# Device Web APIs for 3D Printer

Approved Version 1.0 – 21 Oct 2018

**Open Mobile Alliance**

OMA-ER-Device_WebAPIs_3DP-V1_0-20181021-A

# Contents



# Figures

# Tables

# 1. Scope

This Enabler Release (ER) document is a combined document that includes requirements, architecture and technical specification of the Device WebAPIs Enabler.

The scope of OMA Device WebAPI enabler will include:

- Requirements, architecture and specifications for web-based APIs to expose services available from external devices and internal enablers through Extension Plug-Ins to applications.
- The web-based APIs that will work in the framework that GotAPI (Generic Open Terminal API Framework) defines, where the web-based APIs are implemented in the Extension Plug-Ins and exposing the services from the external devices or internal enablers that are connected with the Extension Plug-Ins.
- The framework provided by the combination of GotAPI and Device WebAPI to enable applications to work through standardized APIs with external devices or internal enablers, as GotAPI itself does not standardize the APIs to be implemented in the Extension Plug-Ins.
- Web-based APIs that will initially address such areas as healthcare devices, DWAPI-PCH (Personal Connected Healthcare), DWAPI-3DP(3-Dimension Printer) and other areas where standardization will help solving application interoperability problems.

OMA will continue expanding the coverage of the standaridized Device WebAPIs in areas where standardization helps the markets to expand and innovate.

# 2. References

## 2.1 Normative References


**[EventSource]** "Server-Sent Events", Worldwide Web Consortium (W3C), URL:http://dev.w3.org/html5/eventsource/ (latest working draft)

**[GotAPI 1.1]** (DRAFT)
Generic Open Terminal API Framework (GotAPI), Candidate Version 1.0 – 10 Feb 2015, URL:http://member.openmobilealliance.org/ftp/Public_documents/CD/CD-GotAPI/Permanent_documents/OMA-ER-GotAPI-V1_1_0-20150803-A.zip

**[HTTP/1.1]** "Hypertext Transfer Protocol -- HTTP/1.1", Internet Engineering Task Force (IETF), URL:http://tools.ietf.org/search/rfc2616

**[HTTP/2.0]** "Hypertext Transfer Protocol version 2.0", Internet Engineering Task Force (IETF), URL:http://tools.ietf.org/search/draft-ietf-httpbis-http2-09 (latest working draft)

**[JSON-RPC]** "JSON-RPC 2.0 Specification", JSON-RPC Working Group, URL:http://www.jsonrpc.org/specification

**[RFC2119]** "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997, URL:http://www.ietf.org/rfc/rfc2119.txt

**[SCRRULES]** "SCR Rules and Procedures", Open Mobile Alliance™, OMA-ORG-SCR_Rules_and_Procedures, URL:http://www.openmobilealliance.org/

**[WebSocket]** "The WebSocket API, Worldwide Web Consortium (W3C), URL:http://dev.w3.org/html5/websockets/ (latest working draft)


## 2.2 Informative References


**[CSEA]** "Client Side Enabler API (CSEA)", Version 1.0, Open Mobile Alliance™, OMA-RRP-CSEA-V1_0, URL:http://www.openmobilealliance.org/

**[OMADICT]** "Dictionary for OMA Specifications", Version 2.9, Open Mobile Alliance™, OMA-ORG-Dictionary-V2.9, URL:http://www.openmobilealliance.org/

**[OMNA]** "OMA Naming Authority". Open Mobile Alliance™. URL:http://www.openmobilealliance.org/tech/omna.aspx

**[WRAPI]** "Web Runtime API (WRAPI", Version 1.0, Open Mobile Alliance™, OMA-ERP-WRAPI-V1_0, URL:http://www.openmobilealliance.org/

# 3. Terminology and Conventions

## 3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

## 3.2 Definitions

| | |
|---|---|
| **Agent** | A node that collects and transmits 3D printer data to an associated manager. |
| **API Patterns** | Design guidelines and requirements for definition of APIs |
| **Browser Context** | Web applications executing under a Web browser as Web runtime environment. |
| **Datagram** | An API providing access to UDP protocol based networking. |
| **Device** | A physical device implementing either an agent or manager role. |
| **ECMAScript** | Use definition from [OMADICT]. |
| **Hybrid Native/Web App** | An application designed to execute under the native OS / middleware environment of a device, and that use native APIs for the execution of web content in addition to native code. |
| **JavaScript** | Use definition from [OMADICT]. |
| **Manager** | A node receiving data from one or more agent systems. Examples of managers include a cellular phone, 3D printing appliance, set top box, or computer system. |
| **Native App** | An application designed to execute under the native OS / middleware environment of a device. |
| **Personal Health Device** | A device used in personal health applications. |
| **Socket** | An API providing access to TCP protocol based networking. |
| **Uniform Resource Identifier** | Use definition from [OMADICT]. |
| **User Agent** | Use definition from [OMADICT]. |
| **Web** | The World Wide Web, a content and application framework based upon hypertext and related technologies, e.g. XML, JavaScript/ECMAScript, CSS, etc. |
| **Web Application** | An application designed using Web technologies (e.g. HTML, CSS, and Javascript). |
| **Web IDL** | An IDL language for Web application APIs |
| **Web Runtime Application** | A client-side Web application that is executed in Web runtime environments. |
| **Web Runtime Environment** | Client software that supports the execution of Web applications (e.g. browsers or widget engines). |
| **WebSocket** | An API providing networking services per the WebSocket standard [WebSocket]. |
| **Widget Context** | Web applications installed and executing under a W3C Widget [W3C-Widgets] engine as Web runtime environment. |
| **Widget Engine** | Software which supports the execution of Web applications running outside a browser context, e.g. with the same functional capabilities as browsers but without the user interface functions provided by a browser, including window frames, menus, toolbars and scroll bars. |

## 3.3 Abbreviations

| | |
|---|---|
| **API** | Application Programming Interface |
| **EventSource** | The EventSource API |

| | |
|---|---|
| **HTTP** | HyperText Transfer Protocol |
| **IDL** | Interface Definition Language |
| **JSON** | JavaScript Object Notation |
| **MDER** | Medical Data Encoding Rules |
| **MIME** | Multipurpose Internet Mail Extensions |
| **OMA** | Open Mobile Alliance |
| **REST** | REpresentational State Transfer |
| **RPC** | Remote Procedure Call |
| **SCR** | Static Conformance Requirements |
| **TS** | Technical Specification |
| **UA** | User Agent |
| **UE** | User Equipment |
| **URI** | Uniform Resource Identifier |
| **URL** | Uniform Resource Locator |
| **W3C** | World Wide Web Consortium |
| **WRAPI** | The OMA Web Runtime API enabler |
| **XML** | eXtensible Markup Language |
| **XSD** | XML Schema Definition |

# 4. Introduction

External devices that are connected with smartphones are increasingly gaining mainstream acceptance and we are starting to see rapid adoption of such devices.

While there are various types of new devices and sensors to be connected with smartphones coming out, there are fundamental issue to be solved for certain markets:

- Since there are no open standardized APIs and frameworks that application developers can use for the same type of devices, developers are required to customize their applications for each and every different device.
- In order to access features from applications, some environments mandate that the users' data must be routed through certain entities, e.g., servers outside user's control. As such, it is difficult to ensure data confidentiality and privacy to such a level where certain vertical markets require.

As the first step to solve this problem, OMA has standardized GotAPI (Generic Open Terminal API Framework) [***]. GotAPI provides the framework to enable applications (native, hybrid and web applications) to work with external devices and internal enablers through GotAPI Servers and Extension Plug-Ins based on web technologies. There are multiple Extension Plug-Ins to be expected and each Extension Plug-In is connected to external devices and internal enablers. Each Extension Plug-Ins implements web-based APIs to expose services (or data) from those connected. The applications securely access the web-based APIs under the framework that GotAPI provides. The figure-1 shows the overview of GotAPI's framework.

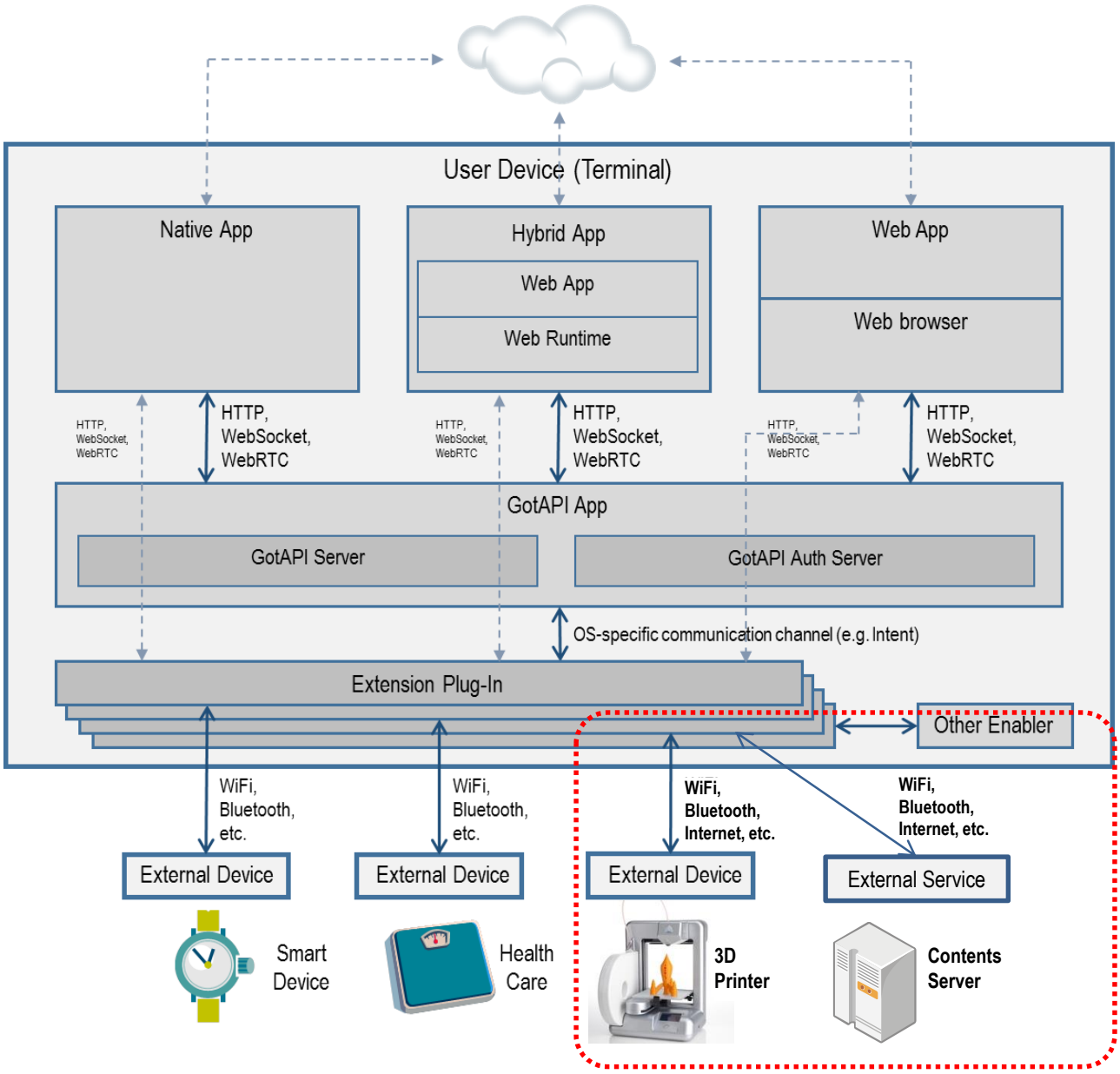


**Figure 1: Overview of GotAPI's Framework**

GotAPI itself does not standardize the web-based APIs that are implemented in the Extension Plug-Ins, and it is left open for implementers of each Extension Plug-In. This openness enables many external device vendors to freely provide unique and differentiated new services through the GotAPI framework.

On the other hand, for certain markets, standardizing the web-based APIs is desired. Standardized web-based API will enable open markets for new applications by 3rd party developers that will rapidly innovat and grow the market, while ensuring the interoperability and security.

This specification specifies OMA Device WebAPIs Enabler. In contrast to OMA GotAPI being a versatile web application framework, OMA Device WebAPIs Enabler specifies web-based APIs for certain types of external devices, external services or internal services to work consistently in the GotAPI framework. It enables applications to access specific types of external devices, external services or internal services. OMA Device WebAPIs Enabler will offer a series of specifications to address different types of devices or services where standardization is needed.

**DWAPI-3DP:**

OMA has identified 3D Printer market is looking for standardized APIs as well, and OMA Device WebAPIs Enabler addresses this issue in order to ensure service interoperability between the same type of devices, as an alternative to silo and non-interoperable devices.

For 3D printer devices, there many technologies are used to perform 3D printing itself. OMA Device WebAPIs Enabler will develop web-based APIs to enable 3D printing services from local to remote, where the APIs will be implemented in Extension Plug-Ins under the GotAPI framework. These API specifications are called as DWAPI-3DP, which stands for Device WebAPI for 3D Printer. It is one of OMA Device WebAPIs Enabler specifications and specifies web-based APIs for supporting various types of 3D Printer devices and services in a series of specifications.

## 4.1 3D Printer Service Flow Overview

The 3D printer service flow is depicted as follow:

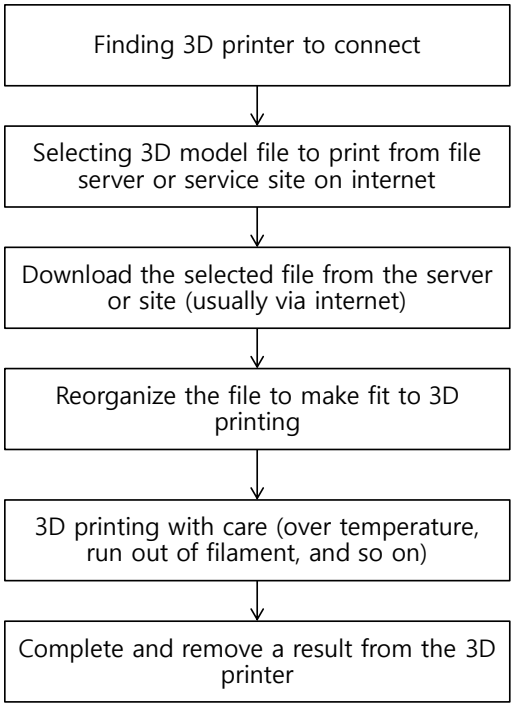


**Figure 2: 3DP Service Flow Overview**

To use 3D printer, the user may select 3D model file to print from file server or service site on internet. It may need functions receiving content server information from 3D printer.

After the selecting, the user downloads the selected file from the server or the internet site. At this moment, it takes time with large data traffic consumption.

Before the 3D printing, the downloaded file should be reorganized to enable 3D printing. It may take long time to complete.

The 3D printer starts printing, it will take “long-long” time to complete. It may depend on the complexity of the 3D model, the printing time takes couples of hours or even days.

After the completion of 3D printing, the user removes the result from the 3D printer.

## 4.2 Version 1.0

Device WebAPIs version 1.0 includes the functionality:

- Requirements and API specifications for DWAPI-3DP, with selected device based on market requirements, based on the GotAPI 1.1 framework
- Supporting assets

# 5. Device WebAPIs Enabler release description (Informative)

This release focuses on the functions of exposing the data from external devices to applications. The Device WebAPIs enabler will utilize the GotAPIs enabler and specify the APIs under the GotAPIs framework.

# 6. Requirements (Normative)

*Edtor's Note: Most of the security requirements should have already been defined in GotAPI 1.1.*

*If there is anything that is missing or unaddressed, we need to articulate them in thie security section.*

## 6.1 High-Level Functional Requirements: GotAPI Adherence

In contrast to OMA GotAPI being a versatile web application framework, OMA Device WebAPIs Enabler specifies web-based APIs for certain types of external devices or internal enablers, to work consistently in the GotAPI framework. Therefore, the OMA Device WebAPIs Enabler must adhere to all the GotAPI 1.1 specifications, including, but not limited to, those of Extension Plug-Ins, data formats, sequence flows, security measures and considerations. Where necessary to ensure interoperability, OMA Device WebAPIs Enabler may standardize specific values or data for certain types of devices consistently within the GotAPI framework.

| Label | Description | Release |
|---|---|---|
| HD-GOT-01 | Device WebAPIs Enabler SHALL adhere to all the GotAPI 1.1 specifications [GotAPI 1.1], including, but not limited to, those of Extension Plug-Ins, data formats, sequence flows, security measures and considerations. | 1.0 |
| HD- GOT -02 | Device WebAPIs Enabler MAY define specific values or data for certain types of devices that will work consistently within the GotAPI 1.1 framework with other types of devices and applications. | 1.0 |
| HD- GOT -03 | The Plug-In SHALL be compliant to the GotAPI Extension Plug-Ins as specified in the GotAPI 1.1 specification. | 1.0 |
| HD- GOT -04 | The framework hosting Plug-Ins SHALL support the security requirements defined in GotAPI 1.1. | 1.0 |

## 6.2 High-Level Functional Requirements: DWAPI-3DP

| Label | Description | Release |
|---|---|---|
| HD-HLF-03 | The Plug-In SHALL have a real time clock that is synchronized to UTC and SHALL be aware of its local time zone. | 1.0 |
| HD-HLF-04 | The Plug-In SHALL have a real time clock with a resolution that matches the resolution of any device that it interacts with. | 1.0 |
| HD-HLF-05 | The Plug-In SHALL be able to obtain the current time from the device if the device reports a current time. (Devices that report a time stamp with their measurements are required to be able to report the device's sense of current time to interoperate with the Plug-In.) | 1.0 |
| HD-HLF-06 | The Plug-In SHALL be able to map any measurement time stamp reported by the device to an ISO-8601 time stamp with offset from UTC to local time. (An ISO-8601 time stamp is yyyy-mm-ddThh:mm:ss+hhmm). "ISO-8601" data type that includes the time zone offset, expressed either as ±hhmm if the civil time zone offset is known or -0000 if UTC time is known but the actual civil time zone offset is not.<br><br>If the device does not report a time stamp with its measurement, the Plug-In SHALL use the time of reception of the measurement as the measurement time stamp. The Plug-In SHALL provide a Boolean indication of 'true' if the measurement was provided by the Plug-In because the device did not provide a measurement time stamp. | 1.0 |

| Label | Description | Release |
|---|---|---|
| HD-HLF-07 | The Plug-In SHALL correct any *measurement* time stamp by the difference between the current time reported by Device and the current time reported by the Plug-In *unless* the Plug-In knows that the device has a superior synchronization to UTC than the Plug-In does. In other words, if the device does not have superior time synchronization and the current time reported by the device is 20 seconds behind that reported by the Plug-In, the Plug-In adds 20 seconds to any of the measurement time stamps reported by the device. If the device has superior time synchronization, the Plug-In reports the device measurement time stamp unmodified.<br><br>Note: the Plug_in has the responsibility to correct the time. Some devices do not know the time or the time need to be set (i.e. do not have superior synchronization) manually (which may be different from the actual time), so that why the Plug-in need to correct the time | 1.0 |
| HD-HLF-09 | The Plug-In SHALL be able to provide the product name of the connected device. If the Plug-In cannot get the product name, it SHALL create a name for the device using an arbitrary algorithm. The algorithm is up to the Plug-In implementation, and this specification does not define any algorithms. Note that the 'algorithm' could be a non-empty user-entry. | 1.0 |
| HD-HLF-10 | The Plug-In SHALL be able to provide the manufacturer name of the connected device if the Plug-In can get the name. It SHALL be reported as a string (may be empty). | 1.0 |
| HD-HLF-11 | The Plug-In SHALL be able to provide the model number of the connected device if the Plug-In can get the model number. It SHALL be reported as a string (may be empty) | 1.0 |
| HD-HLF-12 | The Plug-In SHALL be able to provide the firmware revision of the connected device if the Plug-In can get the firmware revision. It SHALL be reported as a string (may be empty). | 1.0 |
| HD-HLF-13 | The Plug-In SHALL be able to provide the serial number of the connected device if the Plug-In can get the serial number. It SHALL be reported as a string (may be empty). | 1.0 |
| HD-HLF-14 | The Plug-In SHOULD be able to provide the software revision of the connected device if the Plug-In can get the the software revision. If reported it SHALL be reported as a string. | 1.0 |
| HD-HLF-15 | The Plug-In SHOULD be able to provide the hardware revision of the connected device if the Plug-In can get the hardware revision. If reported it SHALL be reported as a string. | 1.0 |
| HD-HLF-16 | The Plug-In SHOULD be able to provide the part number of the connected device if the Plug-In can get the part number. If reported it SHALL be reported as a string. | 1.0 |
| HD-HLF-17 | The Plug-In SHOULD be able to provide the protocol revision of the connected device if the Plug-In can get the protocol revision. If reported it SHALL be reported as a string. | 1.0 |

**Table 1: High-Level Functional Requirements**

## 6.3 3D Printer Specific Functional Requirements

The following requirements outline the 3D printer specific set of options that 3D Printer Plug-Ins implement. The 3D Printer Plug-In technical specifications will address the necessary functions for support of these options.

| Label | Description | Release |
|---|---|---|
| 3D-HSF-01 | The Plug-In SHALL provide the value that reports the information of the 3D printer including device information, printer attributes and status of the printer | 1.0 |
| 3D-HSF-01.1 | The device information SHOULD include product name, manufacturer name, model number, firmware revision, serial number, software revision, hardware revision, part number, protocol revision and system ID. | 1.0 |
| 3D-HSF-01.2 | The printer attributes SHOULD include printer type, print size of X-axis, print size of Y-axis, print size of Z-axis, network and memory size of the 3D printer | 1.0 |
| 3D-HSF-01.3 | The status information SHOUD include operating status and nozzle temperature. | 1.0 |
| 3D-HSF-02 | The Plug-In SHALL support to order the 3D printer to start printing and to end printing. | 1.0 |
| 3D-HSF-03 | The Plug-In SHALL provide functions to connect 3D content servers. | 1.0 |

| Label | Description | Release |
|---|---|---|
| 3D-HSF-04 | The Plug-in SHALL provide security mechanism for the interworking between the 3D printer and the 3D content server. | 1.0 |
| 3D-HSF-04.1 | The Plug-in SHALL get a secure information from a 3D content server and provide the information to the 3D printer to make secure connection between the server and the printer. | 1.0 |
| 3D-HSF-05 | The Plug-in SHALL provide a value that indicates URL of the 3D content server. | 1.0 |
| 3D-HSF-06 | The Plug-in SHALL get a list of printing queue information from the 3D printer and provide the list of information. | 1.0 |
| 3D-HSF-06.1 | The printing queue information SHOULD include an order, an URL of the queued contents and a message which indicates the status of the printing file. | 1.0 |

**Table 2: 3D Printer Specific Functional Requirements**

#### 6.3.1.1 Data Integrity: DWAPI-3DP

#### 6.3.1.2 Confidentiality

| Label | Description | Release |
|---|---|---|
| | | |

# 7. Architectural Model

This section describes the architectural model and related aspects of the Device WebAPI Enabler.

## 7.1 Architectural Diagram

### 7.1.1 GotAPI Framework and 3DP Plug-in External Interfaces

This section summarizes how the GotAPI framework works, in which the DWAPI is functioning. This section adheres to the specifications that are defined by GotAPI 1.1 [GotAPI 1.1].

As defined by GotAPI 1.1 [GotAPI 1.1], when an application is initiated by a user, the application obtains authorization for access to GotAPI-based APIs using the GotAPI-2 Interface. Once the application is authorized by the GotAPI Auth Server, the application can access the GotAPI Server using the GotAPI-1 Interface.

After the authorization, the application asks the GotAPI Server, using the GotAPI-1 Interface, what kind of services are available. Then the GotAPI Server requests the current status of the available services to all the installed Extension Plug-Ins using the GotAPI-4 Interface. This procedure is called the "Service Discovery", which is defined in the GotAPI specification. After the Service Discovery, the application can interact with the specified device (i.e., the service) via the Plug-In. Note that, in the GotAPI specification, external devices and internal enablers are collectively called as “services”.

When an application sends an API request on the GotAPI-1 Interface, the GotAPI Server passes it to the Plug-In using the GotAPI-4 Interface.

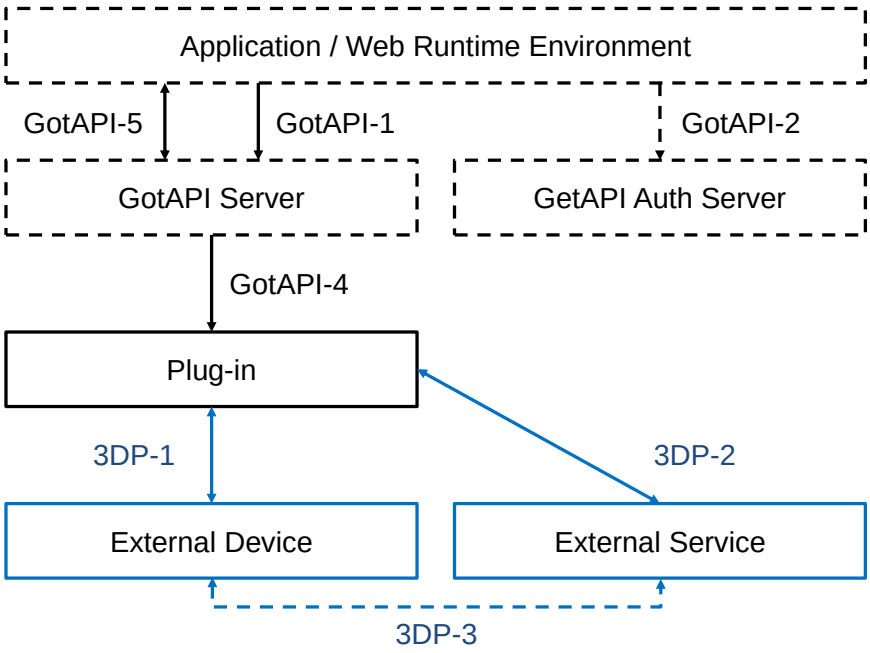


**Figure 3: Architectural Diagram**

In addition to HTTP, an application may connect to the GotAPI Server using WebSocket, which is the GotAPI-5 Interface. The GotAPI-5 Interface enables that whenever the targeted device reports event messages, the application receives the messages on the GotAPI-5 Interface asynchronously.

The GotAPI Server is agnostic to what Plug-Ins do inside. The GotAPI Server just passes a request from an application to a Plug-In and passes a response from the Plug-In to the application.

The Plug-in provides external interfaces to communicate external device and service via 3DP-1 and 3DP-2 for each, and it may provides 3DP-3 external interface between external device and external service to make them cooperation remotely.

## 7.1.2 DWAPI-3DP Basic Data Flow

This section describes how the GotAPI framework and 3D Printer devices work together using Extension Plug-Ins. The following diagram shows the basic flow of DWAPI-3DP.

Under the GotAPI framework, the Plug-In implements web-based APIs, DWAPI-3DP, and the Manager whose function is defined as 3DP-1 and 3DP-2. The 3DP-1, 3DP-2 and 3DP-3 are not in scope of this specification.

The Plug-In with the Manager communicates with 3D Printer devices that implements Agent and Printing Module through some media such as Bluetooth or WIFI. 3DP-1 defines all the necessary protocols, data formats, and the roles for Manager and Agent. From the Manager, some data is made available to DWAPI-3DP to be exposed in the web-based APIs to applications through the GotAPI framework. The Plug-In makes such data available to applications through DWAPI-3DP consistently under the GotAPI framework. The Manager also communicates with Contents Server that provides services which are essential to use 3D Printer devices.

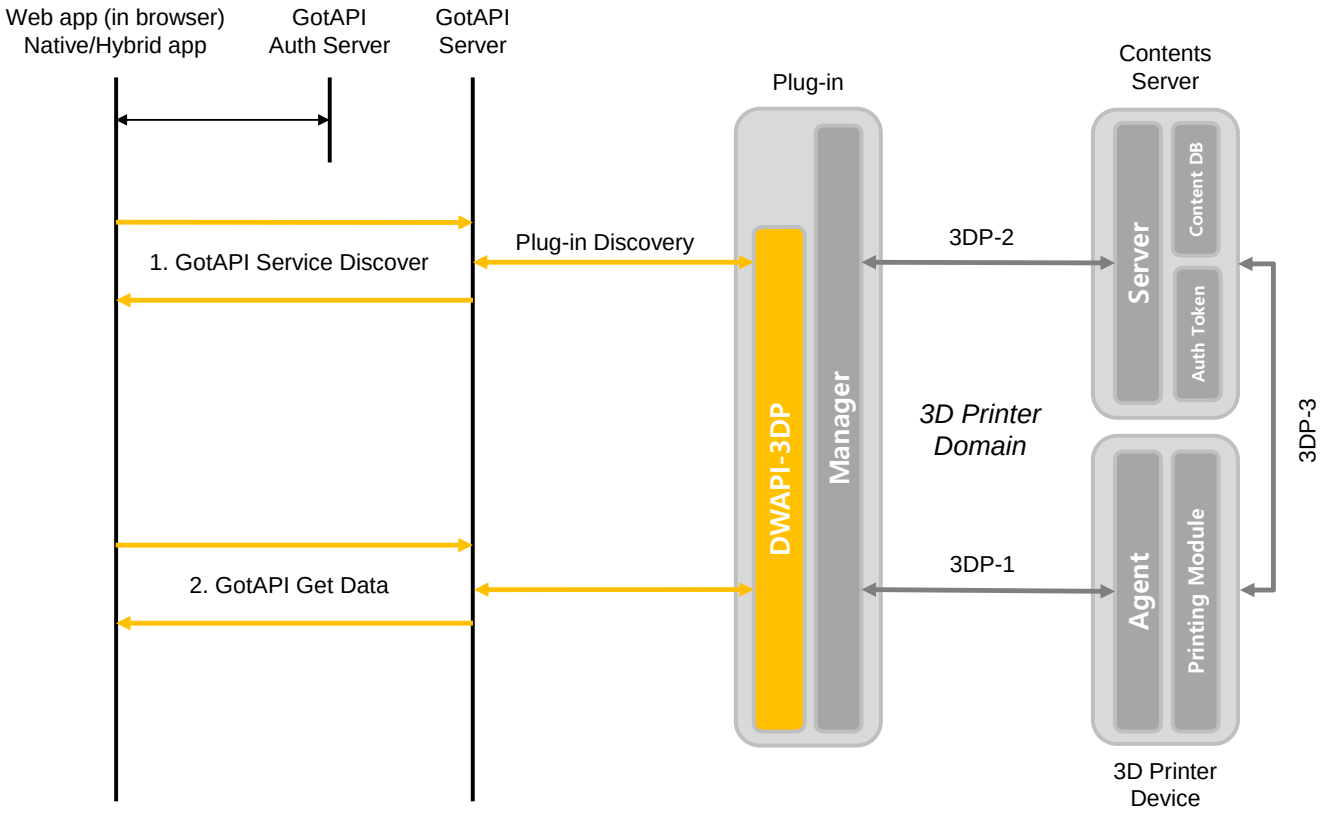


**Figure 4: DWAPI-3DP Basic data flows**

## 7.1.3 Secured Domain between Plug-in, 3DP Device and Contents Server

The 3D printing flow has many points to take times. The 3D file downloading from the contents server, slicing process of 3D model file, and 3D printing itself also takes much time.

Due to this reason, the user could not be near the 3D printer, it needs remote access to each other. For example, the user selects the 3D file to print from the contents server, orders 3D printer to get the file from the server, and the printer prints the file after the pre-3D printing process such as slicing of 3D file. To make this happen, the user, the contents server and the printer should build secure paths to each other.

The figure 5 describes how to build the secure paths between the user, the contents server and the 3D printer.

At the first time, the user requests secure token to contents server. The contents server issues the secure token and sends it to the user. The user, actually the application, forwards the token to the 3D printer when the user makes printing command to the printer. The 3D printer accesses the contents server using the secure token. Through these steps, the secure domain has been built between the user (application), the contents server and the 3D printer.

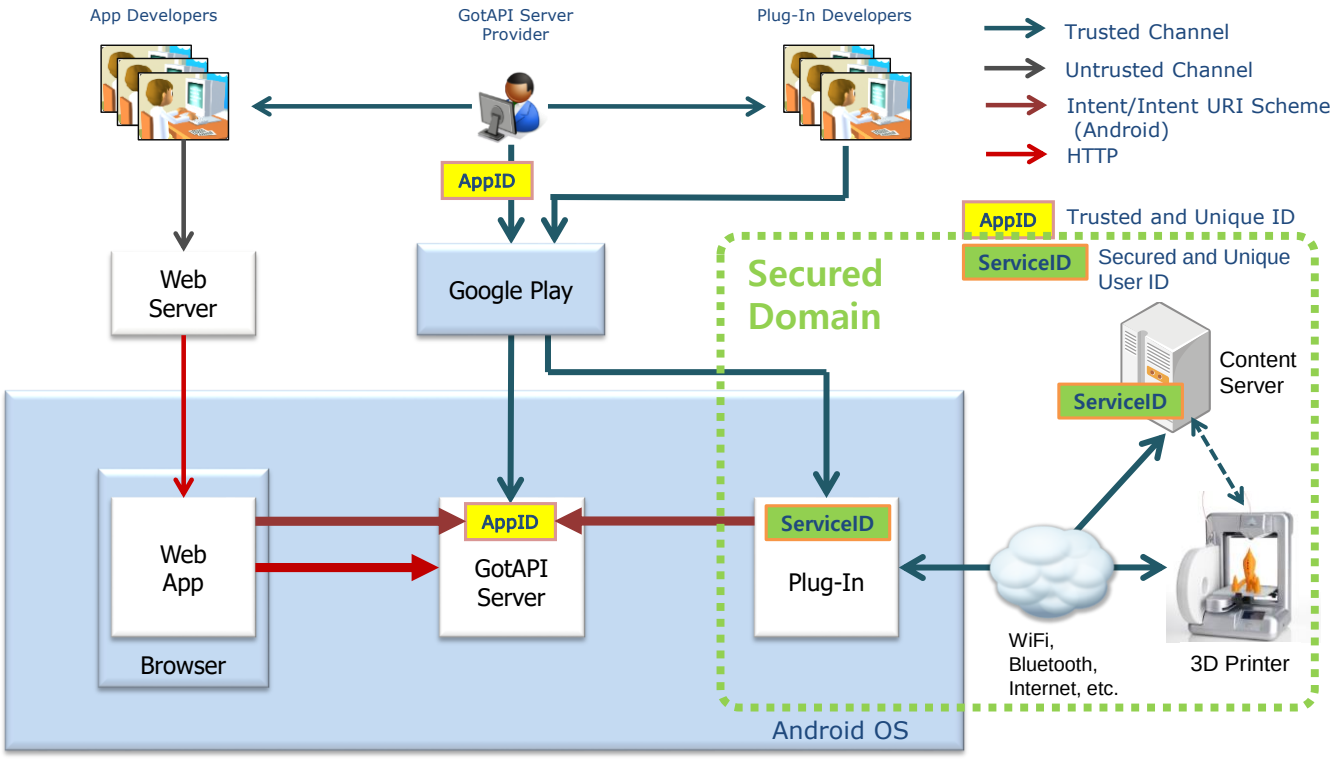


**Figure 5: Build the secure paths between the user, the contents server and the 3D printer**

## 7.2 Functional Components and Interfaces/reference points definition

DWAPI-3DP consists of the following six APIs;

1) The service discovery API enables applications to obtain information of Plug-Ins and 3D Printer devices available.
2) The one-shot measuring API enables applications to get one set of measuring values in response to a request.
3) The asynchronous messaging API enables applications to listen to asynchronous messages from the targeted device via the relevant Plug-In.
4) The service connecting API creates a secured channel between Plug-in and Contents Server, and passes the ServiceID to make the 2nd secured channel between 3D Printer and Contents Server.
5) The authentication API between Plug-in and 3D Printer makes a secured channel, and prepares forming the 2nd secured channel between 3D Printer and Contents Server.
6) The 3D Printing Command API enables applications to control and manage 3D Printer via Plug-in.

### 7.2.1 Service Discovery API

Service Discovery API specification adheres to that of GotAPI 1.1.

As defined by GotAPI 1.1, after the application obtains authorization for access to GotAPI-based APIs using the GotAPI-2 Interface, the application sends the Service Discovery request to the GotAPI Server. Then the GotAPI Server sends the Service Discovery request to all of the installed Extension Plug-Ins. The message flow of the Service Discovery is shown below.

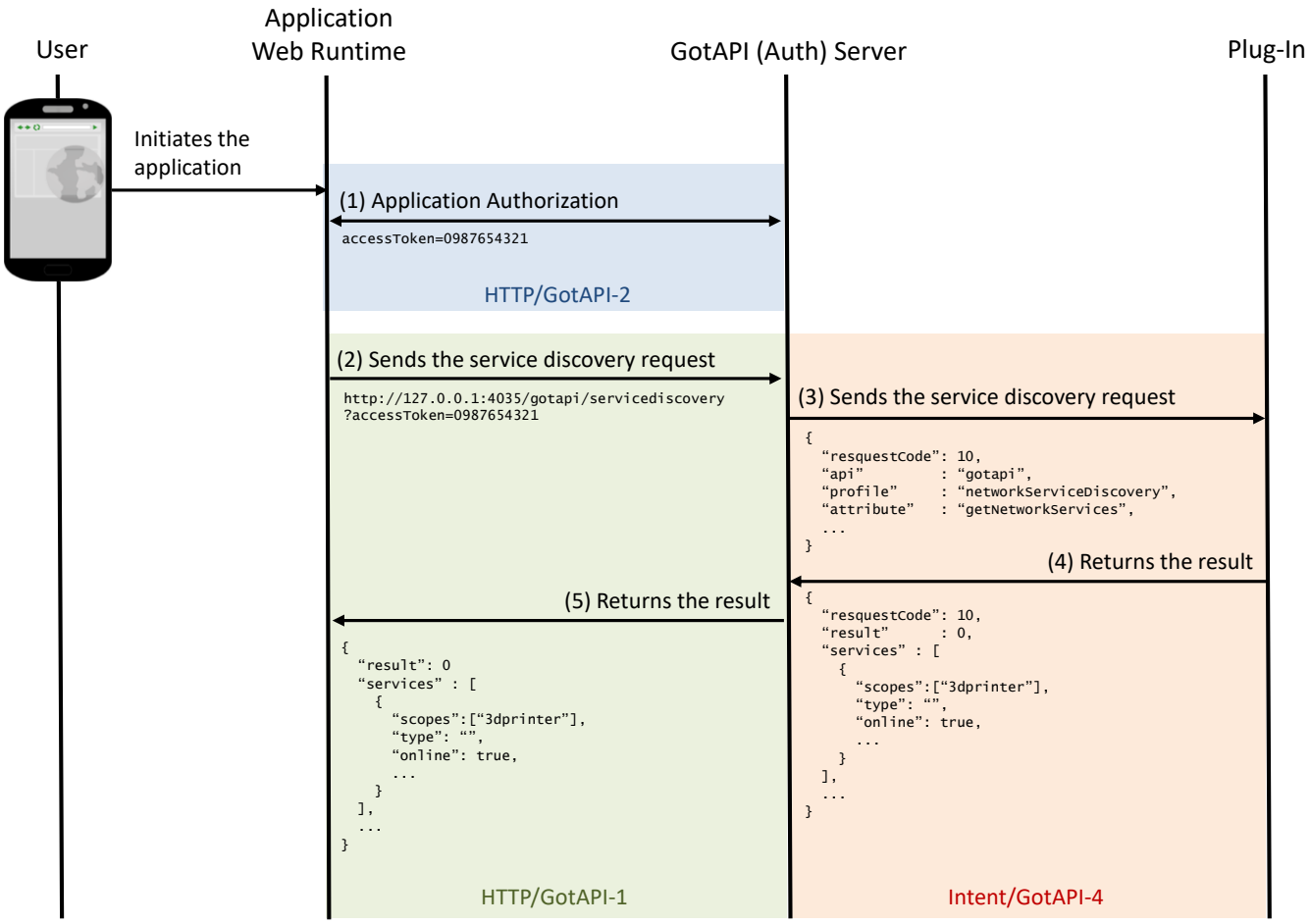


**Figure 6: Message flow of the Service Discovery**

The specific data in the message flows labelled (4) in the figure above are defined by the Plug-In that implements manager functions of 3D printer and is communicating with 3D printing devices.

The other message flows SHALL be consistent to what are defined in the GotAPI 1.1 specification.

## 7.2.2 One-shot measuring API

As defined by GotAPI 1.1, after the application obtains authorization to access GotAPI-based APIs using the GotAPI-2 Interface and completes the Service Discovery, the application can use the service (so called "One-shot measuring API") provided by the Plug-In through the GotAPI Server.

The One-shot measuring API offers a measurement result reported by the targeted device in response to a request. The message flow of this API is as shown below.

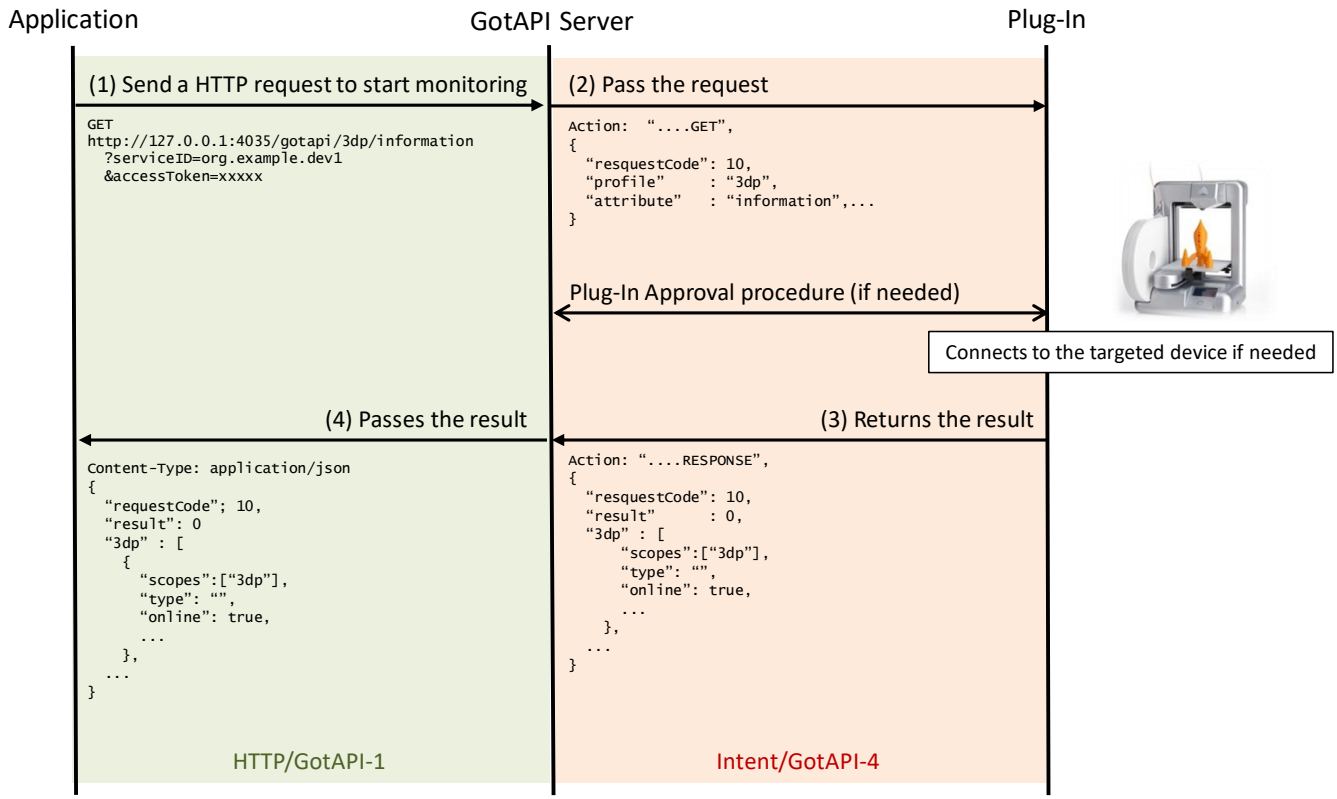


**Figure 7: Message flow of the One-shot measuring API**

1. The user triggers a request of the API in the application.
2. Label (1): The application sends a request to the GotAPI Server using HTTP (REST) over the GotAPI-1 Interface. Note that the HTTP method of the request is "GET".
3. Label (2): The GotAPI Server passes the request to the targeted Plug-In on the GotAPI-4 Interface with the Action name "GET".
4. The GotAPI Server runs the Plug-In Approval procedure if needed, which is defined in the GotAPI 1.1 specification.
5. When the Plug-In receives the request, it connects to the targeted external device if needed.
6. The Plug-In obtains current measurement values from the targeted device.
7. Label (3): The Plug-In sends a response with one set of the measurement values using the GotAPI-4 Interface.
8. Label *(4)*: When the GotAPI Server receives the response from the Plug-In, the GotAPI Server passes the response to the application on the GotAPI-1 Interface as an HTTP response.

The overall message flows to obtain data by sending HTTP request and response over the GotAPI-1 Interface SHALL adhere to the specifications defined in GotAPI 1.1.

The specific data in the message flows labelled (3) and (4) in the figure above are defined by the Plug-In that implements manager functions of 3D printer and is communicating with 3D printing devices.

### 7.2.3 Asynchronous messaging API

As defined by GotAPI 1.1, after the application obtains authorization to access GotAPI-based APIs using the GotAPI-2 Interface and completes the Service Discovery, the application can use the service (so called "Asynchronous messaging API") provided by the Plug-In through the GotAPI Server.

The Asynchronous messaging API offers a series of measurement values reported by the targeted device to an application in real time as the measurement values become available. The timing when and the reasons why such measurement values become available is determined by the Plug-Ins and connected devices, and is out of the scope of this specification.

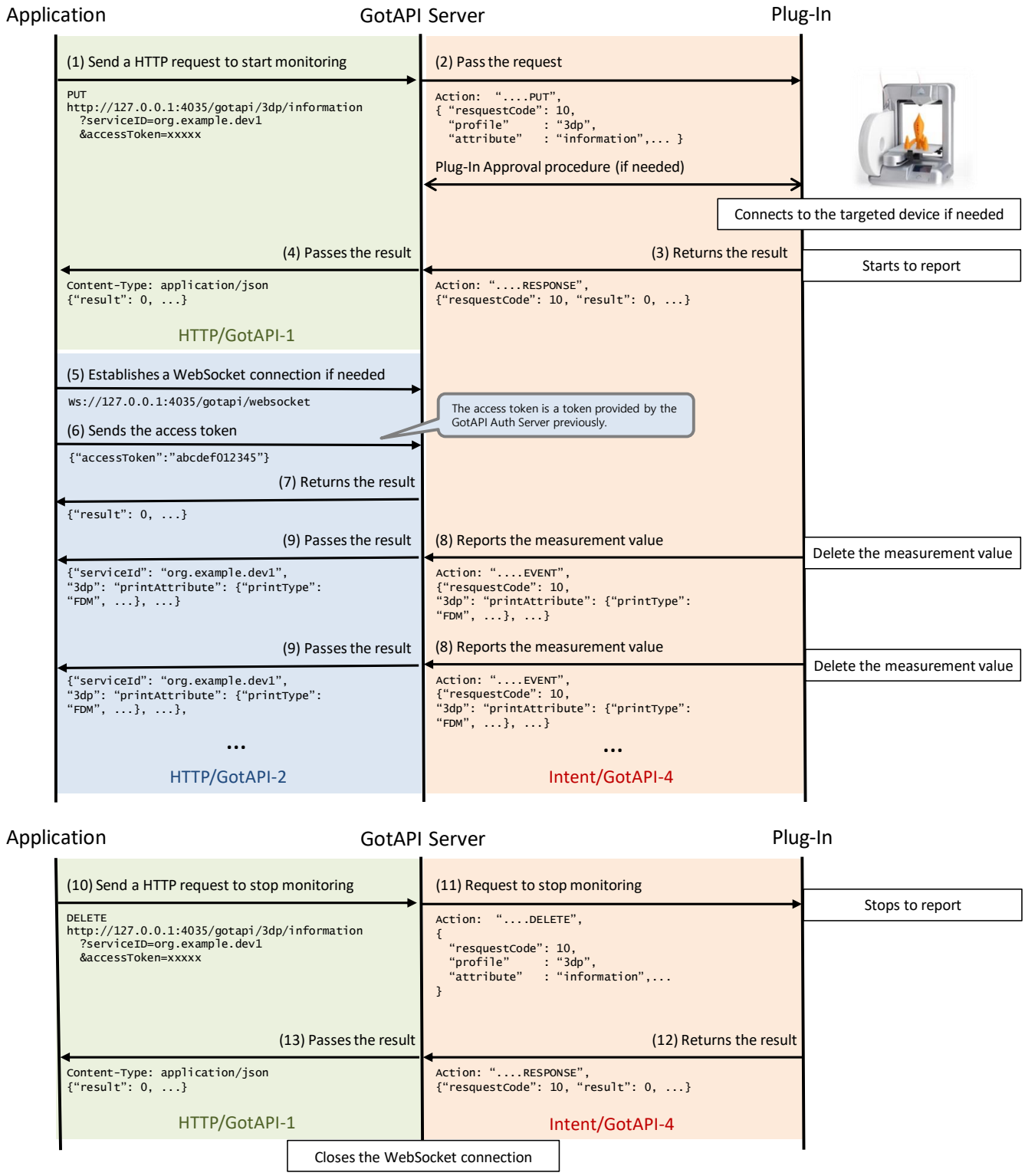


**Figure 8: Message Flow of the Asynchronous messaging API**

1. The user triggers a request of the API in the application.
2. Label (1): The application sends a request to the GotAPI Server using HTTP (REST) over the GotAPI-1 Interface. Note that the HTTP method of the request is "PUT".

3. Label (2): The GotAPI Server passes the request to the targeted Plug-In on the GotAPI-4 Interface with the Action name "PUT".
4. The GotAPI Server runs the Plug-In Approval procedure if needed, which is defined in the GotAPI 1.1 specification.
5. When the Plug-In receives the request, it connects to the targeted external device if needed.
6. Label (3): The Plug-In sends a response with the message using the GotAPI-4 Interface.
7. Label (4): When the GotAPI Server receives the response from the Plug-In, the GotAPI Server passes the response to the application on the HTTP connection as an HTTP response.
8. Label (5): The application establishes a WebSocket connection to the GotAPI Server if the application does not have a WebSocket connection to the GotAPI Server yet.
9. Label (6): As the WebSocket connection has been established, the application sends the access token to the GotAPI Server through the WebSocket connection. The access token is a token which the application obtained from the GotAPI Auth Server when the application was authorized by the GotAPI Auth Server.
10. Label (7): When the GotAPI Server receives the access token from the WebSocket channel, the GotAPI Server returns the result on whether the request is accepted or not.
11. Label (8): Whenever the targeted external device reports a message, e.g., a data or a measurement value, the Plug-In sends the message to the GotAPI Server on the GotAPI-4 Interface with the Action name "EVENT".
12. Label (9): Whenever the GotAPI Server receives a message from the Plug-In, the GotAPI Server passes it to the application on the WebSocket connection.
13. Label (10): When the application finishes or decides to finish using the service, it sends a request to stop the monitoring to the GotAPI Server. The request is sent over the GotAPI-1 Interface using HTTP. Note that the URI is the same as that of the first request except that the HTTP method is "DELETE".
14. Label (11): When the GotAPI Server receives the stop request, it sends a request to the Plug-In to stop the monitoring with the Action name "DELETE". Then the GotAPI server closes the WebSocket connection.
15. Label (12): When the Plug-In receives the stop request from the GotAPI Server, the Plug-In stops reporting messages, and it returns a response to the GotAPI Server on the GotAPI-4 Interface with the Action name “RESPONSE”.
16. Label (13): When the GotAPI Server receives the response, the GotAPI Server passes the response to the application on the GotAPI-1 Interface.

The diagram above shows that the application establishes a WebSocket connection as the GotAPI-5 Interface after the application sends an API request on the GotAPI-1 Interface. It should be noted, as defined in GotAPI 1.1, the application is permitted to establish a WebSocket connection only after the application has received an access token from the GotAPI Auth Server.

The overall message flows to establish/close an asynchronous messaging session and to receive measurement values asynchronously from Plug-Ins SHALL adhere to the specifications defined in GotAPI 1.1.

The specific data in the message flows labelled (1) to (13) in the figure above are defined by the Plug-In that implements manager functions of 3D printer and is communicating with 3D printing devices.

### 7.2.4 Service Connecting API

After the completion of the Service Discovery, the application can get the information of a contents service (so called "Service Connecting API") provided by the Plug-In through the GotAPI Server.

To provide the information to the application, the Plug-in and the Contents Server need a pre-handling process to send the information to the Plug-in from the Contents Server. After that, the application and the Plug-in do the service connecting using the Service Connecting API. The application gets the Contents Server information from the Plug-in via the GotAPI Server and makes a request to the Contents Server for getting a 3D model file list without the GotAPI Server intervention.

For the following advantage, the Service Connecting flow has two additional out-of-scope processes such as pre-handling and requesting 3D model file list:

- No need to define all the format to handle between the 3DP application and the Contents Server in DWAPI-3DP. The Contents Server could provide various contents description and option

The Service Connecting API offers a list of contents server reported by the Plug-In of the 3D printer as a targeted device in response to a request. The message flow of this API is as shown below.

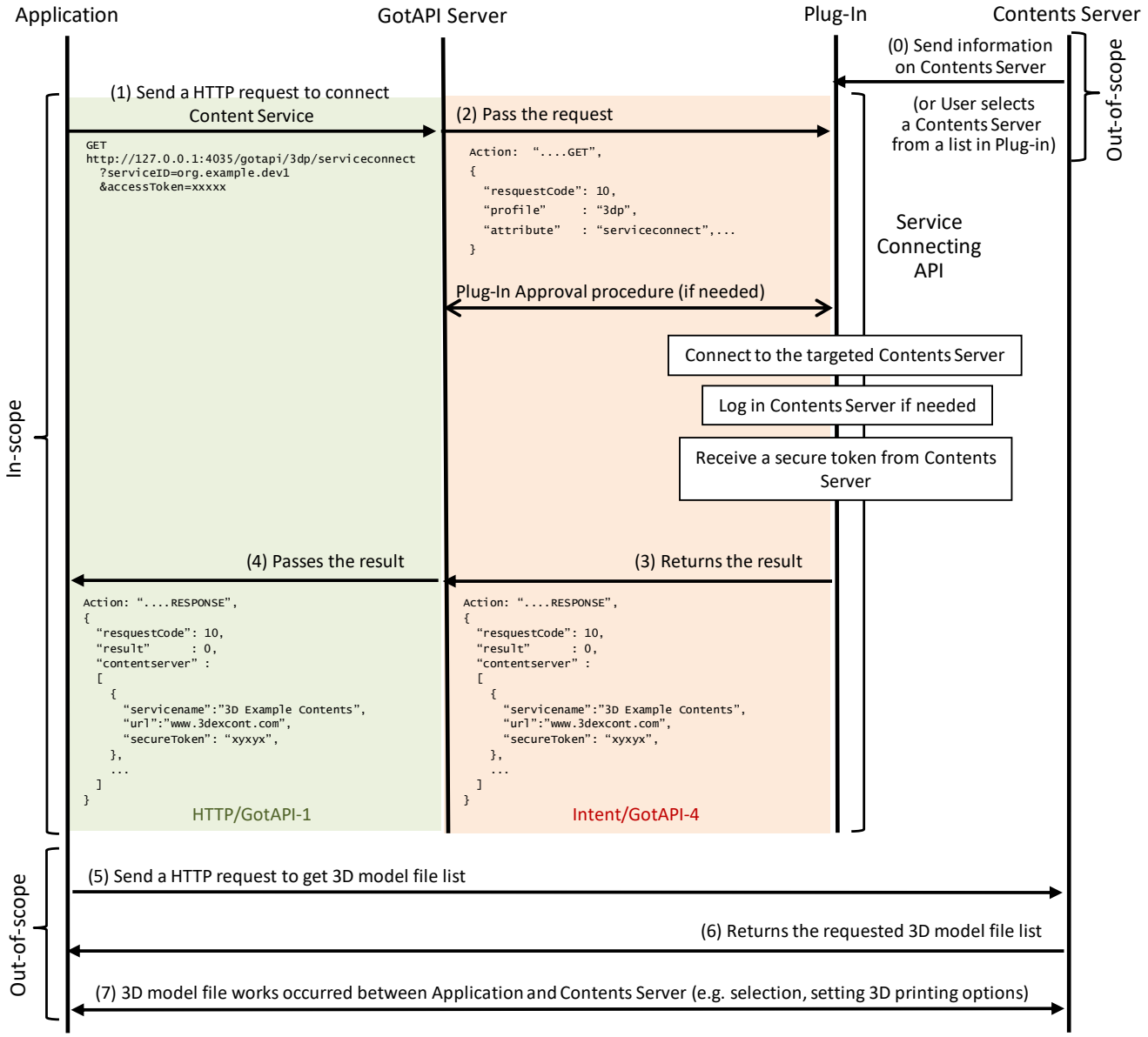


**Figure 9: Message flow of Service Connecting API**

1. Label (0:out-of-scope) The Contents Server sends its information to the Plug-in before the following application request.
2. The user triggers a request of the API in the application.
3. Label (1): The application sends a request to the GotAPI Server using HTTP (REST) over the GotAPI-1 Interface. Note that the HTTP method of the request is "GET".
4. Label (2): The GotAPI Server passes the request to the targeted Plug-In on the GotAPI-4 Interface with the Action name "GET".
5. The GotAPI Server runs the Plug-In Approval procedure if needed, which is defined in the GotAPI 1.1 specification.
6. When the Plug-In receives the request, it connects to the targeted Contents Server.
7. The Plug-In performs log-in process of the contents server if it is requested by the server. During the login-in process, the Plug-In opens a log-in window to get identification information from user, if it is needed.

8. The Plug-In obtains a secure token from the Contents Server.
9. Label (3): The Plug-In sends a response with one set of the list of Contents Server using the GotAPI-4 Interface.
10. Label (4): When the GotAPI Server receives the response from the Plug-In, the GotAPI Server passes the response to the application on the GotAPI-1 Interface as an HTTP response.
11. Label(5; out-of-scope): The application sends a HTTP request to the Contents Server directly to get a 3D model file list using the information which are provided by GotAPI-1..
12. Label (6; out-of-scope): The Contents Server returns the requested 3D model file list to the application.
13. Label (7; out-of-scope): The user does 3D model file works such as selection of file, setting 3D printing options, etc.

The overall message flows to obtain data by sending HTTP request and response over the GotAPI-1 Interface SHALL adhere to the specifications defined in GotAPI 1.1.

The specific data in the message flows labelled (3) and (4) in the figure above are defined by the Plug-In that implements manager functions of 3D printer and is communicating with 3D printing devices.

### 7.2.5 Printing Command API

After the selecting of 3D contents to print through the Service Connecting API, the application can make the 3D Printer to print the contents (so called "Printing Command API") provided by the Plug-In through the GotAPI Server.

The Printing Command API offers a list of printing files reported by the Plug-In of the 3D printer as a targeted device in response to a request. The message flow of this API is as shown below.

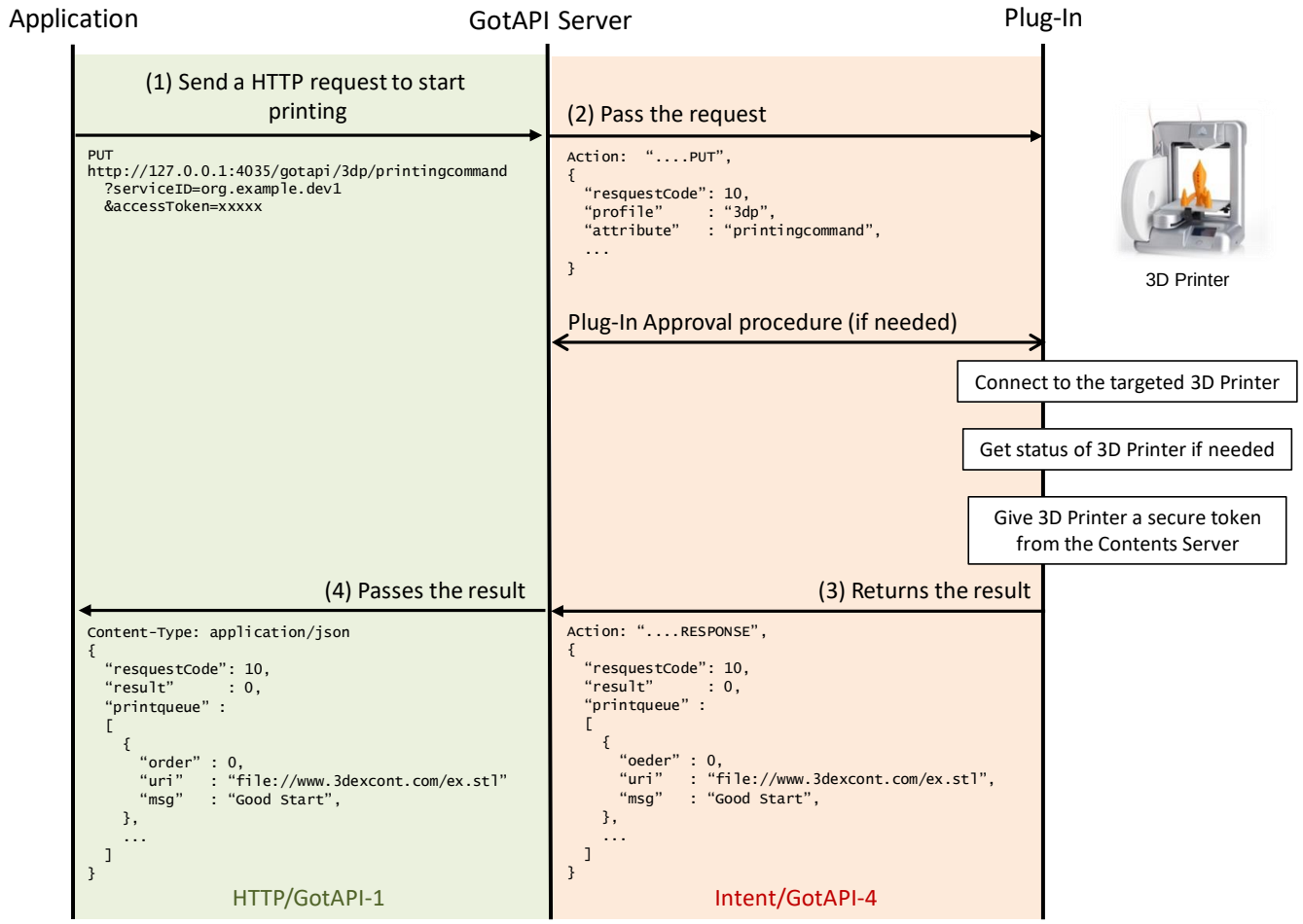


**Figure 10: Message flow of Printing Command API to start**

1. The user triggers a request of the API in the application.
2. Label (1): The application sends a request to the GotAPI Server using HTTP (REST) over the GotAPI-1 Interface. Note that the HTTP method of the request is "PUT".
3. Label (2): The GotAPI Server passes the request to the targeted Plug-In on the GotAPI-4 Interface with the Action name "PUT".
4. The GotAPI Server runs the Plug-In Approval procedure if needed, which is defined in the GotAPI 1.1 specification.
5. When the Plug-In receives the request, it connects to the targeted 3D Printer.
6. The Plug-In gets status of the printer if it is needed.
7. The Plug-In obtains information of printing queue the printer.
8. Label (3): The Plug-In sends a response with one set of the list of printing queue information using the GotAPI-4 Interface.
9. Label (4): When the GotAPI Server receives the response from the Plug-In, the GotAPI Server passes the response to the application on the GotAPI-1 Interface as an HTTP response.

The overall message flows to obtain data by sending HTTP request and response over the GotAPI-1 Interface SHALL adhere to the specifications defined in GotAPI 1.1.

The specific data in the message flows labelled (3) and (4) in the figure above are defined by the Plug-In that implements manager functions of 3D printer and is communicating with 3D printing devices.

During a 3D printing, there is possibility of failure of printing due to many reasons, e.g., run out of filament. After the application receives the error message with Asynchronous messaging API, the application sends the Printing Command to stop printing.

The Printing Command API offers a list of printing files reported by the Plug-In of the 3D printer as a targeted device in response to a request. The message flow of this API is as shown below.

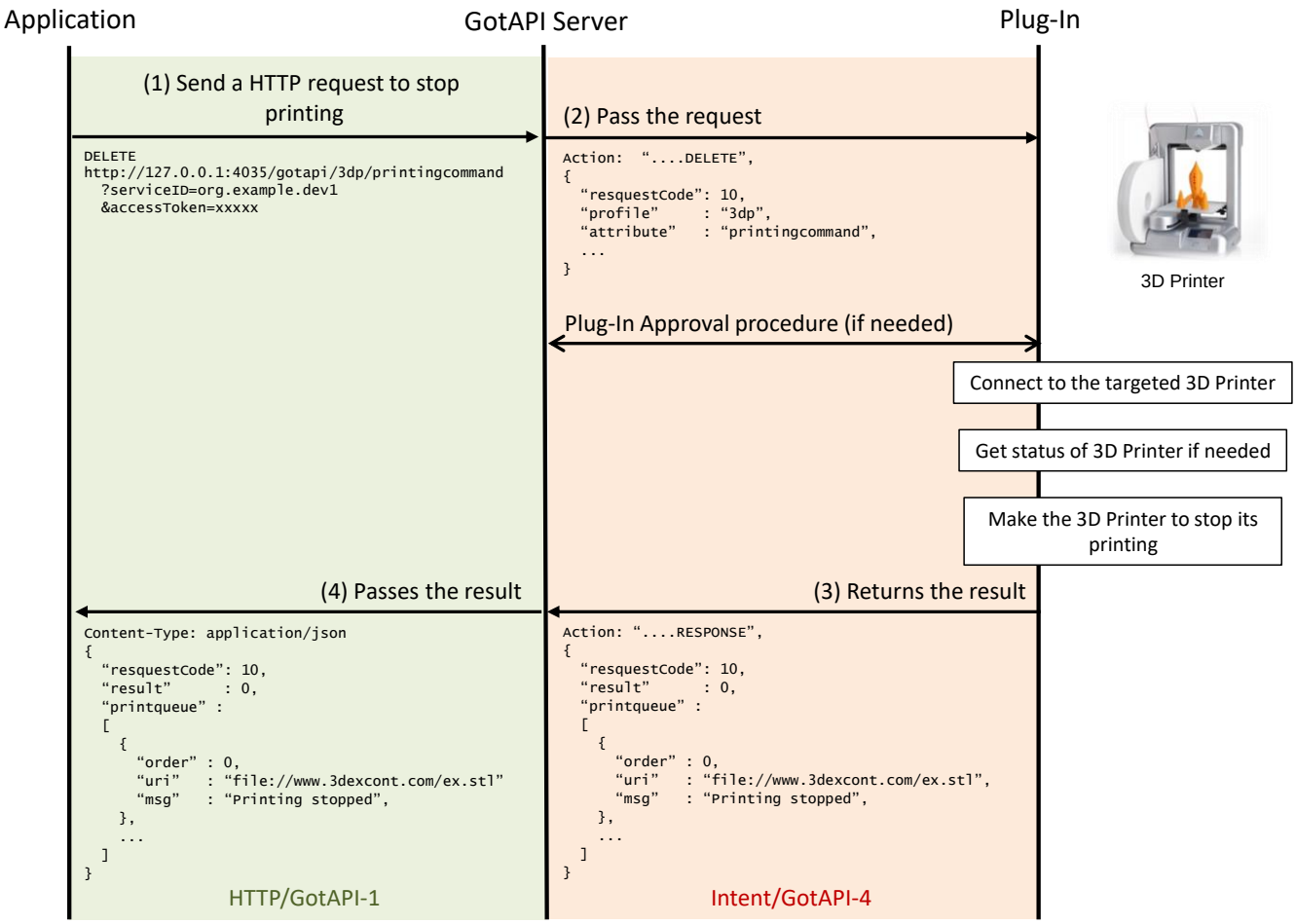


**Figure 11: Message flow of Printing Command API to stop**

1. The user triggers a request of the API in the application.
2. Label (1): The application sends a request to the GotAPI Server using HTTP (REST) over the GotAPI-1 Interface. Note that the HTTP method of the request is "DELETE".
3. Label (2): The GotAPI Server passes the request to the targeted Plug-In on the GotAPI-4 Interface with the Action name "DELETE".
4. The GotAPI Server runs the Plug-In Approval procedure if needed, which is defined in the GotAPI 1.1 specification.
5. When the Plug-In receives the request, it connects to the targeted 3D Printer.
6. The Plug-In gets status of the printer if it is needed.
7. The Plug-In makes the 3D printer to stop its printing.
8. Label (3): The Plug-In sends a response with one set of the list of printing queue information using the GotAPI-4 Interface.
9. Label (4): When the GotAPI Server receives the response from the Plug-In, the GotAPI Server passes the response to the application on the GotAPI-1 Interface as an HTTP response.

The overall message flows to obtain data by sending HTTP request and response over the GotAPI-1 Interface SHALL adhere to the specifications defined in GotAPI 1.1.

The specific data in the message flows labelled (3) and (4) in the figure above are defined by the Plug-In that implements manager functions of 3D printer and is communicating with 3D printing devices.

## 7.2.6 Printer Authentication API

After the selecting of 3D contents to print through the Service Connecting API, the application may make the 3D Printer to print the contents.

Between the service connecting and the printing command, the 3D Printer should connect the Contents Server by itself to get the content and its information to start printing.

During the Service Connecting process, the 3D Printer Plug-In obtains a Secure Token from the Contents Server, and the token will be send to the printer to obtain the authentication from the server. The message flow of this API is as shown below.

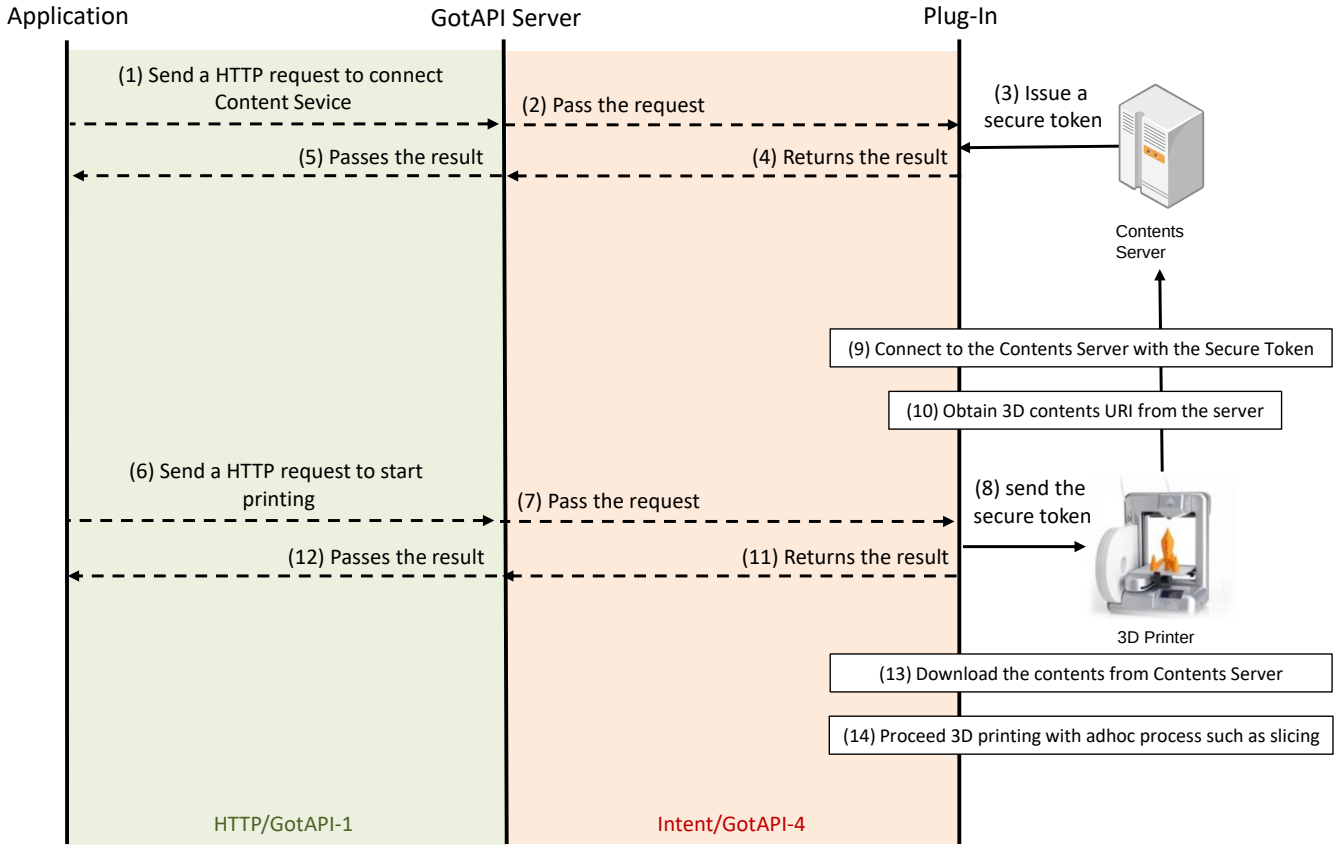


**Figure 12: Message flow of Printer Authentication API**

1. Label (1~2): The Service Connecting API is requested by the the application.
2. Label (3): During the process of Service Connecting API, a Secure Token is issued by the Contents Server for the usage of Printing Command API. The Secure Token is used by the application to get contents information from the server.
3. Label (4~5): The application receives the response of the Service Connecting API from the Plug-In.
4. Label (6~7): The application sends a Printing Command request to the Plug-In.
5. Label (8): The Plug-In sends the 3D Printer the Secure Token to connect the Contents Server.
6. Label (9~10): The 3D Printer connects the server with the Secure Token, and obtains the URI of 3D contents that is pre-selected by the application.
7. Label (11~12): The application receives the response of the Printing Command API from the Plug-In
8. Label (13~14): The 3D Printer downloads the contents and proceeds next step such as starting to print the contents with adhoc processing, e.g. slicing.

The overall message between the application and the Plug-In flows to obtain data by sending HTTP request and response over the GotAPI-1 Interface SHALL adhere to the specifications defined in GotAPI 1.1.

The definition of other messages are out of scope of this specification, and is depended on the implementation of each Plug-Ins and 3D printers that are supporting this specification. The implementation of the messages is highly recommended to use the web interface technology which is used in this specification as well.

### 7.2.7 Printer Status API

Before the start 3D printing or during the 3D printing, there is some need to catch the status of 3D printer including the information on printing list.

The 3D printing application can get the status of the 3D Printer through the GotAPI Server.

The Printer Status API provides a list of the printer status reported by the Plug-In of the 3D printer as a targeted device in response to a request. The message flow of this API is as shown below.

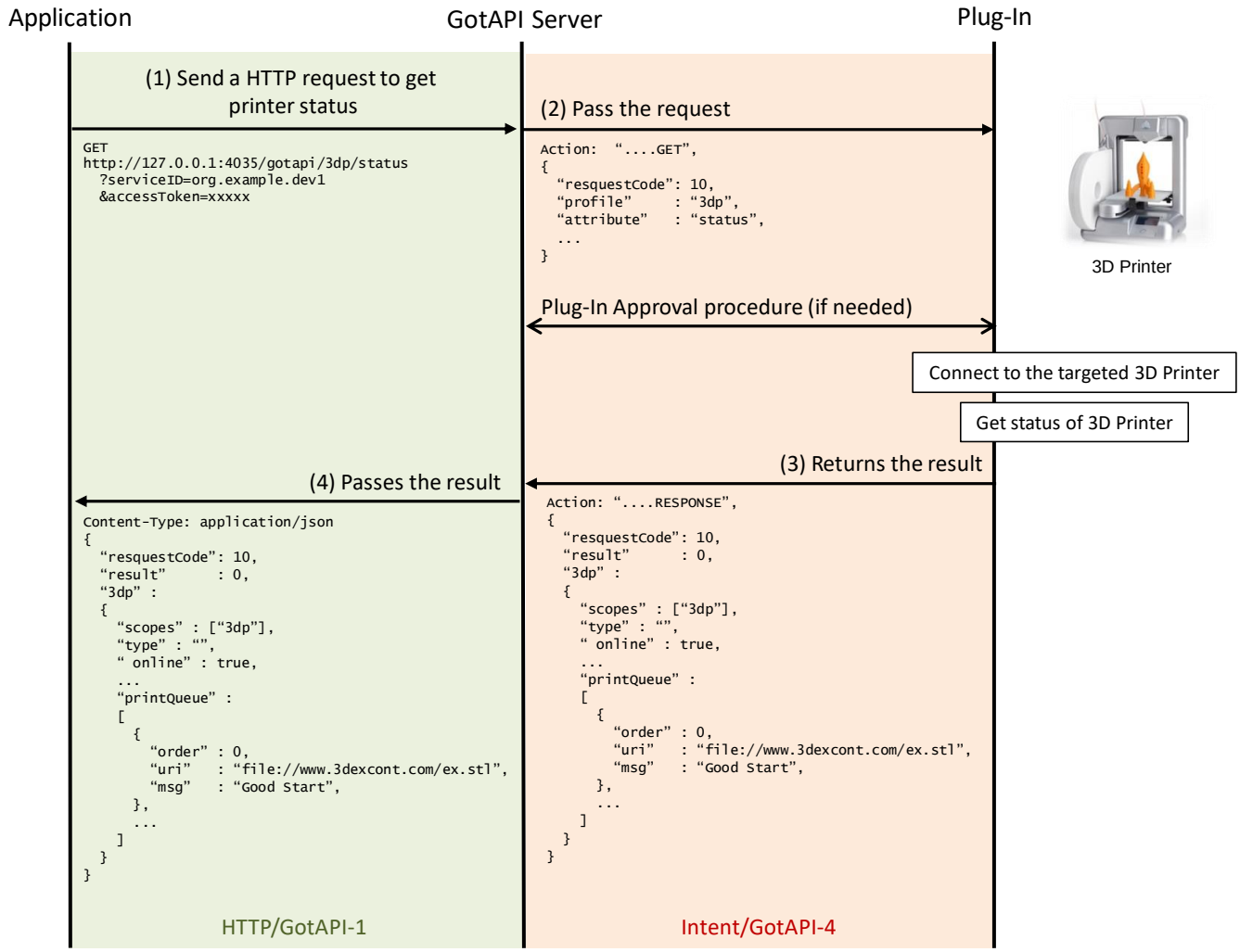


**Figure 13: Message flow of Printer Status API**

1. The user triggers a request of the API in the application.
2. Label (1): The application sends a request to the GotAPI Server using HTTP (REST) over the GotAPI-1 Interface. Note that the HTTP method of the request is "GET.
3. Label (2): The GotAPI Server passes the request to the targeted Plug-In on the GotAPI-4 Interface with the Action name "GET".
4. The GotAPI Server runs the Plug-In Approval procedure if needed, which is defined in the GotAPI 1.1 specification.
5. When the Plug-In receives the request, it connects to the targeted 3D Printer.

6. The Plug-In gets status of the printer and information of printing queue the printer.
7. Label (3): The Plug-In sends a response with one set of the status and printing queue information using the GotAPI-4 Interface.
8. Label (4): When the GotAPI Server receives the response from the Plug-In, the GotAPI Server passes the response to the application on the GotAPI-1 Interface as an HTTP response.

The overall message flows to obtain data by sending HTTP request and response over the GotAPI-1 Interface SHALL adhere to the specifications defined in GotAPI 1.1.

The specific data in the message flows labelled (3) and (4) in the figure above are defined by the Plug-In that implements manager functions of 3D printer and is communicating with 3D printing devices.

# 7.3 Behaviors of Plug-Ins for reporting measurements to applications

## 7.3.1 Measurement modes and one shot/asynchronous messaging

There are two measurement modes for 3D printer (3DP) devices, (i) the single measurement mode and (ii) the continuous measurement mode. Depending on the nature of measurements that the device generates, some support only one mode while others support both modes.

The single measurement mode provides only one measurement per one measurement process, whereas the continuous measurement mode provides multiple measurements continuously. An example of the single measurement is getting printer information, which provides printer's initial information. An example of continuous measurement mode is printer command, which provides printing status information periodicallyand continuously.

Table 3 shows measurement mode support for 3D Printer device.

| | Measurement mode | |
|---|---|---|
| | Single | Continuous |
| 3D Printer | Yes | Yes |

**Table 3: Measurement modes for 3D Printer**

Generally, the implementation of 3DP devices measure and provide the data in the following steps:

1. The user completes pairing of a 3DP device and a smartphone. Pairing must be completed before any measurements or data transfer to take place.
2. The user starts up the application on the smartphone, and takes a measurement using the 3DP device.

3A. Single measurement mode:

When the 3DP device acquires a measurement result successfully;

(i) the connection between the 3DP device and the smartphone (i.e., the Plug-In) is established,

(ii) the measurement result is sent to the smartphone, and

(iii) the connection is closed by the 3DP device.

3B. Continuous measurement mode:

When the 3DP device acquires a measurement result successfully;

(i) the connection between the 3DP device and the smartphone (i.e., the Plug-In) is established,

(ii) the measurement result is sent to the smartphone,

(iii) the series of results that are continuously measured afterward are sent to the smartphone one after another whenever they are acquired by the 3DP device, e.g., every one second, and

(iv) the connection is closed by the 3DP device, e.g., the user stopped measurement with the 3DP device or an error occurred with the measurement.

The Plug-In is not able to detect the status of measurement being conducted by the 3DP device. The Plug-In just receives measurement results only when the measurement succeeds.

### 7.3.2 Policy for one-shot messages

One-shot messages can be sent to a Plug-In by an application arbitrarily at any time without knowing the status of the Plug-In or the measurement that is underway. Therefore, in order to clarify what applications can obtain by use of one-shot messages, the policy for responding to a one-shot API request is specified as follows:

1. If the Plug-In has the latest measurement result, the Plug-In SHALL return the result immediately.
2. Otherwise, the Plug-In SHALL return an error.

This means that if the application receives an error, it may have to send additional one-shot API requests to the Plug-In in order to obtain measurement data for single measurement devices. For continuous measurement devices, the application need to send one-shot API requests one after another in order to get new measurement data that are sent from the 3DP device.

The following figures show examples of responses to one-shot API requests from Plug-Ins depending on various measurement status.

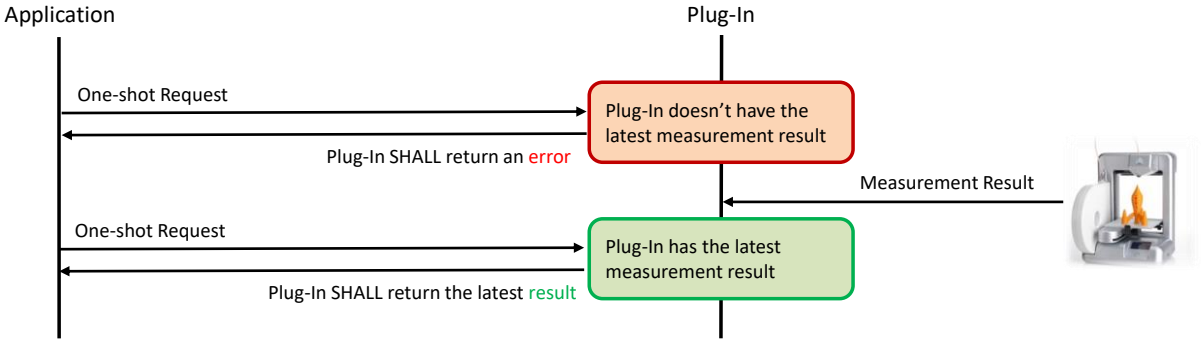


**Figure 14: Example of single measurement**

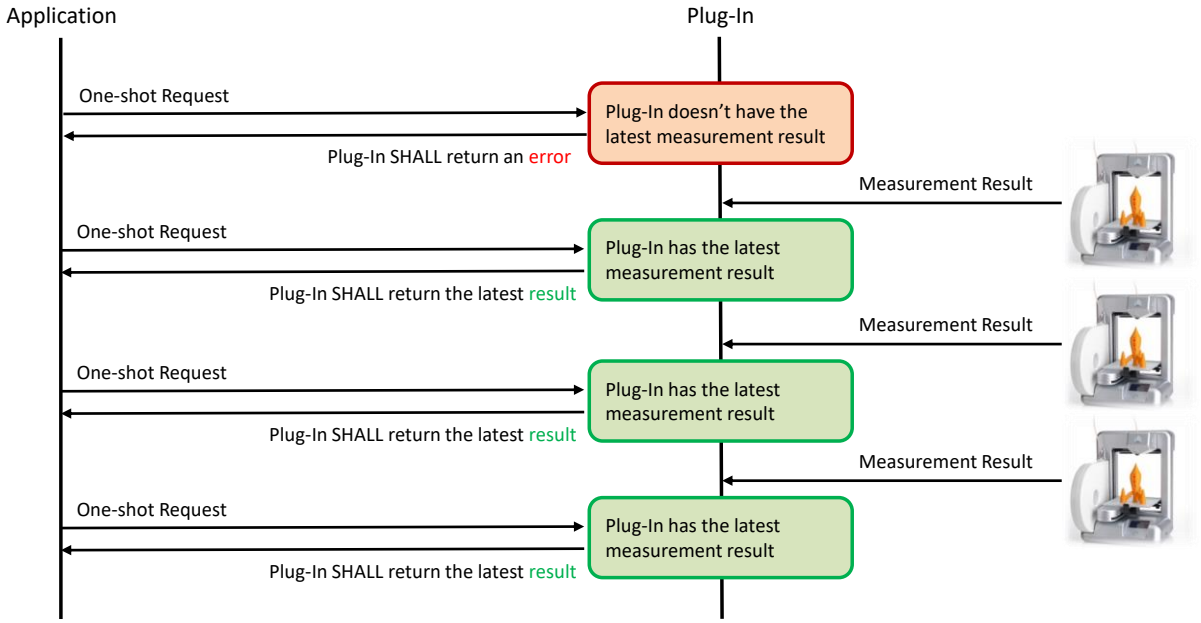


**Figure 15: Example of continuous measurement**

### 7.3.3 Policy for asynchronous messages

When the Plug-In receives a request of asynchronous messaging API over the GotAPI-1 Interface:

- If the Plug-In has the latest result, it SHALL return the result to the application immediately over the GotAPI-5 Interface.
- Otherwise, the Plug-In SHALL NOT send anything.

After the Plug-In receives a request of asynchronous messaging API over the GotAPI-1 Interface:

- Whenever the Plug-In gets the latest result from the connected 3DP device, it SHALL return the result to the application immediately over the GotAPI-5 Interface.
- At other times, the Plug-In SHALL NOT send anything.

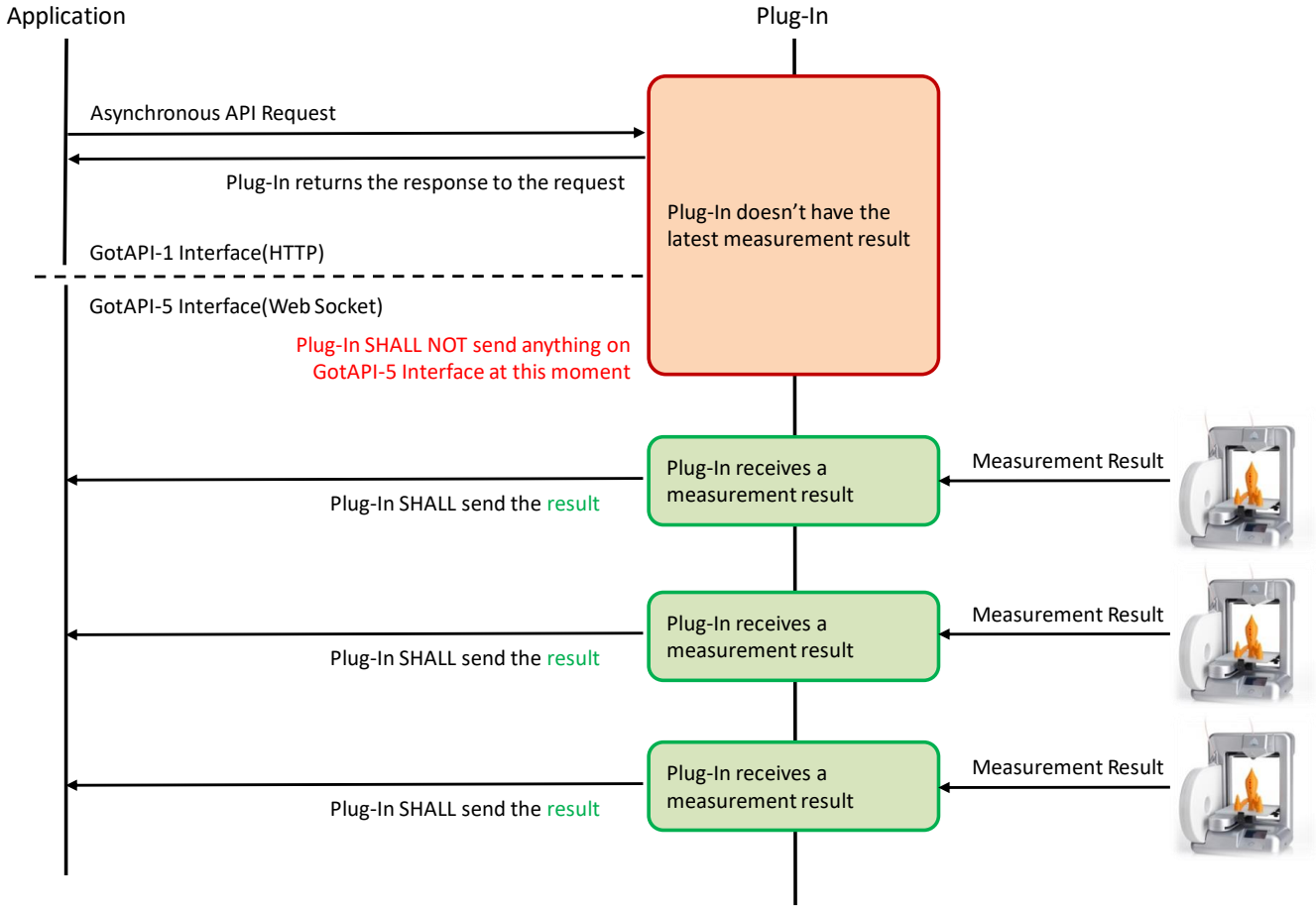


**Figure 16: Example of asynchronous messages (continuous measurement).**

### 7.3.4 Intermediate measurements

## 7.4 Security Considerations

This specification SHALL adhere to all the security requirements that are defined in GotAPI 1.1.

The GotAPI 1.1 specification considers every security risks and implements necessary counter measures for them. For example:

- The GotAPI 1.1 has an application-authorization mechanism. Applications can't access the APIs without user permissions. Besides, when applications access devices via Plug-Ins, the relevant Plug-In obtains a permission from the user.

- The GotAPI 1.1 has an HMAC server authentication mechanism. Applications are able to detect if the GotAPI Server is spoofed.

See the section "7.3 Security Considerations" in the GotAPI 1.1 specification for the details of the security considerations of the GotAPI 1.1.

# Appendix A. Change History (Informative)

## A.1 Approved Version History

| Reference | Date | Description |
|---|---|---|
| OMA-ER-Device_WebAPIs_3DP-V1_0-20181021-A | 21 Oct 2018 | Status changed to Approved by CD<br>Doc Ref # OMA-CD-DWAPI-2018-0001-INP_DWAPI_3DP_V1_0_ERP_for_final_Approval |

# Appendix B. Call Flows (Informative)

This is a placeholder to be populated, as required.

# Appendix C. Static Conformance Requirements (Normative)

The notation used in this appendix is specified in [SCRRULES].

## C.1 ERDEF for Device WebAPI 1.0 - Client Requirements

This section is normative.

| Item | Feature / Application | Requirement |
|---|---|---|
| OMA-ERDEF-Device WebAPIs 1.0-C-001-<<M/O>> | | |

**Table 4: ERDEF**

## C.2 ERDEF for GotAPI 1.0 - Server Requirements

This section is normative.

| Item | Feature / Application | Requirement |
|---|---|---|
| OMA-ERDEF-GotAPI 1.0-S-001-<<M/O>> | GotAPI 1.0 Server | |

**Table 5: ERDEF for GotAPI 1.0 Server-side Requirements**

# Appendix D. Device WebAPI Enabler Deployment Considerations

This is a placeholder, to be populated as required.