": "4.0.0.110.66a650c",

"new\_version": true,

"new\_software\_list": [{

"name": "SCDUInstaller",

"type": "Software Bundle",

"md5": "a76b314fc99ff6994df6a2521e73107593c953dd",

"status": "In Production",

"fileId": 489,

"expiration": 3700,

"hash": "0985b08a8c567d215c1fc49e95e4bcd6",

"language": "English",

"part\_number": "0",

"revision": "1.0.3",

"file\_size": 846494,

"file\_name": "SCDUInstaller\_v1-0-3.exe",

"comparison\_order": 3,

"regulatory\_exclusion": []

}],

"new\_business\_rule\_list": [{

"name": "SCDUBusinessRules",

"type": "Business Rules",

"md5": "b5da06da25fb2fd93065631faa7f3c78c1ca9a09",

"status": "In Production",

"fileId": 487,

"expiration": 3700,

"hash": "1d3fb5495df841a0f497a6e2414ebf4d",

"language": "English",

"part\_number": "0",

"revision": "6.0",

"file\_size": 13454,

"file\_name": "SCD700\_Updater.config",

"comparison\_order": 6,

"regulatory\_exclusion": []

}, {

"name": "SCDUBusinessRules",

"type": "Business Rules",

"md5": "35be3df5ca53e835f8da06206758699d539d3654",

"status": "Archived",

"fileId": 423,

"expiration": 3700,

"hash": "b2ca31eb16f50cfa68565f737f7e76db",

"language": "US English",

"part\_number": "0",

"revision": "4.0",

"file\_size": 13332,

"file\_name": "SCD700\_Updater.config",

"comparison\_order": 3,

"regulatory\_exclusion": []

}, {

"name": "SCDUBusinessRules",

"type": "Business Rules",

"md5": "7b9bf26371d5426ec19181346396bb9ad432d5ec",

"status": "Archived",

"fileId": 387,

"expiration": 3700,

"hash": "fb7025fed14a39a32dd8e93fc23a1fd0",

"language": "US English",

"part\_number": "0",

"revision": "3.0",

"file\_size": 13143,

"file\_name": "SCD700\_Updater.config",

"comparison\_order": 2,

"regulatory\_exclusion": []

}, {

"name": "SCDUBusinessRules",

"type": "Business Rules",

"md5": "93a83197198cb2103ecd6e07b89e96f7306fc73f",

"status": "Archived",

"fileId": 23,

"expiration": 3700,

"hash": "4cc79cddb300ccaec443124e4a6f2031",

"language": "US English",

"part\_number": "0",

"revision": "2.0",

"file\_size": 12356,

"file\_name": "SCD700\_Updater.config",

"comparison\_order": 1,

"regulatory\_exclusion": []

}]

}