Generic Open Terminal API Framework (GotAPI)

Candidate Version 1.0 – 10 Feb 2015

Open Mobile Alliance

OMA-ER-GotAPI-V1_0-20150210-C

Contents

Figures

Tables

1. Scope

This Enabler Release (ER) document is a combined document that includes requirements, architecture and technical specification of the Generic Open Terminal API Framework (GotAPI) Enabler.

The scope of this enabler will include:

- Architecture and specifications for an API framework enabling web-based APIs to be exposed to apps running in web browsers and as native apps (including but not limited to hybrid native/web apps)
- Supporting assets for the localhost API server framework, e.g. JavaScript libraries enabling abstractions of common API functions (e.g. discovery, access, and session management)
- A registry of well-known API resources for OMA enablers, to be maintained as part of the OMNA
- Specification of API exposure patterns that are in general globally applicable to native device platforms

2. References

2.1 Normative References

[CORS]	"Cross-Origin Resource Sharing", Worldwide Web Consortium (W3C), URL: http://www.w3.org/TR/cors/
[HTTP/1.1]	"Hypertext Transfer Protocol -- HTTP/1.1", Internet Engineering Task Force (IETF), URL: http://tools.ietf.org/search/rfc2616
[HTTP2]	"Hypertext Transfer Protocol version 2", Internet Engineering Task Force (IETF), URL: https://tools.ietf.org/html/draft-ietf-httpbis-http2
[JSON-RPC]	"JSON-RPC 2.0 Specification", JSON-RPC Working Group, URL: http://www.jsonrpc.org/specification
[OAuth2.0]	"The OAuth 2.0 Authorization Framework", Internet Engineering Task Force (IETF), URL: http://tools.ietf.org/html/rfc6749
[OMA DM]	"OMA Device Management V2.0", Open Mobile Alliance™, 10 Dec 2013, URL: http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-device-management-v2-0
[RFC2119]	"Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997, URL:http://www.ietf.org/rfc/rfc2119.txt
[RFC6454]	"The Web Origin Concept", Internet Engineering Task Force (IETF), URL: http://tools.ietf.org/html/rfc6454
[Rtcweb]	"Rtcweb", Internet Engineering Task Force (IETF), URL: http://tools.ietf.org/wg/rtcweb/
[SCRRULES]	"SCR Rules and Procedures", Open Mobile Alliance™, OMA-ORG-SCR_Rules_and_Procedures, URL:http://www.openmobilealliance.org/
[SSE]	"Server-Sent Events", Worldwide Web Consortium (W3C), URL: http://www.w3.org/TR/eventsource/
[WebRTC]	"WebRTC 1.0: Real-time Communication Between Browsers", Worldwide Web Consortium (W3C), URL: http://www.w3.org/TR/webrtc/
[WebSocket]	"The WebSocket API", Worldwide Web Consortium (W3C), URL: http://www.w3.org/TR/websockets/
[WebSocketProtocol]	"The WebSocket Protocol", Internet Engineering Task Force (IETF), URL: https://tools.ietf.org/html/rfc6455
[XHR]	"XMLHttpRequest Level 1", Worldwide Web Consortium (W3C), URL: http://www.w3.org/TR/XMLHttpRequest/

2.2 Informative References

[OMADICT]	"Dictionary for OMA Specifications", Version 2.8, Open Mobile Alliance™, OMA-ORG-Dictionary-V2.8, URL:http://www.openmobilealliance.org/
[OMNA]	"OMA Naming Authority". Open Mobile Alliance™. URL:http://www.openmobilealliance.org/tech/omna.aspx
[RFC6454]	"The Web Origin Concept", URL: https://www.ietf.org/rfc/rfc6454.txt
[RFC6973]	"Privacy Considerations for Internet Protocols", A. Cooper, et al, July 2013, URL:http://tools.ietf.org/html/rfc6973

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

API Patterns	Design guidelines and requirements for definition of APIs
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].
Native App	An application designed to execute under the native OS / middleware environment of a device.
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).
WebSocket	An API providing networking services per the WebSocket standard [WebSocket].

3.3 Abbreviations

API	Application Programming Interface
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
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
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WRAPI	The OMA Web Runtime API enabler

4. Introduction

The concept of a common OMA device API framework, through which OMA enablers can be exposed to applications executing in various execution environments, has been a discussion thread in OMA for several years. For the web, the discussion started with Client Side Enabler API [CSEA] work item, and followed by the Web Runtime API [WRAPI] work item which established an initial pattern for OMA API exposure to web applications, focused on the requirements of the OMA Push enabler. The need for more broadly applicable API patterns was recognized during development of APIs for the Mobile Codes 2.0 [MCAPI] enabler and the Open Connection Management 1.1 API [CMAPI] enabler. Interest in incorporating the WRAPI local API server concept has further been expressed for the RCS-enabling enablers of COM, and the Device Management enablers, among others.

This specification defines a variety of API exposure patterns for use in development of OMA enablers, and the functions required by API servers that support those exposure patterns. APIs designed per these patterns are intended to be "web-based" (i.e. accessed via the use of web technologies) and exposed to apps running in the following contexts:

- Web Browser apps, i.e. web apps running in a window of a web browser
- Web Runtime apps, i.e. web apps running outside a browser, e.g. under a Widget Engine or other "chromeless" runtime for execution of web content as standalone apps
- Hybrid Native/Web apps, i.e. apps that run web content through native APIs for that purpose
- Native Apps that directly use native platform APIs enabling use of the web-related protocols described in this document. Though not leveraging a full web execution environment, such apps can use the same network-based APIs as web apps.

For simplicity, the web API client environment provided by each of these contexts is referred to here as the User Agent (UA). The OMA enabler clients that expose APIs via the GotAPI patterns are referred to here as the GotAPI Server. GotAPI Servers may also act as OAuth servers for other GotAPI Servers, and in that role are referred to here as GotAPI Auth Servers.

The web-based methods defined by GotAPI are intended to offer a flexible set of options for OMA enablers to expose their services to apps via web-based APIs. Such APIs are primarily intended to be exposed to apps running in the device hosting the OMA enabler client, but in principle could also be exposed to apps in other devices that are networked with the OMA enabler client host device.

The exposure of OMA enabler-based services via such web-based APIs is intended to broaden the reach of OMA enabler deployments, by making it possible for web apps to access them, without explicit UA support of APIs specifically designed per the requirements of OMA enablers.

The figure below illustrates the relationships and conceptual interfaces between web apps, the UA, GotAPI-specified functions (shaded), and other OMA-specified functions.

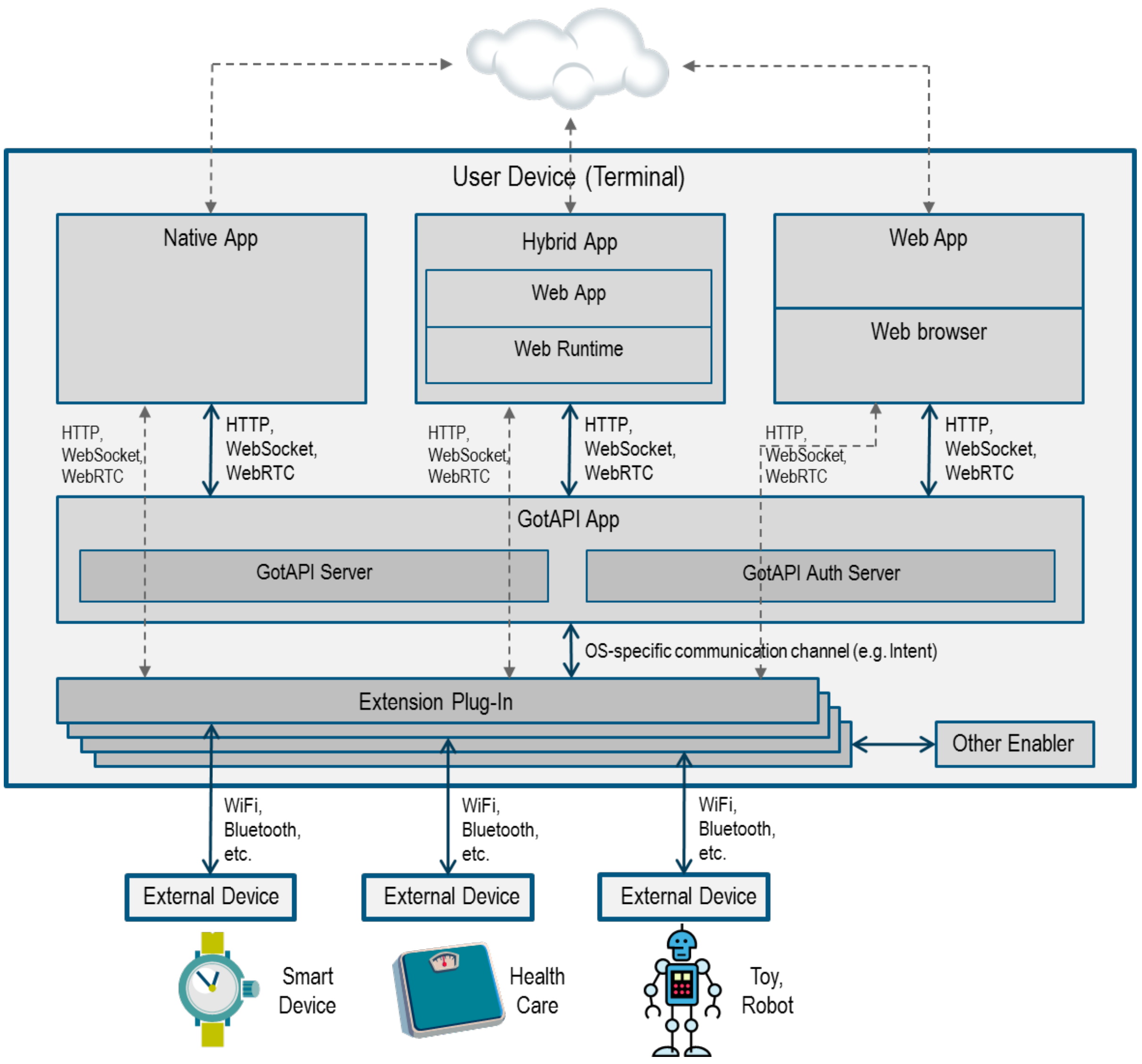


Figure 1: Conceptual Implementation (Informative)

In the figure above:

- The GotAPI functions include the GotAPI Server (including optionally acting as a distinct Auth server) and a database of API access permissions.
- The device OS provides intra-device and inter-device communication via network protocols such as HTTP, WebSocket, Server-Sent Events, and WebRTC, via which the web-based APIs can be exposed.
- The access permissions database is logically specified per its supporting operations and policy structure, but interfaces to it are unspecified by GotAPI.
- Interfaces to other OMA enabler clients are also unspecified.
- GotAPI Servers may expose APIs for multiple OMA enablers, and either directly implement the related OMA enabler functions or as above use unspecified interfaces exposed by the specific OMA enabler clients.

- User interface functions can include a variety of means for assessing user consent for API access by apps, including basic means such as device display and keyboard, or more advanced means such as Trusted User Interfaces (TUI) or biometrics.

4.1 Version 1.0

GotAPI version 1.0 includes the functionality:

- Architecture and specifications for GotAPI Servers and GotAPI Auth Servers in an API framework enabling web-based APIs to be exposed apps running in web browsers and as native apps (including but not limited to hybrid native/web apps)
- Supporting assets for the localhost API server framework, e.g. JavaScript libraries enabling abstractions of common API functions (e.g. discovery, access, and session management)
- A registry of well-known API resources for OMA enablers, to be maintained as part of the OMNA
- Specification of API exposure patterns that are in general globally applicable to native device platforms

5. GotAPI Enabler release description (Informative)

This release focuses on:

- the functions of GotAPI Servers and GotAPI Auth Servers, through which OMA enabler based services can be exposed and access to the APIs managed
- the Extension Plug-Ins for external devices and internal enablers through which they communicate with the GotAPI Server

6. Requirements (Normative)

6.1 High-Level Functional Requirements

The following requirements outline the high-level set of options that GotAPI Servers may implement. The GotAPI technical specifications will address the necessary functions for support of these options.

Label	Description	Release
GotAPI-HLF-01	GotAPI Servers SHALL support APIs exposed via HTTP/1.1 [HTTP/1.1].	1.0
GotAPI-HLF-02	GotAPI Servers SHALL support APIs exposed via custom URI scheme handlers.	>1.0
GotAPI-HLF-03	GotAPI Servers MAY support APIs exposed via the WebSocket API [WebSocket].	1.0
GotAPI-HLF-04	GotAPI Servers MAY support APIs exposed via Server-Sent Events [SSE].	1.0
GotAPI-HLF-05	GotAPI Servers MAY support APIs exposed via the WebRTC API [WebRTC].	1.0
GotAPI-HLF-06	GotAPI Servers MAY support APIs exposed via HTTP/2.0 [HTTP/2.0].	1.0
GotAPI-HLF-07	GotAPI Servers SHALL support APIs exposed using the REST design pattern.	1.0
GotAPI-HLF-08	GotAPI Servers MAY support APIs exposed using the RPC design pattern, including APIs exposed using JSON-RPC 2.0 [JSON-RPC] as payload protocol.	1.0
GotAPI-HLF-09	GotAPI Servers MAY support APIs that include transfer of any discrete media type.	1.0
GotAPI-HLF-10	GotAPI Servers MAY support APIs that include transfer of any streamed media type.	1.0
GotAPI-HLF-11	GotAPI Servers SHALL expose APIs to UAs in the GotAPI Server host device.	1.0
GotAPI-HLF-12	GotAPI Servers MAY expose APIs to UAs in devices other than the GotAPI Server host device.	>1.0
GotAPI-HLF-13	Multiple GotAPI Servers SHALL be implementable and functional simultaneously on a device where possible.	>1.0
GotAPI-HLF-14	GotAPI SHALL support an API that provides applications with availability of GotAPI in the device in response to query requests from the application.	1.0
GotAPI-HLF-15	GotAPI Servers SHALL support invocation via a custom URI scheme, to startup the server when it is not running.	1.0
GotAPI-HLF-16	GotAPI Server SHALL expose an interface to communicate with external devices and internal enablers, so that different manufacturers are able to develop Extension Plug-Ins for the GotAPI Server, and application developers are able to develop applications that can communicate with such other external devices and internal enablers through GotAPI Server. Note: The APIs for each device or enabler are out of the scope of this specification.	1.0

Table 1: High-Level Functional Requirements

6.1.1 Security and Privacy

The following requirements address the generic security and privacy enabling features of GotAPI Servers.

Label	Description	Release
GotAPI-SEC-01	For clients in the host device, GotAPI Servers MAY support APIs exposed over TLS 1.2-secured connections.	>1.0
GotAPI-SEC-02	For clients in other devices, GotAPI Servers SHALL support APIs exposed over TLS 1.2-secured connections.	>1.0
GotAPI-SEC-03	GotAPI Servers SHALL support measures to minimize security risks including Intrusion and Denial-of-Service attacks.	1.0

Table 2: High-Level Functional Requirements – Security and Privacy Items

6.1.1.1 Authentication and Authorization

The following requirements address the ability of GotAPI Servers to identify API client apps and manage access to APIs.

Label	Description	Release
GotAPI-AUTH-01	GotAPI Servers SHALL support Cross-Origin Resource Sharing.	1.0
GotAPI-AUTH-02	GotAPI Servers SHALL support management of API access permissions.	1.0
GotAPI-AUTH-03	GotAPI Servers MAY support OAuth-based API access.	1.0
GotAPI-AUTH-04	GotAPI Servers MAY act as an OAuth 2.0 [OAuth2.0] server for authorization of API access permissions.	1.0
GotAPI-AUTH-05	GotAPI Servers MAY support user interfaces (UI) via which users authorize API access permissions.	1.0
GotAPI-AUTH-06	GotAPI Servers MAY support pre-configured, fixed API access permissions.	1.0
GotAPI-AUTH-07	GotAPI Servers MAY support dynamic, updatable API access permissions.	1.0
GotAPI-AUTH-08	GotAPI Servers MAY support API access permissions managed through OMA Device Management.	>1.0

Table 3: High-Level Functional Requirements – Authentication and Authorization Items

6.1.1.2 Data Integrity

The following requirements address the ability of GotAPI Servers to protect the integrity of data transferred via APIs.

Label	Description	Release
GotAPI-DATI-01	GotAPI Servers SHOULD support data integrity for all data exchanged with clients.	1.0
GotAPI-DATI-02	GotAPI Servers SHOULD support data integrity verification via digitally signed API request/response payloads,	1.0

Table 4: High-Level Functional Requirements – Data Integrity Items

6.1.1.3 Confidentiality

The following requirements address the ability of GotAPI Servers to protect the confidentiality of data transferred via APIs.

Label	Description	Release
GotAPI-CONF-01	GotAPI Servers SHOULD support confidentiality for all data exchanged with clients.	1.0
GotAPI-CONF-02	GotAPI Servers SHOULD support data confidentiality via encrypted API request/response payloads,	1.0

Table 5: High-Level Functional Requirements – Confidentiality Items

7. Architectural Model

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

The architecture definition and functionalities are based on the requirements defined in the Section 6.

7.1 Architectural Diagram

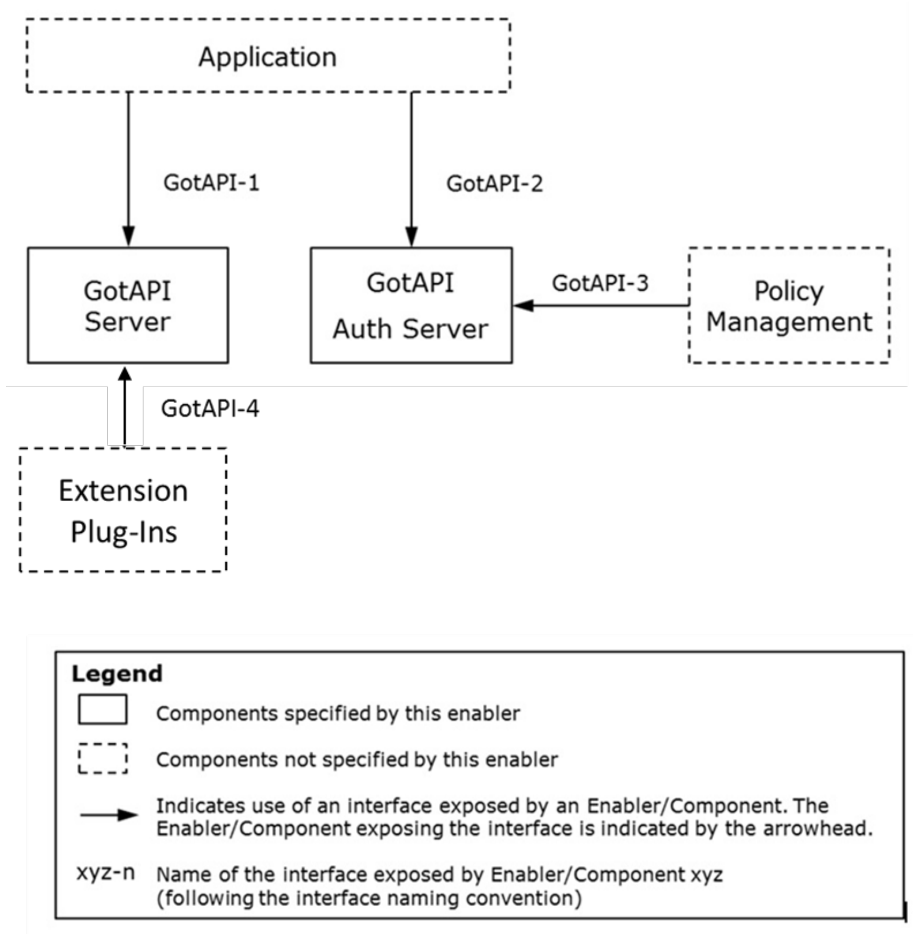

Application
GotAPI-1
GotAPI-2
GotAPI Server
GotAPI Auth Server
GotAPI-3
Policy Management
GotAPI-4
Extension Plug-Ins
Legend
Components specified by this enabler
Components not specified by this enabler
Indicates use of an interface exposed by an Enabler/Component. The Enabler/Component exposing the interface is indicated by the arrowhead.
xyz-n
Name of the interface exposed by Enabler/Component xyz (following the interface naming convention)

Figure 2: GotAPI Architectural Diagram

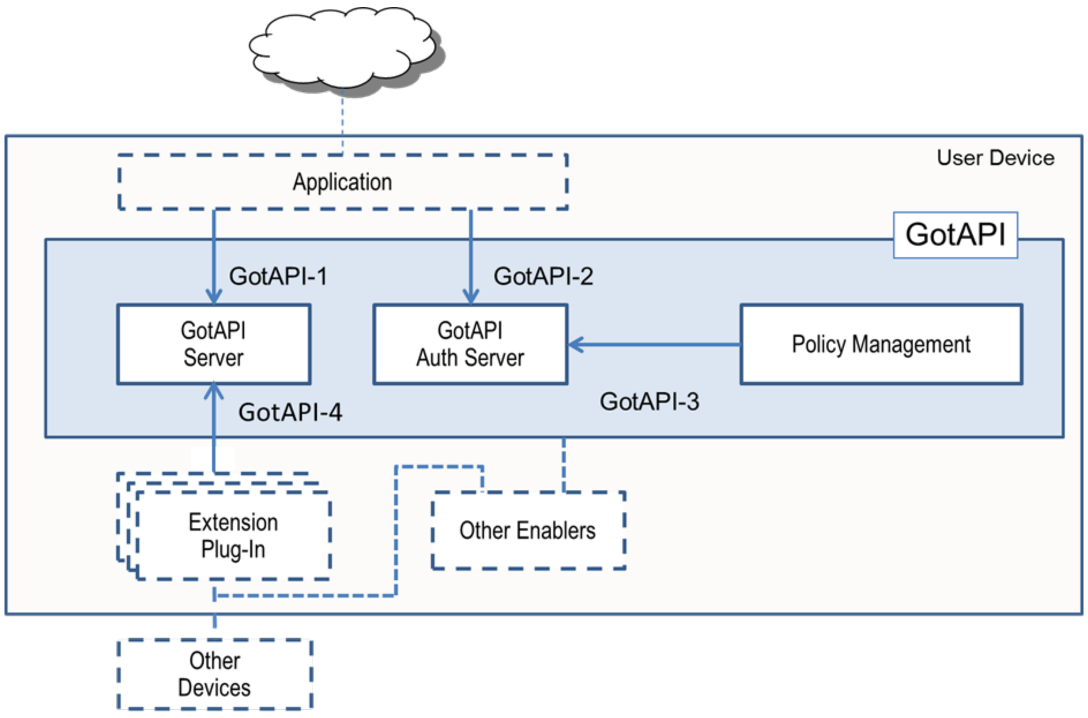


Figure 3: GotAPI Architectural Diagram (Informative)

The diagrams above are not implementation diagrams but logical diagrams. In practice, the GotAPI Server and the GotAPI Auth Server are implemented as a single application which is called as "GotAPI application". The GotAPI 1.0 specification assumes this implementation model. Namely, both servers listen to the same port number on one IP address.

Theoretically, it's possible to implement the GotAPI Server and the GotAPI Auth Server as separate applications. But this implementation model is not supported in the GotAPI 1.0 specification.

7.2 Functional Components and Interfaces/reference points definition

7.2.1 Functional Components

7.2.1.1 GotAPI Server

The GotAPI Server provides the following functions:

- Exposure of the GotAPI-1 interface, via which Applications can issue API requests and receive responses
- Binding of the GotAPI-1 interface to various specific interface technologies and payload protocols / design patterns.
- Security and privacy protection for requests via the GotAPI-1 interface
- Protection of the GotAPI-1 interface from server spoofing, Intrusion and Denial of Service attacks
- Exposure of the GotAPI-4 interface, via which Extension Plug-Ins can receive API requests from applications and send responses to applications through the GotAPI Server
- Security and privacy protection for requests via the GotAPI-4 interface
- Protection of the GotAPI-4 interface from Intrusion and Denial of Service attacks

Web applications running on web browsers have to use the XMLHttpRequest to send requests to the GotAPI Server on the GotAPI-1 interface. The origin of the web application is different from the origin of the GotAPI Server. Therefore, the

GotAPI Server SHALL support Cross-Origin Resource Sharing [CORS] so that the web browser allows the web application to send HTTP requests to the GotAPI Server (i.e. cross-origin requests).

7.2.1.2 GotAPI Authorization Server

The GotAPI Authorization Server provides the following functions:

- Exposure of the GotAPI-2 interface, via which Applications can obtain authorization to make API requests
- User Interface functions as required to locally provide user information and consent for API access by applications
- Acting as a proxy for user consent obtained through host-device external functions (e.g. OAuth servers)
- Database of authorizations and user consent history
- Binding of the GotAPI-2 interface to various specific interface technologies and payload protocols / design patterns.
- Security and privacy protection for requests via the GotAPI-2 interface
- Protection of the GotAPI-2 interface from application identity spoofing, server spoofing, Intrusion and Denial of Service attacks
- Optionally,exposure of the GotAPI-3 interface via which GotAPI authorizations can be provisioned through OMA Device Management or an implementation-specific policy management service

Web applications running on web browsers have to use the XMLHttpRequest to send requests to the GotAPI Authorization Server on the GotAPI-2 interface. The origin of the web application is different from the origin of the GotAPI Authorization Server. Therefore, the GotAPI Authorization Server SHALL support Cross-Origin Resource Sharing [CORS] so that the web browser allows the web application to send HTTP requests to the GotAPI Authorization Server (i.e. cross-origin requests).

7.2.2 Interfaces

7.2.2.1 GotAPI-1

The GotAPI-1 interface enables applications to make API requests and receive responses. This interface is generically specified by GotAPI, as GotAPI-based API specifications will define specific request/response transactions that can be utilized in host devices based upon the available interface technologies, payload protocols, and their applicable design patterns. These options include:

- The interface technologies TLS 1.2, HTTP/1.1, HTTP/2, WebSocket, Server-Sent Events, WebRTC
- The design patterns REST and JSON, such as JSON-RPC
- The Temporary Server Feed (TSF) mechanism for binary data responses and triggering a different protocols, as described below

The GotAPI Server SHALL support HTTP/1.1 as a communication protocol on the GotAPI-1 interface.

Additionally, the GotAPI Server MAY support HTTP/2 [HTTP2], WebSocket [WebSocket] [WebSocketProtocol], Server-Sent Events [SSE], and WebRTC [WebRTC][Rtcweb] as needed.

For example, if the GotAPI Server provides an API for enabling asynchronous notifications such as an event listener (One-way push API), the API can use Server-Sent Events.

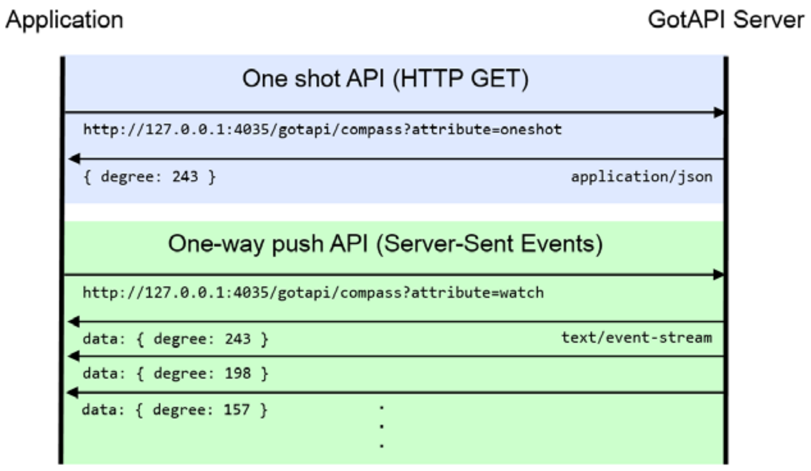


Figure 4: One shot API and One-way push API

If the GotAPI Server provides an API for enabling full-duplex real-time communications such as a chat service, the API can use WebSocket. WebSocket, however, requires another port numbers in addition to 4035/4036 for HTTP/HTTPS to be assigned. This specification does not specify the port numbers for WebSocket (ws: and wss:). Therefore, the GotAPI Server is encouraged to use the TSF mechanism described in the next section for such APIs.

The GotAPI Server SHALL support JSON as a data container format on the GotAPI-1 interface. Additionally, the GotAPI Server MAY support JSON-RPC [JSON-RPC] as needed.

7.2.2.1.1 The Temporary Server Feed (TSF) Mechanism

There are two possible approaches which the GotAPI Server returns API result data to applications:

- Direct response approach:
  - GotAPI Server returns binary data as a response directly
  - This approach is very common and GotAPI-1 already supports it
- Temporary Server Feed (TSF) approach:
  - When an app request something to the GotAPI Server on GotAPI-1, the GotAPI Server creates a temporary URI for the requested data, then return it to the app with additional information
  - Then the app accesses the URI in order to fetch the binary data

The TSF approach has advantages below:

- Flexible API design
  - The TSF mechanism brings flexibility to API design for GotAPI-1
  - APIs can provide additional information relevant to the requested binary data with applications
  - For example, APIs can provide adaptive streaming protocols over HTTP, such as SME + MPEG-DASH
    1. An application requests MPD (Media Presentation Description) data over GotAPI-1
    2. The application fetches fragments of the video data sequentially following the URLs defined in the MPD

- Web developer friendly
    - Lots of existing server-side Web APIs on the Internet provide APIs similar to TSF with developers

The GotAPI Server MAY support the TSF mechanism.

If the GotAPI Server supports the TSF mechanism, the GotAPI Server SHALL support the following steps for data retrieved via a TSF:

- An application sends a request for accessing certain data to the GotAPI Server over the GotAPI-1.
- When the GotAPI Server receives the request, the GotAPI Server creates a non-predictable random URI for the binary data that is requested, and associates the URI with the binary data. The port number of the URI is not necessarily 4035 or 4036. The GotAPI Server MAY decide the port number of the URI appropriately as needed.
- The GotAPI Server sends a response with the URI and additional information (if needed) to the application over the GotAPI-1.
- Receiving the URI, the application accesses the URI in order to get the requested binary data from the GotAPI Server. The GotAPI Server works as a Web server.
- The GotAPI Server discards the URI after the application gets the binary data and/or after certain while for the purpose of security.

Example:

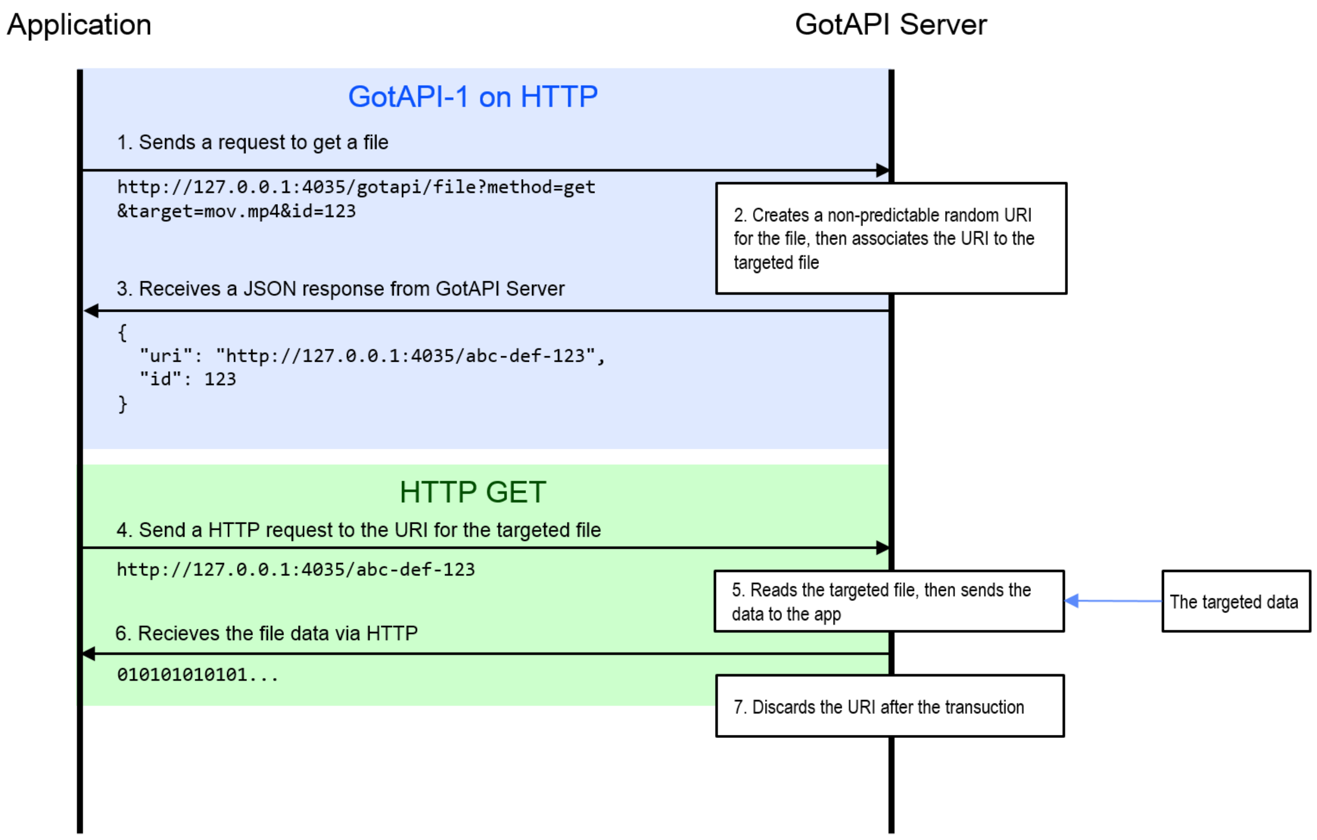

Application
GotAPI Server
GotAPI-1 on HTTP
1. Sends a request to get a file
http://127.0.0.1:4035/gotapi/file?method=get
&target=mov.mp4&id=123
2. Creates a non-predictable random URI for the file, then associates the URI to the targeted file
3. Receives a JSON response from GotAPI Server
{
"uri": "http://127.0.0.1:4035/abc-def-123",
"id": 123
}
HTTP GET
4. Send a HTTP request to the URI for the targeted file
http://127.0.0.1:4035/abc-def-123
5. Reads the targeted file, then sends the data to the app
The targeted data
6. Recieves the file data via HTTP
010101010101...
7. Discards the URI after the transuction

Figure 5: The TSF Procedure

The JSON data and some URIs in the diagram above are just sample code. What protocol/format is used for the GotAPI-1 depends on the GotAPI enabler implementation, and is out of the scope of this specification.GotAPI-2.

Though the example in the figure above shows thea case where HTTP is used, the GotAPI Server MAY use HTTP/2, WebSocket, Server-Sent Events, and WebRTC as needed.

The TSF mechanism can be used for triggering communications using these protocols as well as transferring binary data. The GotAPI Server MAY use the TSF mechanism for other types of data and triggering other protocols.

For example, if the GotAPI Server provides an API for enabling full-duplex real-time communications such as a chat service, the GotAPI Server can use WebSocket instead of HTTP GET.

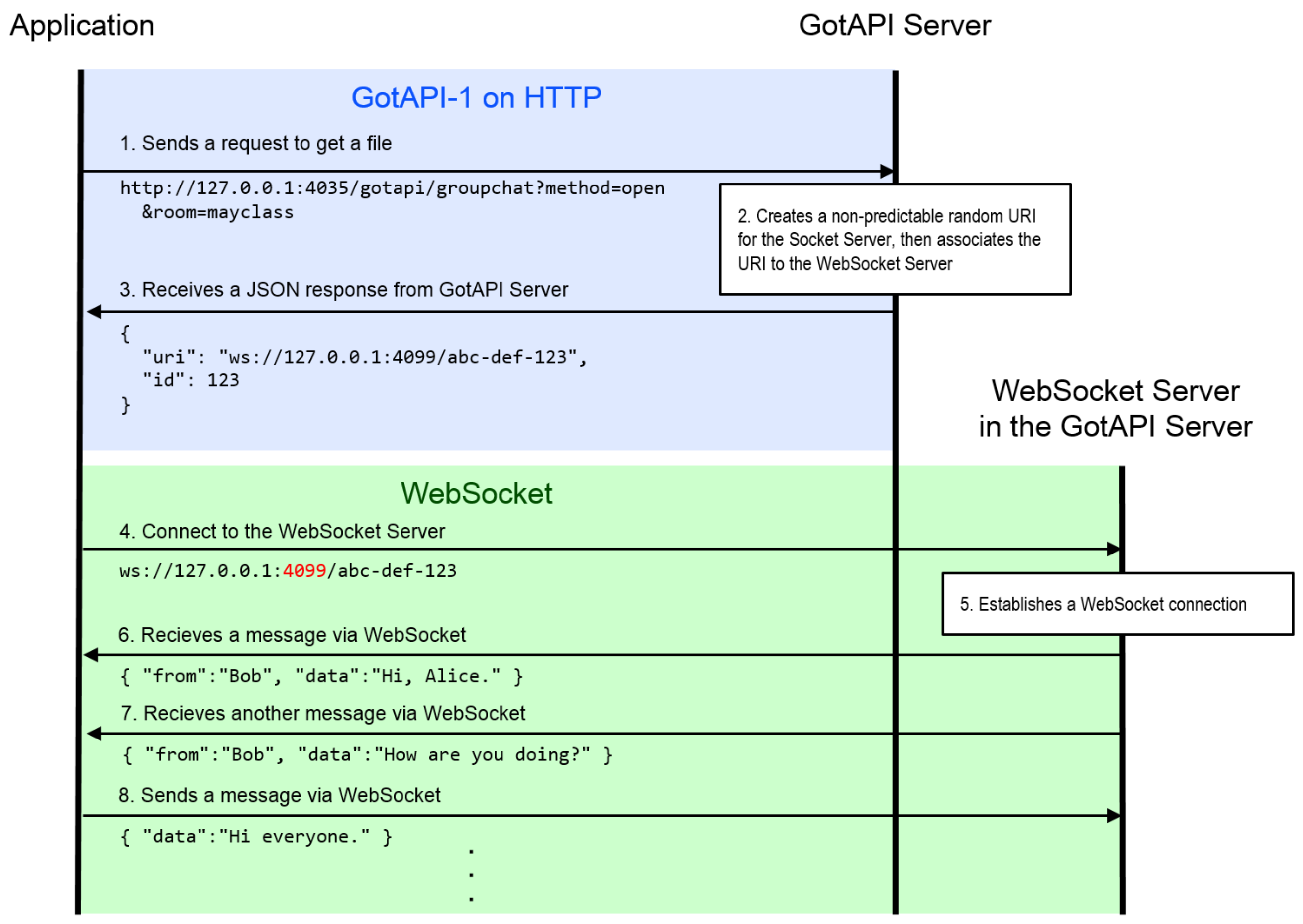

Application
GotAPI Server
GotAPI-1 on HTTP
1. Sends a request to get a file
http://127.0.0.1:4035/gotapi/groupchat?method=open &room=mayclass
2. Creates a non-predictable random URI for the Socket Server, then associates the URI to the WebSocket Server
3. Receives a JSON response from GotAPI Server
{ "uri": "ws://127.0.0.1:4099/abc-def-123", "id": 123 }
WebSocket Server in the GotAPI Server
WebSocket
4. Connect to the WebSocket Server
ws://127.0.0.1:4099/abc-def-123
5. Establishes a WebSocket connection
6. Recieves a message via WebSocket
{ "from":"Bob", "data":"Hi, Alice." }
7. Recieves another message via WebSocket
{ "from":"Bob", "data":"How are you doing?" }
8. Sends a message via WebSocket
{ "data":"Hi everyone." }

Figure 6: WebSocket used for TSF

7.2.2.2 GotAPI-2

The GotAPI-2 interface enables applications to obtain authorization for access to GotAPI-based APIs. This interface is fully specified by GotAPI, being a common (though optionally used) support function for all GotAPI-based APIs. GotAPI-2 supports bindings and request/response transactions that can be utilized in host devices based upon the available interface technologies. These options include the interface technologies TLS 1.2, HTTP/1.1, HTTP/2, and URI scheme handling.

The GotAPI-2 interface is based upon the concepts of OAuth, though with different semantics as necessary for adaptation to the available interface technologies.

In this specification, an "origin" is an identifier of an application, which is globally unique.

- If the application is a web application, the origin is literally an origin as defined by RFC6454, which is a concatenating string that is composed of the scheme, the fully qualified host name, and the TCP port number. For example, if the URL of a web application is "https://app.example.com:443/index.html", the origin is "https://app.example.com:443".If the application is an OS-specific native application, the origin is an application identifier managed by the underlying OS, such as a package name. For example, if the OS is Android, the origin could be "com.example.app".

- If the application is a Hybrid Native/Web App, it is treated as an OS-specific native application by the underlying OS. Therefore the origin is an application identifier managed by the underlying OS, such as a package name.

The origin is embedded in the HTTP request header by the application itself or by the web browser automatically as described in the section “8.2 GotAPI Authorization Server”.

The authenticity of the origin of the application is crucial for the entire operation of the GotAPI system.

The GotAPI Auth Server SHALL be able to extract the origin from the HTTP request header appropriately.

A) Basic GotAPI-2 Interface procedures:

The GotAPI-2 Interface must be able to run the steps as follows:

1) Authorization of application

The application sends an authorization request to the GotAPI Auth Server with the origin.

The GotAPI Auth Server MAY support a white-list of origins that have been pre-authenticated by the GotAPI service provider. If the GotAPI Auth Server supports such a white-list and the origin is listed in the white-list, the GotAPI Auth Server MAY determine if the application is acceptable or not.

If the GotAPI Auth Server accepts the application, the GotAPI Auth Server SHALL create a series of random digits, called as “grant”, that is long enough not to be predicted, and, then, SHALL send the response to the application with the grant.

2) Issuance of an access token

When the application receives the grant from the GotAPI Auth Server, it immediately sends an access token request to the GotAPI Auth Server with the origin, the grant, and the scope which is a list of functions the application wants to use. When the GotAPI Auth Server receives the request, the GotAPI Auth Server SHALL ask the user if the application may use the requested scope (the list of functions). In practice, the GotAPI Auth Server shows an OS-specific dialog box to the user.

The GotAPI Auth Server SHALL show the items in the dialog box as below:

- o The information of the application (e.g. the origin of the web application or the package name of the OS-specific native application)
- o The list of the functions which the application want to use (scope)
- o A button which the user presses if the user accepts the application
- o A button which the user presses if the user declines the application

If the user declines the request, the GotAPI Auth Server SHALL NOT allow the application to proceed any further beyond the point.

When the GotAPI Auth Server accepts the request, the GotAPI Auth Server SHALL create an access token for the application, which is a series of random digits that is long enough not to be predicted. Then the GotAPI Auth Server SHALL send the response with the access token to the application.

The application subsequently sends API requests with the access token on the GotAPI-1 Interface.

B) Security enforced GotAPI-2 Interface procedures:

In addition to the basic GotAPI-2 Interface procedures as described above, there are two security concerns for the GotAPI-2 Interface that needs to be addressed.

- GotAPI Auth Server spoofing, and
- Application’s origin spoofing

1) GotAPI Auth Server spoofing:

   The GotAPI Auth Server spoofing is an attack where a bogus GotAPI Auth Server takes over the genuine GotAPI Auth Server and pretends as if it was the GotAPI Auth Server. When the application sends a request to the GotAPI Server for the first time, there may even be a case where the GotAPI Server has already been taken over by the attacker.

   Since GotAPI Auth Server spoofing is an easy-to-do attack, applications need to be able to verify if the GotAPI Auth Server is genuine or not. The HMAC server authentication resolves this spoofing attack.

   The GotAPI Auth Server SHOULD be able to support the HMAC server authentication using the Trusted Channel with the Application ID as described in the section 7.3.3.2

2) Application’s origin spoofing:

   The application’s origin spoofing is an attack where a malicious application acts as other application by sending a fake identity (a.k.a. origin) to the GotAPI Auth Server. The GotAPI Auth Server needs to be able to verify if the identity that it has received from the application is authentic or not.

   When an application sends a request for authentication over the GotAPI-2 Interface, the origin of the application is included in the HTTP request header.

   If the application is a web application running on a web browser, the application cannot override the Origin header in the HTTP request header [W3C XHR]. Therefore, the origin coming from a web application is trustable.

   On the other hand, if the application is an OS-specific native application, the application may send a fake identity in the HTTP request header. Besides, a malicious native application may set the Origin header to be a fake origin pretending a web application running on a web browser.

   When an application sends a request for authentication, the GotAPI Auth Server SHOULD be able to verify the origin coming from the application to determine if the origin is authentic or not.

   The verification needs to depend on the features of the underlying OS domain on which the GotAPI Auth Server is running.

Here is an example of such a verification mechanism if the underlying OS is Android.

An example of verifying origin spoofing on Android:

- Android supports the netstat command by default. But it does not provide the process ID of the native application establishing the HTTP connection.
- Using Android NDK, however, the full-featured netstat can be built and packaged within an Android native application (i.e., GotAPI Server application).
- The GotAPI Auth Server embedding the full-featured netstat identifies the process ID of the native application from the result of the full-featured netstat, and it can get the package name and application name from the process ID using the Android API.
- The GotAPI Auth Server uses a white-list of application names of legitimate browsers that have been verified to be compliant to the origin header’s not-over-ridden requirements [W3C XHR]. The white-list enables the GotAPI Auth Server to distinguish the case of (i) a web application declaring an origin from (ii) a malicious native application fakes origin header to pretend a web application.

The Table below shows all the cases where an attacking native application, com.attacker.app, declares various origins in the HTTP header and what the netstat can find. As shown in the table, the faked origins can be completely found by the netstat.

	Real package name	Declared origin in HTTP header	True/Fake	What netstat finds	Results	Notes
1	com.attacker.app	http://example.com	Fake	com.attacker.app	Fake found	*1
2	com.attacker.app	com.example.app	Fake	com.attacker.app	Fake found	*2
3	com.attacker.app	com.attacker.app	True	com.attacker.app	True confirmed	*3

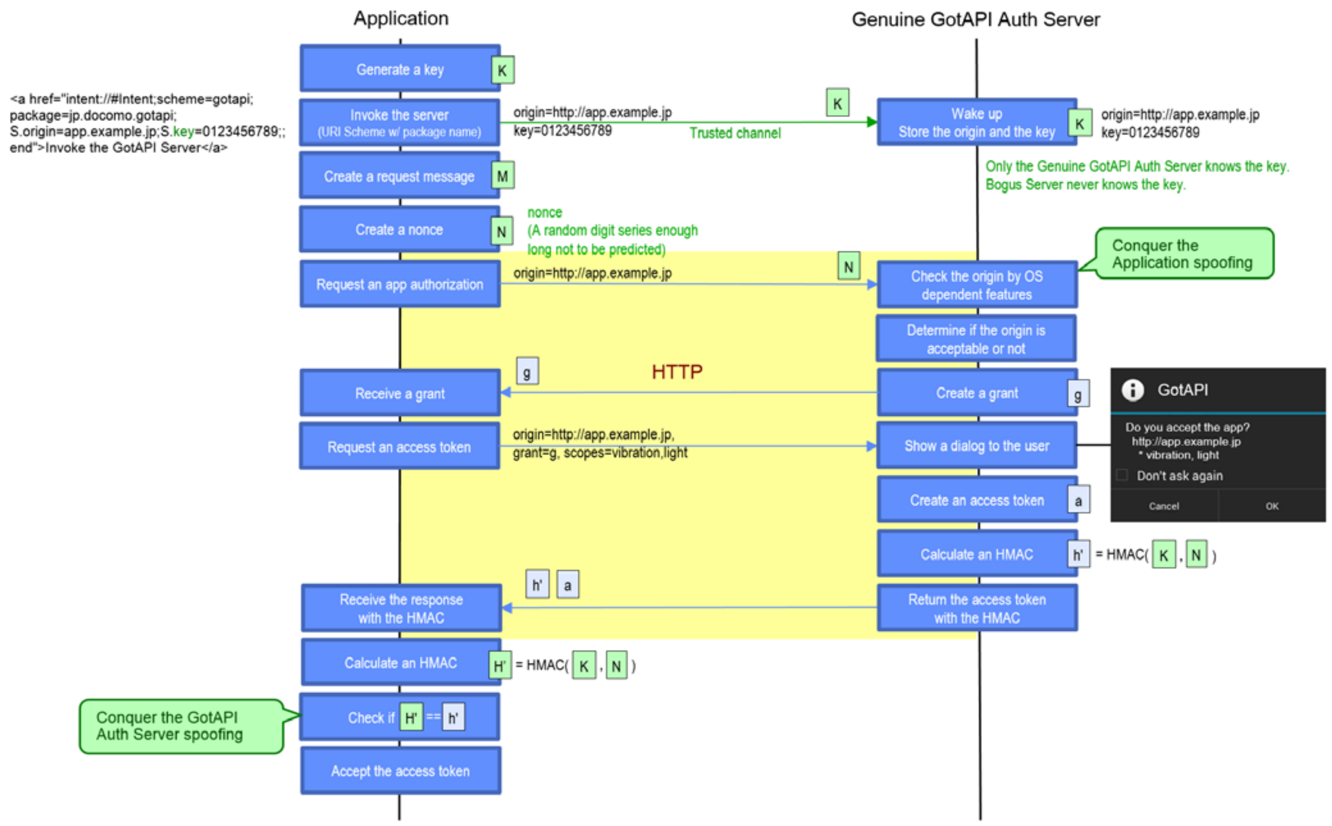
Table 6: Cases of origins declaration by attacking native application and what the netstat can find

*1: If the declaration of the origin was from a web application running in a legitimate browser, the netstat should have found the name of the browser that is registered in the white-list of legitimate browsers, instead of the package name of the attacking native application, com.attacker.app.

*2: The package name that the netstat has found is different from the origin that the application is declaring in the HTTP header.

*3: GotAPI Auth Server confirms what the native application is declaring in the HTTP header is authentic.

The figure below shows the procedure of application authorization on the GotAPI-2 Interface including the security enforced measures against the threats, GotAPI Auth Server spoofing, and application origin spoofing.


```
<a href="intent://#Intent;scheme=gotapi;
package=jp.docomo.gotapi;
S.origin=app.example.jp;S.key=0123456789;;
end">Invoke the GotAPI Server</a>
```

Do you accept the app?
http://app.example.jp
* vibration, light

Application
Genuine GotAPI Auth Server
Generate a key
K
Invoke the server
(URI Scheme w/ package name)
origin=http://app.example.jp
key=0123456789
Trusted channel
K
Wake up
Store the origin and the key
K
origin=http://app.example.jp
key=0123456789
Only the Genuine GotAPI Auth Server knows the key.
Bogus Server never knows the key.
Create a request message
M
Create a nonce
N
nonce
(A random digit series enough
long not to be predicted)
Request an app authorization
origin=http://app.example.jp
N
Check the origin by OS dependent features
Conquer the Application spoofing
Determine if the origin is acceptable or not
HTTP
Receive a grant
g
Create a grant
g
GotAPI
Don't ask again
Cancel
OK
Request an access token
origin=http://app.example.jp,
grant=g, scopes=vibration,light
Show a dialog to the user
Create an access token
a
Calculate an HMAC
h' = HMAC(K , N)
h'
a
Receive the response with the HMAC
Return the access token with the HMAC
Calculate an HMAC
H' = HMAC(K , N)
Conquer the GotAPI Auth Server spoofing
Check if H' == h'
Accept the access token

Figure 7: Procedure of security enforced application authorization on the GotAPI-2 Interface

In order to prevent GotAPI Auth Server spoofing and application's origin spoofing, the GotAPI Auth Server SHOULD be able to support the security enforced GotAPI-2 Interface procedures as described in Figure 7.

7.2.2.3 GotAPI-3

The GotAPI-3 interface enables the remote provisioning of API access authorizations through a policy management function, which may include one or more of:

- OMA Device Management, using a Managed Object (MO) defined by the GotAPI enabler [OMA DM]
- An implementation-specific policy management service

Note: The GotAPI 1.0 does not specify the Managed Object (MO) for the GotAPI enabler.

7.2.2.4 GotAPI-4

The GotAPI-4 interface enables Extension Plug-Ins for external devices and internal enablers through which they communicate with the GotAPI Server. Note that host-device-internal enablers/applications may also be connected to GotAPI servers directly in implementation specific ways without using the GotAPI-4 interface and Extension Plug-Ins.

The Extension Plug-Ins are independent applications. They are the mediators between the GotAPI Server, and external devices and internal enablers/applications. Typically, there are expected to be multiple Extension Plug-In applications installed on a device by the user or preinstalled on the device. An Extension Plug-In application may be developed:

- For a group of devices, e.g., a series of devices from a company, or a single device or an enabler,
- By a developer that is different from the provider of GotAPI or applications that use the devices or enablers through the Extension Plug-In.

The GotAPI-4 interface provides the following functions with respect to Extension Plug-Ins:

1. **Plug-In Discovery**: GotAPI-4 Plug-In Discovery enables the GotAPI Server to discover the targeted Extension Plug-In which an application wants to access and communicate with.
2. **Service Discovery**: GotAPI-4 Service Discovery enables the GotAPI Server to find all the services provided by an Extension Plug-In. In this context, the "service" means an external device or a function provided by an internal enabler through an Extension Plug-In. The Service Discovery provides not only the list of services but also the availability of each service at the time.
3. **Approval**: GotAPI-4 Approval is the function to ensure security, especially to protect users' data and privacy from unwanted exploits, so that the users can safely use the application with external devices and enablers that are connected via Extension Plug-Ins.
4. **Data Forwarding**: GotAPI-4 Data Forwarding is the function that enables an application to communicate with the targeted Extension Plug-In through the GotAPI Server. Data Forwarding takes place after Plug-In Discovery (optional) and Approval processes have been successfully completed. GotAPI-4 Data Forwarding uses the "pass-through" mechanism, so that the application can access and communicate with the APIs that (i) are implemented in the Extension Plug-In and (ii) expose the features of the external devices or internal enablers.

   Note that the APIs to be implemented in Extension Plug-Ins that expose features of external devices and internal enablers are out of the scope of this specification.

7.2.2.4.1 Plug-In Discovery

When applications use Extension Plug-Ins through the GotAPI Server, the GotAPI Server has to know what Extension Plug-Ins are installed. This section describes how the GotAPI Server discovers the installed Extension Plug-Ins in the local devices.

Therefore:

- The GotAPI SHALL support the Plug-In Discovery if GotAPI-4 is supported.
- The GotAPI Server SHALL support a mechanism to know what Plug-Ins are installed in the local device.

To discover the installed Extension Plug-Ins, the GotAPI Server has to use OS-specific mechanisms and functions. Regarding Android, see the section "Appendix F. Finding the installed Plug-Ins for Android".

Note: how the GotAPI Server discovers Extension Plug-Ins is out of the scope of this specification.

How to ensure that Extension Plug-Ins on an OS from different vendors are able to be discovered by a GotAPI Server implementation is the responsibility of the provider of the GotAPI Server implementation.

7.2.2.4.2 Service Discovery

In many cases, one Extension Plug-In is associated with one external device. Some Extension Plug-Ins are associated with multiple external devices. Some Extension Plug-Ins provide functions that work in the local device but is not associated with any external device (a.k.a. an internal enabler itself). Such external devices or functions are called "services".

When an application wants to use a service, it needs to specify the identifier of the service rather than the Extension Plug-In that is supporting the service. Applications basically don't care about what Extension Plug-In is associated with the service.

The Service Discovery enables applications to find services. Some applications, however, may be pre-programmed with specific services. Others may want to find what services are available.

Therefore:

- The use of the Service Discovery is OPTIONAL for applications wishing to use services.
- The GotAPI SHALL support the Service Discovery if the GotAPI-4 is supported.
- The GotAPI-1 SHALL support the Requests and Responses for the Service Discovery if GotAPI-4 is supported. This is to ensure consistent interface for application developers.
- GotAPI-4 SHALL support the protocol (the data container format) between the GotAPI Server and the Extension Plug-Ins. This is to ensure consistent interface for Extension Plug-In developers.

<u>Example of Service Discovery</u>

The following example is based on an implementation on Android.

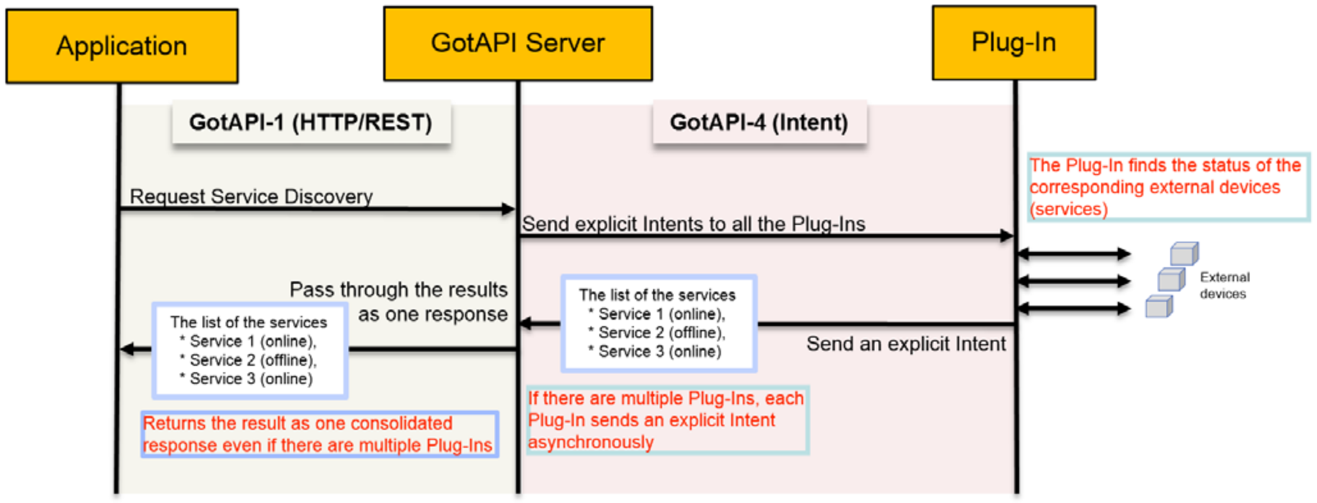

Application
GotAPI Server
Plug-In
GotAPI-1 (HTTP/REST)
GotAPI-4 (Intent)
The Plug-In finds the status of the corresponding external devices (services)
Request Service Discovery
Send explicit Intents to all the Plug-Ins
External devices
Pass through the results as one response
The list of the services * Service 1 (online), * Service 2 (offline), * Service 3 (online)
The list of the services * Service 1 (online), * Service 2 (offline), * Service 3 (online)
Send an explicit Intent
Returns the result as one consolidated response even if there are multiple Plug-Ins
If there are multiple Plug-Ins, each Plug-In sends an explicit Intent asynchronously

Figure 8: The procedure of the Service Discovery

Terminology:

- **Explicit Intents** are Intents with a specific application identifier, enabling the sending application to specify the exact receiving application to be run.

Description of operation:

General operation:

(1) When the GotAPI Server has received a Service Discovery request from an application over the GotAPI-1 interface, the GotAPI Server sends a Service Discovery Command to each of the installed Extension Plug-Ins using the protocol (the data container format) of GotAPI-4 over an Explicit Intent. Note: The Plug-In Discovery has already found Extension Plug-Ins that are installed on the device.

(2) When an Extension Plug-In receives a Service Discovery Command from the GotAPI Server, the Extension Plug-In checks the availability of the service that is requested by the Service Discovery. When the Extension Plug-In completes checking the availability of the service, the Extension Plug-In sends a response to the GotAPI Server over an Explicit Intent.

(3) When the GotAPI Server has received responses from all of the Extension Plug-Ins, the GotAPI Server returns the result to the application as one response.

Multiple plug-ins and asynchronous responses:

Since there can be multiple Extension Plug-Ins installed on the device, each response is sent to the GotAPI Server asynchronously.

White List:

When an Extension Plug-In sends a response, it uses an Explicit Intent to the pre-defined GotAPI Server that is listed in the White List. The White List is provided in each Extension Plug-In by the provider of the Extension Plug-In. The White List enables Extension Plug-Ins to send responses only to the GotAPI Server applications that are listed in the list and prevents Extension Plug-Ins from sending responses to unknown GotAPI Servers. This is to disable spoofed GotAPI Servers to use Extension Plug-Ins.

Consolidated response from GotAPI Server:

After consolidating the responses that are sent from multiple Extension Plug-Ins asynchronously, the GotAPI Server sends a response to the originating application with the information that are received from the Extension Plug-Ins.

Stateless:

The GotAPI Server itself does not keep the status of the services that are discovered by the Service Discovery Command, keeping GotAPI Server stateless in terms of services. It is the sole responsibility of the applications that have received the service status information how to keep or use it.

Note: As described in the Plug-In Discovery section, the GotAPI Server must keep the up-to-date status of the Extension Plug-Ins that are installed on the device.

7.2.2.4.3 Approval

After an application is registered by the GotAPI Authentication Server through user permission, the application is eligible for accessing Services provided by Plug-Ins. To ensure protecting user’s data and privacy, however, before the user is able to access the Services via the Plug-In using the application, the user shall be able to authorize the application to access the Plug-In and the Service. To enable this requirement:

- The application SHOULD be authorized to access the Service via the Plug-In by the user.
- The GotAPI-1 SHALL support the Requests and Responses for authorization for an application to access the Service via the Plug-In via user authorization. This is to ensure consistent interface for application developers.
- The GotAPI-1 SHALL support the Requests and Responses for authorization for an application to access the Service via the Plug-In via user authorization if GotAPI-4 is supported

The authorization mechanism for Services and Plug-Ins over the GotAPI-4, based on an implementation on Android consists of the following steps:

Plug-In Service API Access Request: Typically after the application performing Service Discovery to get the information of the available Services via installed Plug-Ins, the application requests a Plug-In Service API Access Request to the GotAPI Server over the GotAPI-1 interface.

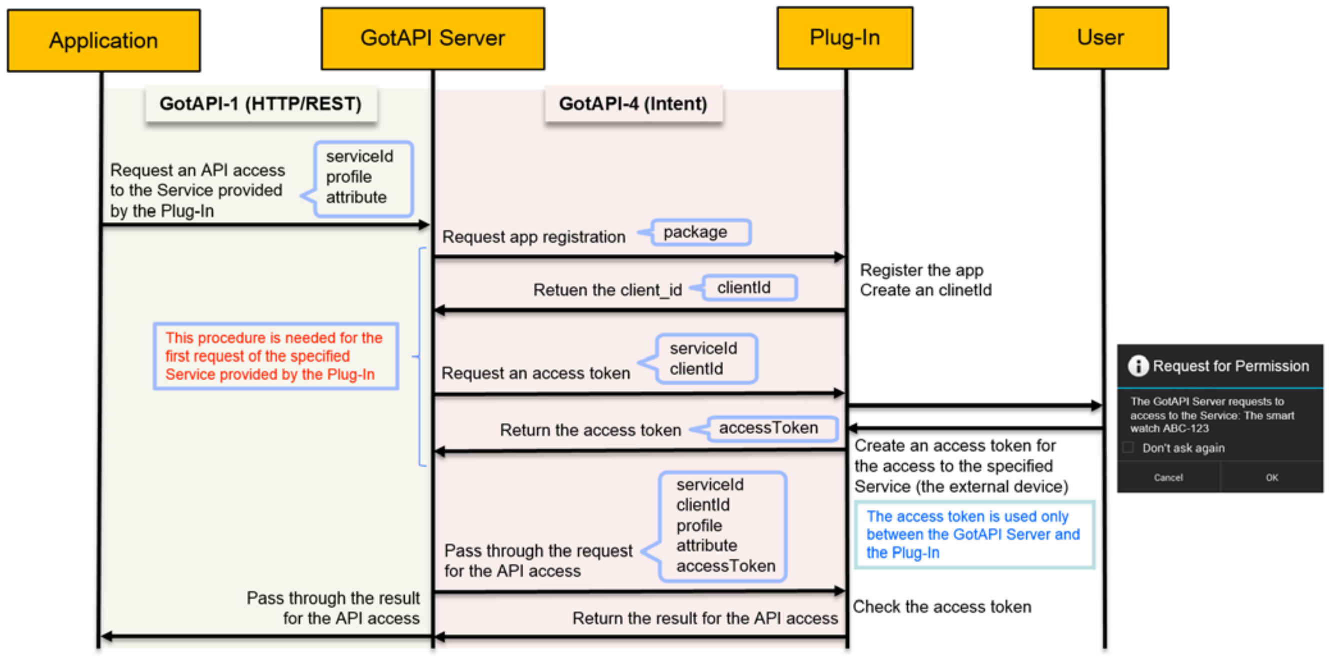
Application Registration to Plug-Ins: If the request is made for the first time, the GotAPI Server requests Application Registration to the targeted Plug-In over the GotAPI-4 with the origin of the application. Then the Plug-In registers the application and creates a clientId which is an identifier of the application managed by the Plug-In.

ClientId and White List: The Plug-In returns the clientId to the GotAPI Server over the GotAPI-4 interface, using an Explicit Intent. The Explicit Intent with the White List in the Plug-In disables providing a clientId to a spoofed GotAPI Server (the same mechanism as in the Plug-In Discovery).

Access Token and User Authorization: The GotAPI Server requests an access token with the clientId and the serviceId provided by the application over the GotAPI-1 Interface. Upon receiving the access token request, the Plug-In pops up a dialog box to the user, which prompts the user to select the permission for the Service provided by the Plug-In. If the user permits the access request, the Plug-In creates an access token and returns it to the GotAPI Server. Note that an access token is used only between the GotAPI Server and Plug-Ins over the GotAPI-4 interface.

Accessing API using access token: When the GotAPI Server receives the access token from the Plug-In, the GotAPI Server passes the Plug-In Service API Access Request from the application to the Plug-In over GotAPI-4 with the access token. When the GotAPI Server receives the response form the Plug-In, the GotAPI Server passes the response from the Plug-In to the application over the GotAPI-1 interface.

Reusable access token and life time: Once the GotAPI Server receives an access token for an application, the GotAPI Server doesn't need to request another Application Registration or an access token as long as the requested Service is the same. The GotAPI Server can continue using the same access token for a while as long as the Plug-In accepts the access token. An access tokens is given a life time, so that the same access token can be used before the life time is expires. After the life time is expired, the GotAPI Server must request another access token using the same procedure.


Application
GotAPI Server
Plug-In
User
GotAPI-1 (HTTP/REST)
GotAPI-4 (Intent)
Request an API access to the Service provided by the Plug-In
serviceId profile attribute
Request app registration
package
Register the app Create an clinetId
Retuen the client_id
clientId
This procedure is needed for the first request of the specified Service provided by the Plug-In
Request an access token
serviceId clientId
Request for Permission
Don't ask again
Cancel
OK
Return the access token
accessToken
Create an access token for the access to the specified Service (the external device)
The access token is used only between the GotAPI Server and the Plug-In
Pass through the request for the API access
serviceId clientId profile attribute accessToken
Check the access token
Pass through the result for the API access
Return the result for the API access

The GotAPI Server requests to access to the Service: The smart watch ABC-123

Figure 9: Service and Plug-In Approval

The parameters (e.g. profile, attribute, etc.) in the diagram above are simplified examples. See the section "8.3 GotAPI Server" for the exact definition of the data set.

7.2.2.4.4 Data Forwarding

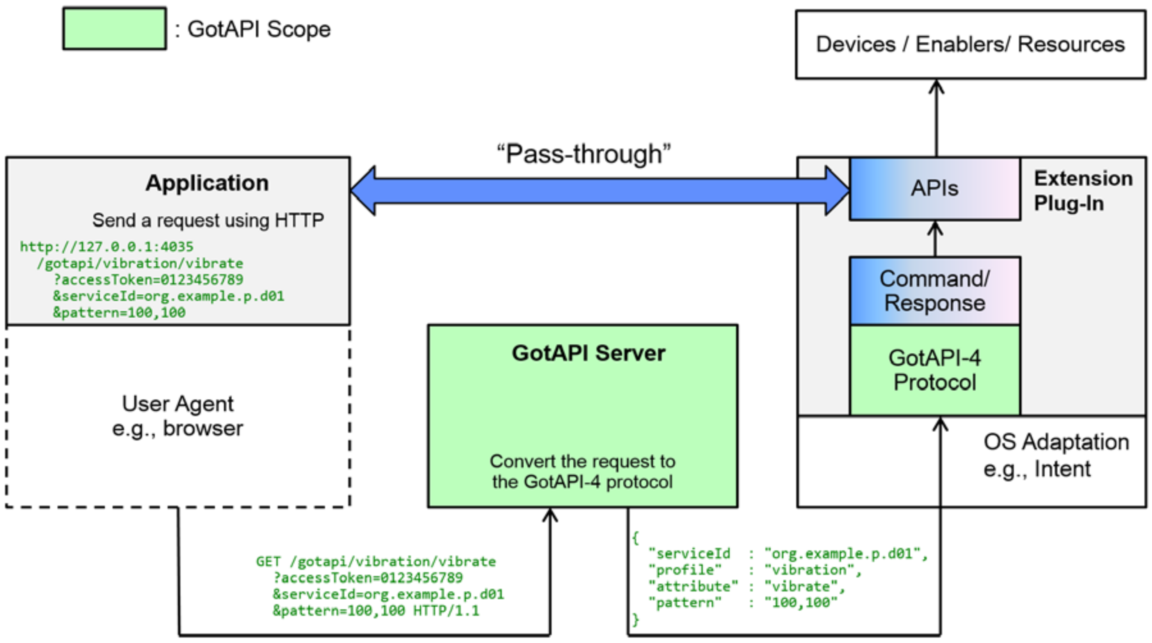
Once a connection between the GotAPI Server and the targeted Extension Plug-In is established (i.e., GotAPI-4 Plug-In Discovery (optional) and GotAPI-4 Approval have been successfully completed), the application can communicate with the targeted Extension Plug-In. The data transferred between the application and the Extension Plug-In pass-through the GotAPI Server.

The GotAPI-4 Data Forwarding defines the protocol (the data container format) between the GotAPI Server and the Extension Plug-Ins that are connected with external devices or internal enablers.

Example:

The following description and Figure-7 show how the pass-through mechanism of the Data Forwarding works:

An application sends a request to the GotAPI Server using an HTTP connection with some parameters in accordance with the GotAPI-1. The GotAPI Server converts the request to the data format (protocol) in accordance with the GotAPI-4 Data Forwarding specification. Then the GotAPI Server conveys the converted data to the targeted Extension Plug-in using the OS adaptation, such as Intent for Android. Finally, the Extension Plug-In invokes the APIs with the received and re-converted data. The APIs are implemented in the Extension Plug-In. This mechanism allows requests and responses between applications and external or internal entities to be passed-through to the APIs.



```
http://127.0.0.1:4035
  /gotapi/vibration/vibrate
    ?accessToken=0123456789
    &serviceId=org.example.p.d01
    &pattern=100,100
```

```
GET /gotapi/vibration/vibrate
  ?accessToken=0123456789
  &serviceId=org.example.p.d01
  &pattern=100,100 HTTP/1.1
```

```
{
  "serviceId  : "org.example.p.d01",
  "profile"   : "vibration",
  "attribute" : "vibrate",
  "pattern"   : "100,100"
}
```

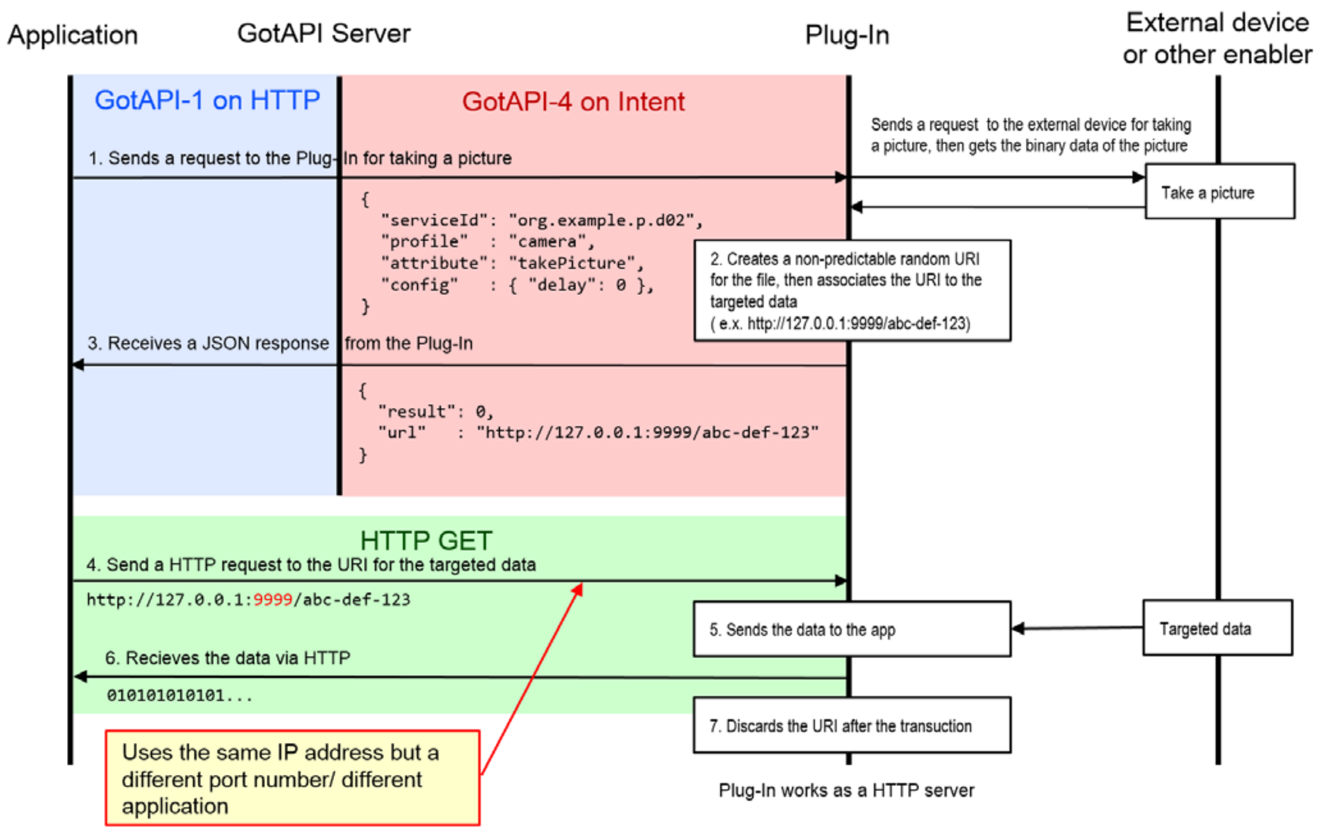
Figure 10: Pass-through mechanism of Data Forwarding

The JSON data and some URIs in the diagram above are just samples and simplified. See the section "8.1 GotAPI Server" for the exact definition of the data set.

In order to get data (binary files, streaming, event notifications, etc.) using HTTP or a different protocol, e.g., WebSocket, Server-Sent Events , WebRTC, from the Plug-Ins, the Temporary Server Feed (TSF) mechanism may be used:

- An application sends a request to the GotAPI Server over the GotAPI-1 interface.
- The GotAPI Server passes the request to the Plug-In over the GotAPI-4 interface.
- When the Plug-In receives the request, the Plug-In creates a non-predictable random URI for the data that is requested, and associates the URI with the data.

- The Plug-In sends a response with the URI and additional information (if needed) to the GotAPI Server over the GotAPI-4 interface, and the GotAPI Server passes the response to the application over the GotAPI-1 interface.
- Receiving the URI, the application accesses the URI using the protocol that is indicated by the Plug-In in order to get the requested data from the Plug-In directly. The Plug-In works as a Web server.
- The Plug-In discards the URI after the application gets the data or a preset life time expires for the purpose of security.
- The URI may use the same IP address as the GotAPI Server but with a different port number. This enables the Plug-In to be a separate application than the GotAPI Server application.


Application
GotAPI Server
Plug-In
External device or other enabler
GotAPI-1 on HTTP
GotAPI-4 on Intent
1. Sends a request to the Plug-In for taking a picture
Sends a request to the external device for taking a picture, then gets the binary data of the picture
Take a picture
2. Creates a non-predictable random URI for the file, then associates the URI to the targeted data (e.x. http://127.0.0.1:9999/abc-def-123)
3. Receives a JSON response from the Plug-In
HTTP GET
4. Send a HTTP request to the URI for the targeted data
http://127.0.0.1:9999/abc-def-123
5. Sends the data to the app
Targeted data
6. Recieves the data via HTTP
010101010101...
7. Discards the URI after the transuction
Uses the same IP address but a different port number/ different application
Plug-In works as a HTTP server

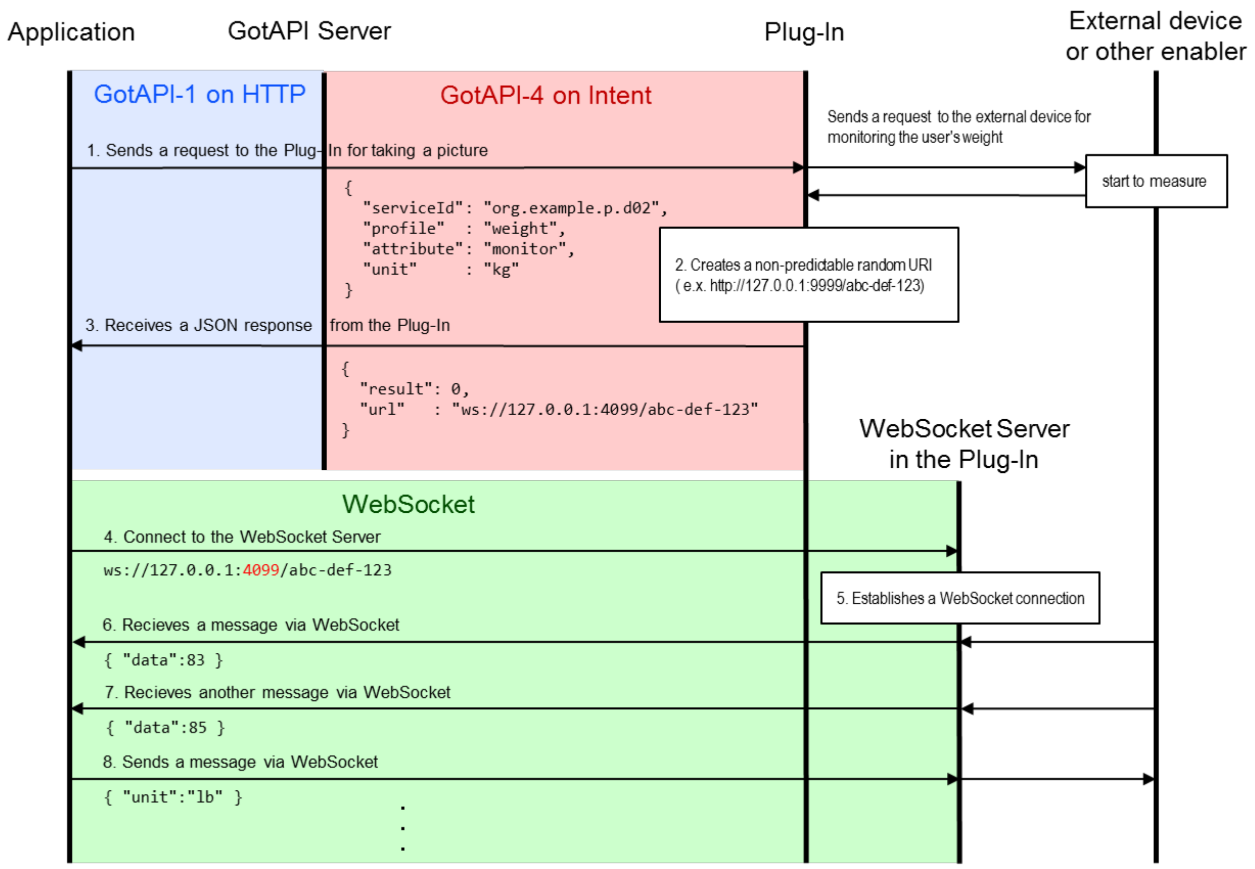
```
{
  "serviceId": "org.example.p.d02",
  "profile"  : "camera",
  "attribute": "takePicture",
  "config"   : { "delay": 0 },
}
```

```
{
  "result": 0,
  "url"   : "http://127.0.0.1:9999/abc-def-123"
}
```

Figure 11: The TSF mechanism for GotAPI-4

The JSON in the diagram above are simplified examples. See the section “8.3 GotAPI Server” for the exact definition of the data set.

The example in the figure below shows how the Plug-in can use WebSocket for the TSF mechanism. WebSocket allows the Plug-in to push asynchronous real-time event notifications fired by the external device to the application, and to accept commands from the application.


Application
GotAPI Server
Plug-In
External device or other enabler
GotAPI-1 on HTTP
GotAPI-4 on Intent
1. Sends a request to the Plug-In for taking a picture
Sends a request to the external device for monitoring the user's weight
start to measure
2. Creates a non-predictable random URI (e.x. http://127.0.0.1:9999/abc-def-123)
3. Receives a JSON response from the Plug-In
WebSocket Server in the Plug-In
WebSocket
4. Connect to the WebSocket Server
5. Establishes a WebSocket connection
6. Recieves a message via WebSocket
7. Recieves another message via WebSocket
8. Sends a message via WebSocket

```
{
  "serviceId": "org.example.p.d02",
  "profile"  : "weight",
  "attribute": "monitor",
  "unit"     : "kg"
}
```

```
{
  "result": 0,
  "url"   : "ws://127.0.0.1:4099/abc-def-123"
}
```

```
ws://127.0.0.1:4099/abc-def-123
{ "data":83 }
{ "data":85 }
{ "unit":"lb" }
```

Figure 12: WebSocket used for the TSF

7.3 Security Considerations

7.3.1 Authorization

GotAPI may be used to expose APIs which provide access to sensitive device functions or data. This presents a risk of exposure of device functions and data without informing the user or obtaining consent, or from rogue applications that seek to fraudulently access GotAPI based APIs. Note that whether a specific GotAPI based API is considered to expose sensitive functions or data must be clarified by the specification for the GotAPI based API. The following requirements are intended to address the general risk of unauthorized access to GotAPI based APIs:

- If sensitive functions or data are exposed by a GotAPI based API, the GotAPI Server SHALL verify API access permission for the specific application, before providing API service.
- GotAPI Servers SHALL provide at least one means of managing API access permissions, including one or more of:
  - Permissions that are remotely managed via OMA Device Management [OMA DM] using the GotAPI Permissions managed object [GotAPI MO]
  - Permissions that are remotely managed via implementation-specific means
  - Permissions that are managed by the user through GotAPI Authorization Server user interfaces

GotAPI Servers SHALL support access permissions granted on a variety of bases, including:

- Free access, i.e. no permission required
- Global access, meaning that once access is granted to any device or application, all further requests are allowed
- Device-specific access, meaning that access is authorized on a per-device basis, including local device access, subnet-based access (e.g. to enable access by any device in a private network), or specific devices by source IP address
- Application-specific access, meaning that access is authorized for each application which can be reliably identified using:
    - For web-based applications, the HTTP Origin header
    - For native applications and Hybrid Native/Web Apps, an application identifier as specific to the native platform

7.3.2 Confidentialty and Integrity

GotAPI is intended to support securely-exposed APIs to help ensure confidentiality and integrity of API operations when needed. When protected by transport layer security (e.g. TLS 1.2), GotAPI based APIs require consideration of the following potential issues:

- GotAPI Servers are unlikely to be provisionable with server certificates that can be validated by clients. For APIs exposed over TLS, this will likely result at least in certificate warnings at the client, and possibly failure of the client to connect to the GotAPI Server.

Other means of ensuring confidentiality and integrity of API operations may also be supported, such as API request/response payload encryption based upon pre-shared or dynamically established encryption keys.

7.3.2.1 Confidentiality

Depending on the underlying platforms or UAs, there are cases where confidentiality protection is already granted. For such cases, there is no need to support this requirement. If, however, the underlying mechanisms do not support data confidentiality protection, this requirement should be supported. To support message confidentiality (1) transport encryption, e.g., TLS/SSL, or (2) end-to-end encryption are available. But transport encryption e.g., TLS/SSL, has some issues applying to the GotAPI environments. For end-to-end encryption, the encryption keys may be distributed through the Trusted Channel as defined in section 7.3.3.3.

7.3.2.2 Integrity

Depending on the underlying platforms or UAs, there are cases where integrity protection is already granted. For such cases, there is no need to support integrity requirement by the GotAPI enabler. If, however, the underlying mechanisms do not support integrity protection, this requirement should be supported.

In case where message integrity check is needed depending on the OS, the UA or the environment, HMAC message authentication can be used. The HMAC server authentication that is defined in 7.3.3 is optimized only for server authentication, not including message authentication. In order to apply HMAC for message integrity check as well as server authentication, an HMAC needs (i) to incorporate both the message and the nonce, and (ii) to be generated, sent and verified in both ways symmetrically.

7.3.3 Immunity from Attack

7.3.3.1 Traffic based attack

Since it exposes a service on host devices, the GotAPI enabler by nature consumes device resources in handling service requests. This presents a risk if the GotAPI Server and the GotAPI Auth Server are not adequately protected from rogue applications that may launch intrusion or denial-of-service (DOS) attacks on the host device, which may cause GotAPI host device instability, unusability, or excessive resource consumption (e.g. battery). Such attacks can involve excessive API requests or malformed API requests. The following requirements are intended to address these risks:

- GotAPI Servers and GotAPI Auth Servers SHALL limit API request volume to an unspecified maximum rate, in order to limit exposure to DOS attacks. GotAPI Servers and GotAPI Auth Servers SHALL temporarily disable API permissions for applications that are suspected of excessive API requests.
- GotAPI Servers and GotAPI Auth Servers SHALL ensure the validity of API requests prior to processing them. GotAPI Servers and GotAPI Auth Servers SHALL temporarily disable API permissions for applications that are suspected of sending maliciously malformed API requests.

GotAPI Servers and GotAPI Auth Servers SHALL provide a means for users to be informed of applications that have been suspended from API access due to suspected security violations, and a means to re-authorize API access for those applications.

7.3.3.2 Server spoofing attack

If the underling operating system allows for an application to kill other applications that are running in the background , it is possible for a bogus application to spoof the genuine GotAPI Server or GotAPI Auth Server, and provide fake or harmful information to the application. An attack can be made by 1) terminating the running GotAPI Server and 2) taking over the port that the Server been listening to. If this attack is made, the application that is communicating only through the port, has no way to know that the Server is spoofed. This type of attack is called the server spoofing attack. To prevent this attack, applications must be able to authenticate the Server that they are communicating with.

There are two approaches possible to enable such authentication of the genuine Server.

(1) Approaches not to embed any credentials in the GotAPI application, and

(2) Approaches to embed credentials in the GotAPI application

The first approach is based on the trust that may be provided by the operating system and/or the application market ecosystem. Many application market ecosystems provide an Application ID for an application that is guaranteed to be unique in the ecosystem including the operating system and the devices. An Application ID may be used for the trust of the GotAPI application for the authenticity. This approach, however, may be a solution depending on the operating system.

The second approach is based on the credential embedded in the application as the trust. It typically requires an external server to verify the authenticity of the credential of the Server for the application. The challenge of this approach is how to protect the credential that is embedded in GotAPI applications from attackers who are able to reverse engineer the applications. If the same credential is embedded in all the application packages and distributed to many devices and if the credential is compromised on one of the devices, all the devices implementing the application would be compromised.

7.3.3.3 HMAC server authentication using trusted Application ID for the Server spoofing attack

This counter measure works for a platform and a UA that satisfy the following requirements. This is based on the trust provided by the Application ID of the native application, and not embedding any credentials in the GotAPI native application.

- The Application ID is unique and trusted, which is guaranteed by the platform.
- The execution environment, e.g., UA, provides a one-way channel for an application to connect directly and send messages to a native application by designating its Application ID, e.g. a URI scheme.
- The application can be connected exclusively and securely with a native application by designating its Application ID. Namely, there is no eavesdropping, no man-in-the-middle, or no spoofed destination in the channel from the application to the destination native application.

We call this type of channel as “Trusted Channel”.

Note: *Intent URI Scheme* for qualified browsers on Android and *Explicit intents* for Android native application satisfy all these requirements. The destination is designated by the package name of the native application to which applications attempts to send messages.

Other assumptions are:

- The HTTP channel may be eavesdropped (*).
- Any application can terminate other applications that are running in the background and take over the port that the application is listening to.
- The application knows the Application ID of the genuine GotAPI application, implementing both the GotAPI Server and the GotAPI Auth Server. The Application ID is provided to the application out of the band in a trusted manner.

Note: It has been shown that eavesdropping is not possible over the GotAPI-1 or the GotAPI-2 HTTP interfaces on Android unless the device is rooted. Nonetheless, this assumption is introduced here because there may be a way for eavesdropping the HTTP connection that we are not aware of.

If the Trusted Channel is available, the Server spoofing attack is prevented using the HMAC server authentication as follows:

Shared key distribution using the Trusted Channel:

1. The application generates a key, K, composed of unpredictable random characters, and stores the key securely.
2. The application sends the key, K, to the genuine rServer through the Trusted Channel designating the Application ID of the genuine GotAPI application. The application knows the genuine GotAPI application’s Application ID in an out-of-band trusted channel.
3. The genuine GotAPI application stores the key securely.

HMAC calculation and sending messages through the GotAPI-1 or the GotAPI-2 Interface:

1. Before the application sends a request, it creates a nonce, N, which is a series of random digits that is long enough not to be predicted, and, then, it sends the message, M, and the nonce, N, through the GotAPI-1 or the GotAPI-2 Interface.
2. When the genuine Server receives the request, it calculates an HMAC, h'=HMAC (K,N), with the nonce, N, and the key, K, that the application distributed through the Trusted Channel before.
3. The genuine Server sends a response with the HMAC, h', and the response message, m, to the application through the GotAPI-1 or the GotAPI-2 interface.
4. The application calculates an HMAC, H'=HMAC (K, N), and it checks if H' is equal to h' that is received from the GotAPI Server. If equal, verification of the Server authentication is successful and the application verifies that the response has surely been sent by the genuine Server. If not, application determines that the Server that sent the message is spoofed.

The figure below presents a normal case of the HMAC server authentication for the GotAPI Server (the GotAPI Auth Sever case is the same) for a web application.

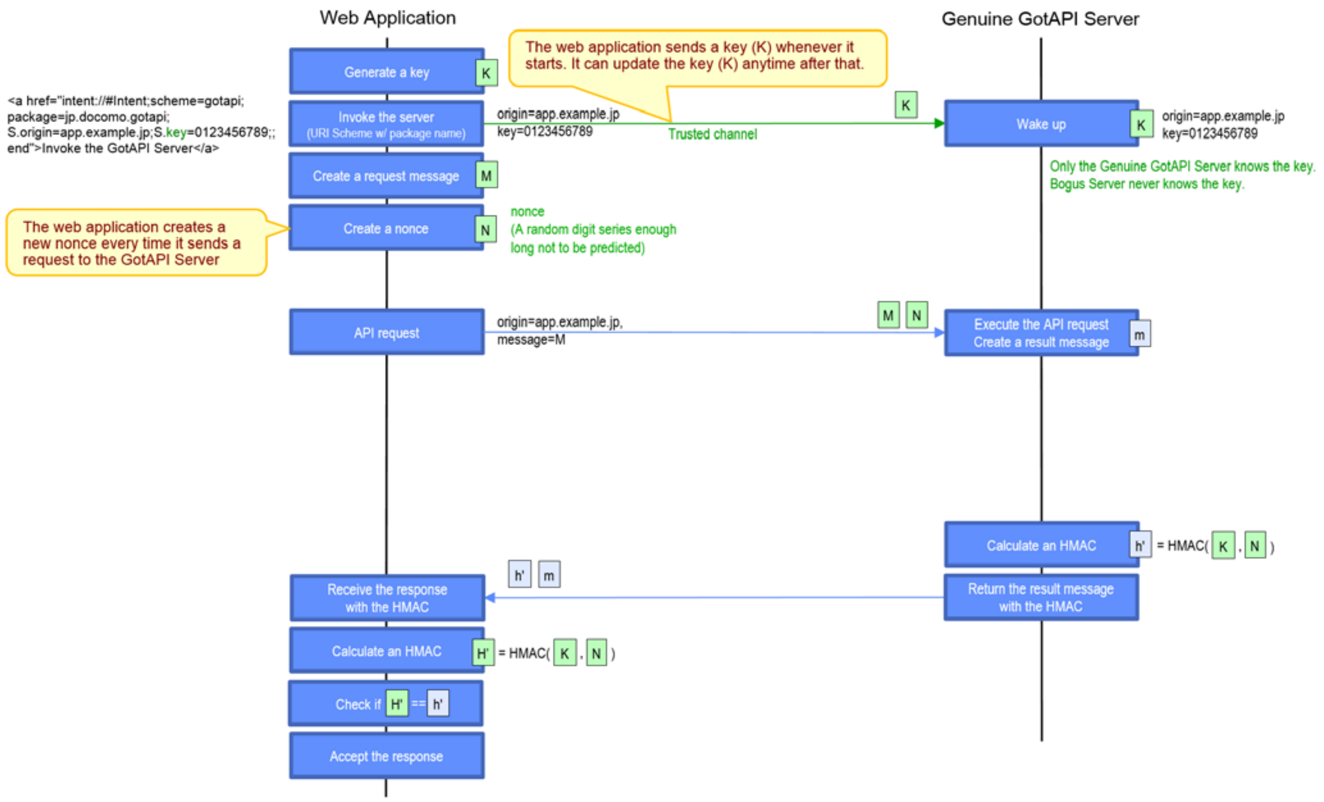

Web Application
Genuine GotAPI Server
Generate a key
K
The web application sends a key (K) whenever it starts. It can update the key (K) anytime after that.
<a href="intent://#Intent;scheme=gotapi;package=jp.docomo.gotapi;S.origin=app.example.jp;S.key=0123456789;;end">Invoke the GotAPI Server</a>
Invoke the server (URI Scheme w/ package name)
origin=app.example.jp key=0123456789
Trusted channel
Wake up
origin=app.example.jp key=0123456789
Only the Genuine GotAPI Server knows the key. Bogus Server never knows the key.
Create a request message
M
The web application creates a new nonce every time it sends a request to the GotAPI Server
Create a nonce
N
nonce (A random digit series enough long not to be predicted)
API request
origin=app.example.jp, message=M
Execute the API request Create a result message
m
Calculate an HMAC
h' = HMAC(K , N)
Receive the response with the HMAC
Return the result message with the HMAC
Calculate an HMAC
H' = HMAC(K , N)
Check if H' == h'
Accept the response

Figure 13: HMAC server authentication – Normal Case

(a web application communicating with the GotAPI Server over GotAPI-1)

The Figure 10 presents a spoofing attack case of the HMAC server authentication, where a web application is communicating with the GotAPI Server over GotAPI-1. The same flows apply to the case of a web application is communicating with the GotAPI Auth Server.

When a bogus server attempts to spoof the genuine GotAPI Server, the bogus server can't calculate a right HMAC because it never knows the key, K, generated by the application.

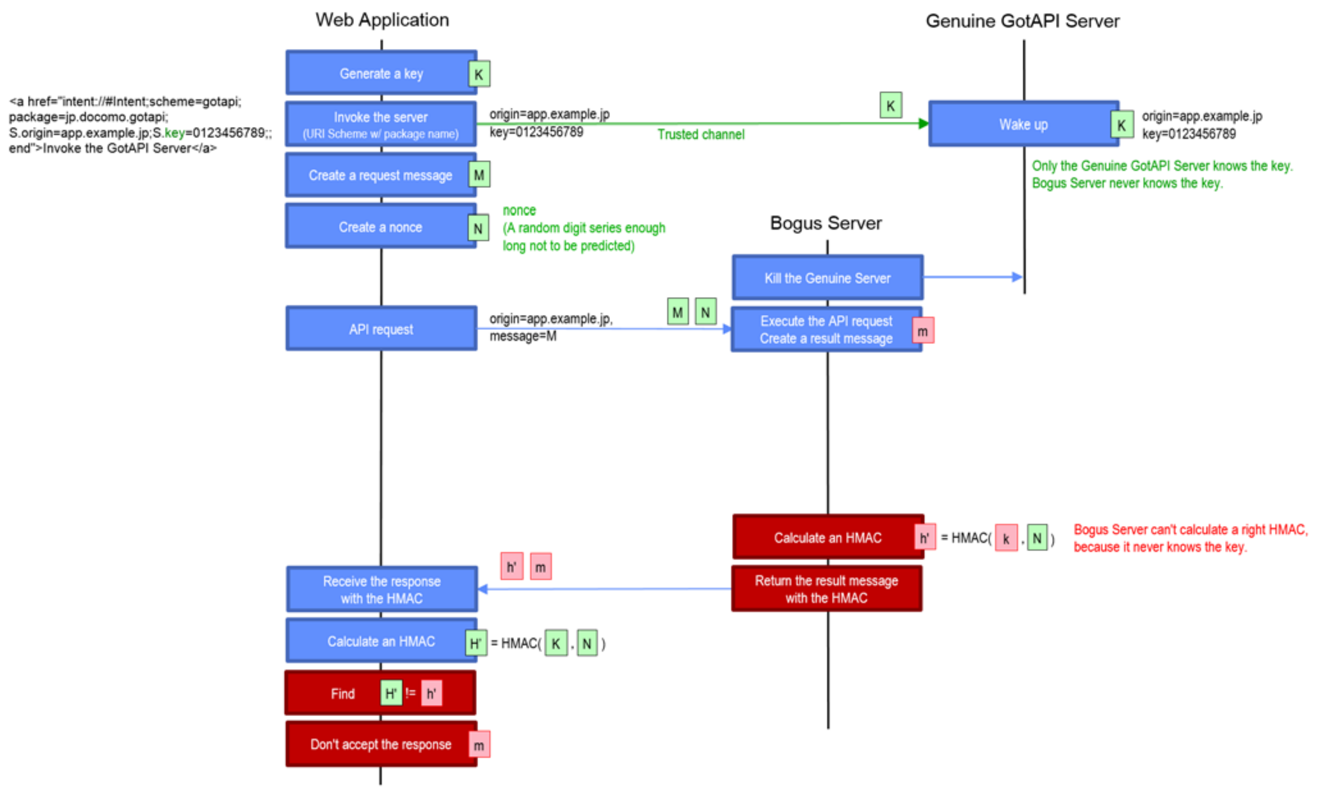


Figure 14: HMAC server authentication – Spoofing Attack Case

(a web application communicating with the GotAPI Server over GotAPI-1)

Since GotAPI Server spoofing is an easy-to-do attack, the GotAPI enabler needs to protect it from the attack.

- If the Trusted Channel is available in the device from the application to the genuine Servers, the GotAPI Server and the GotAPI Auth Server SHOULD support the HMAC server authentication described in this section to prevent the Server spoofing attack.
- If the HMAC server authentication is supported, the GotAPI Server and the GotAPI Auth Server SHALL support SHA-256 for the hash algorithm to calculate an HMAC.
- The GotAPI Server and the GotAPI Auth Server SHALL be able to support and respond to the application regardless of the application being using the HMAC server authentication or not. But the GotAPI Server SHALL NOT respond to applications which have not been authenticated by the GotAPI Auth Server through the GotAPI-2 Interface.
- The GotAPI Server and the GotAPI Auth Server SHALL accept keys sent by applications anytime through the Trusted Channel. The GotAPI Server and the GotAPI Auth Server SHALL calculate an HMAC using the new key that was most recently provided from the application.

Recommendations for applications (non-normative)

- The use of the HMAC server authentication is OPTIONAL for an application.
- An application SHALL generate a new key, K, and use it whenever the application is invoked.
- An application SHALL create a new nonce every time it sends a request to the GotAPI Server.

8. Technical Specifications

8.1 Common APIs for GotAPI applications

In practice, a GotAPI Server and a GotAPI Auth Server are implemented as a single application which is called as a "GotAPI application".

Applications (web applications running on web browsers or OS-specific native applications) have to check if the GotAPI application is alive, then applications have to invoke the GotAPI application if these servers are not running (i.e. if the GotAPI application is not running).

This section defines the Availability API and the way to invoke GotAPI applications.

8.1.1 Availability API on the GotAPI-1 Interface

This API provides the status whether the GotAPI application, consisting of the GotAPI Server and the GotAPI Auth Server, is running or not. The application s (web applications running on web browserss or OS-specific native applications) use this API before the application authorization on the GotAPI-2 Interface. The GotAPI Server and the GotAPI Auth Server SHALL accept and respond to the requests for this API coming from any application even if the application authorization on the GotAPI-2 Interface has not been completed.

Definition of the request

	Definitions
Method	HTTP GET (REST)
Request URL	http://127.0.0.1:4035/gotapi/availability https://127.0.0.1:4036/gotapi/availability
Parameters	none

If the GotAPI application is running, the GotAPI Server and the GotAPI Auth Server SHALL respond as follows:

Definition of the response

	Definitions
MIME-Type	application/json
HTTP status	200 OK

Definition of the response

Name	Sub name	Type	Definition of value	Mandatory/Optional
result		Number	This must be 0.	Mandatory

Example of the response

```
{
    "result":0
}
```

The GotAPI Server and the GotAPI Auth Server SHALL NOT add any extra information in the response for the purpose of protecting the user’s privacy from fingerprinting [RFC6973].

8.1.2 Invoking the GotAPI application

If the application finds that the GotAPI application is not running, it has to invoke the GotAPI application. To do so, the application needs to use an OS-specific or a UA-specific method that enables the application to invoke the GotAPI application.

For example, most of the Android browsers support URI Schemes to invoke a native application with its package name for the web application running on the web browser. If the web application shows a hyper-link embedding the URI scheme, the user can invoke the GotAPI application by tapping it.

Example of the URI Scheme with a package name (Android Chrome)

```
<a href="intent://#Intent;scheme=gotapi;package=com.example.gotapi;S.origin=app.example.jp;S.key=0123456789;;
end">Invoke the GotAPI Server</a>
```

In the implementation model that GotAPI 1.0 supports, the GotAPI Server and GotAPI Auth Server are implemented as a single application. Therefore, when the GotAPI application is invoked, both the GotAPI Server and the GotAPI Auth Server are invoked, so that the application can use the GotAPI-1 and GotAPI-2 interfaces.

If the application is an Android native application, it can use an Explicit Intent for invoking the GotAPI application and sending the necessary information.

(1) HMAC server authentication used:

In order to support the HMAC server authentication as defined in 7.3.3.3, the application is expected to provide the GotAPI application with the following data using the OS-specific or the UA-specific method when invoking the GotAPI application as shown in the example above;

origin,

The origin is an identifier of the application. If the application is a web application, this value is the part of the URL specified as “origin” in RFC6454 (e.g. http://app.example.com). If the application is an OS-specific native application or Hybrid Native/Web App, the application ID recognized by the OS, such as a package name. (e.g. com.example.app).

key

The key is generated by the application, composed of unpredictable random characters. The key is used for HMAC server authentication for the purpose of preventing GotAPI Server spoofing.

When using the HMAC server authentication, it must be ensured that the OS-specific or the UA-specific method to invoke the GotAPI application must qualify as a Trusted Channel as defined in section 7.3.3.3.

When the GotAPI application is started by an application, the GotAPI Server SHALL securely store the origin and key provided by the application.

The GotAPI and the GotAPI Auth Server SHALL retain and associate the origin and key for use in validating subsequent requests from the application.

The GotAPI and the GotAPI Auth Server MAY set an expiration date/time to the pair of the origin and the key, and MAY revoke the pair when the expiration is reached.

GotAPI and the GotAPI Auth Server SHALL allow applications to update their key at any time, and any number of times.

(2) HMAC server authentication not used:

The application may decide whether or not to use the HMAC server authentication. Hence, the application is not mandated to send the origin and the key in the URI Scheme.

If the GotAPI application does not receive the key or receives an empty string as the key from the application, the GotAPI Server and the GotAPI Auth Server SHALL recognize that the application is not using the HMAC server authentication and act accordingly.

GotAPI Servers and the GotAPI Auth Server SHALL support the ability of applications to start using HMAC server authentication at any time.

Recommendations for applications (non-normative)

- The application MAY decide whether or not to use the HMAC server authentication.
- The application MAY update the key at any time, as many times as it likes.
- The application SHOULD update the key whenever the application starts to run, and start the session with a new key.

8.2 GotAPI Authorization Server

This section defines the data format used over the GotAPI-2 Interface.

As described in the section “7.2.2.2 GotAPI-2”, the application has to obtain authorization for accessing the GotAPI-based APIs from the GotAPI Auth Server over the GotAPI-2 Interface. The GotAPI-2 interface is based upon the concepts of OAuth. The request and response are sent over the HTTP protocol.

Firstly, the application sends an application authorization request with the origin of the application, an identifier of the application, to the GotAPI Auth Server. If the application is a Web Application, the origin is provided in the HTTP Origin request header by the browser as described in [CORS]. If the application is an OS-specific native application or a Hybrid Native/Web App, the origin is an application identifier managed by the underlying OS, such as a package name. For example, if the OS is Android, the origin could be “com.example.app”. It has to be set as the value of the HTTP X-GotAPI-Origin request header by the application.

When the GotAPI Auth Server receives the request, the GotAPI Auth Server may check if the origin is acceptable or not. If the origin is acceptable, the GotAPI Auth Server creates a series of random digits, called as a “grant”, that is long enough not to be predicted. Then the GotAPI Auth Server returns the grant to the application. This transaction is defined in section “8.2.1 Grant”.

Secondary, the application sends an access token request with (i) the origin, which is provided by the browser as described above, (ii) the grant, which was obtained in the previous transaction, and (iii) the scope, which is a collection of the functions that the application wants to use.

When the GotAPI Auth Server receives the request, the GotAPI Auth Server asks the user if the application may use the requested scope shown in the dialog box. If the request is acceptable, the GotAPI Auth Server creates a series of random digits, called as an "access token", that is long enough not to be predicted. This transaction is defined in the section “8.2.2 Access token”.

8.2.1 Grant

The application sends an application authorization request as below. The GotAPI Auth Server SHALL be able to receive and process the request that is sent by the application appropriately.

Definition of the request

	Definitions
Method	HTTP GET (REST)
Request URL	http://127.0.0.1:4035/gotapi/authorization/grant

	https://127.0.0.1:4036/gotapi/authorization/grant
HTTP request Header	Origin: This value is the origin of the web application running on the web browser. For example, it could be "http://app.example.com". The application developers do not need to take care of this because this header is automatically set by the web browser. This value is MANDATORY if the application is a web application running on a web browser. X-GotAPI-Origin: This value is the origin of the OS-specific native application or the Hybrid Native/Web App. For example, an Android native application could set this header to "com.example.app". This value is MANDATORY if the application is an OS-specific native application or a Hybrid Native/Web App.
Parameters	None

Example of the request by a web application running on a web browser

```
GET /gotapi/authorization/grant HTTP/1.1

Host: 127.0.0.1

Origin: http://app.example.com

...(Other headers)
```

Example of the request by an OS-specific native application or a Hybrid Native/Web App

```
GET /gotapi/authorization/grant HTTP/1.1

Host: 127.0.0.1

X-GotAPI-Origin: com.example.app

...(Other headers)
```

If the X-GotAPI-Origin header exists in the HTTP request header, the GotAPI Auth Server SHALL assume that the application is an OS-specific native application or a Hybrid Native/Web App and the value is the origin of the application. If both of the X-GotAPI-Origin and the Origin exist in the HTTP request header, the GotAPI Auth Server SHALL take the value of the X-GotAPI-Origin as the origin. If only the Origin header exists in the request header, the GotAPI Server SHALL assume that the application is a web application running on a web browser. If neither the Origin header nor the X-GotAPI-Origin header exists, the GotAPI Server SHALL return the error code as defined by the definition of the JSON format of the response below.

After the GotAPI Auth Server determines if the origin of the application is acceptable or not, it SHALL respond as follows:

Definition of the response

	Definitions
MIME-Type	application/json
HTTP status	200 OK

Definition of the JSON format of the response

Name	Sub name	Type	Definition of value	Mandatory/Optional

result		Number	If the origin was accepted by the GotAPI Auth Server, the value is 0, otherwise an integer other than 0, which indicates an error code. This specification doesn't define error codes.	Mandatory
clientId		String	The grant which the GotAPI Auth Server created for the accepted origin. If the origin was not accepted, this value must be an empty string.	Mandatory
errorCode		String	If the origin was not accepted or an error occurred, this value is set to an integer other than 0. Otherwise, this value must be 0. This specification doesn't define error codes.	Mandatory
errorMessage		String	If the origin was not accepted or an error occurred, this value is set to a human-readable letter string describing the error. Otherwise, this value must be an empty string.	Mandatory
hmac		String	An HMAC generated for the counter measure against the Server spoofing attack described in the section "7.4.3.2.1 HMAC server authentication using trusted Application ID for the Server spoofing attack".	Mandatory if the application provides a key to the GotAPI Auth Server

Example of the response when the origin was accepted successfully

```
{

    "result":0,

    "clientId": "0123456789",

    "errorCode": 0,

    "errorMessage": "" ,

    "hmac": "0123.....xyz"

}
```

Example of the response when the origin was not accepted

```
{

    "result":1,

    "clientId": "",

    "errorCode": 1,

    "errorMessage": "The application is not an official application of the Example Service." ,

    "hmac": "0123.....xyz"

}
```

The GotAPI Auth Server MAY append additional name-value pairs in the JSON data as needed.

8.2.2 Access token

If the application receives the grant (clientId) from the GotAPI Auth Server, the application sends an access token request as below. The GotAPI Auth Server SHALL be able to receive and process the request that is sent by the application appropriately.

Definition of the request

	Definitions
Method	HTTP GET (REST)
Request URL	http://127.0.0.1:4035/gotapi/authorization/accesstoken https://127.0.0.1:4036/gotapi/authorization/accesstoken
HTTP request Header	Origin: This value is the origin of the web application running on the web browser. For example, it could be "http://app.example.com". As this header is automatically set by the web browser, the developer of the application does not need to take care of it. This value is MANDATORY if the application is a web application running on a web browser. X-GotAPI-Origin: This value is the origin of the OS-specific native application or the Hybrid Native/Web App. For example, the origin of an Android native application, it could be "com.example.app". This value is MANDATORY if the application is an OS-specific native application or a Hybrid Native/Web App.
Parameters	clientId: This value is the grant that the application received from the GotAPI Auth Server previously. scope: The list of functions that the application wants to use. This value is a comma-separated string such as "notification,vibration". This value must not include any white-space. This specification doesn't define the names of the functions. The names of the functions are defined by the GotAPI service provider. applicationName: This value is the name of the application. This parameter is OPTIONAL. This value will be shown to the user when the user is requested for authorization of the application.

Example of the request by a web application running on a web browser

```
GET /gotapi/authorization/accesstoken?clientId=0123456789&scope=notification,vibration&
applicationName=Smart%20Watch&20Controller HTTP/1.1

Host: 127.0.0.1

Origin: http://app.example.com

...(Other headers)
```

Example of the request by an OS-specific native application or a Hybrid Native/Web App

```
GET /gotapi/authorization/accesstoken?clientId=0123456789&scope=notification,vibration&
applicationName=Smart%20Watch&20Controller HTTP/1.1
```

```
Host: 127.0.0.1

X-GotAPI-Origin: com.example.app

...(Other headers)
```

If the X-GotAPI-Orign header exists in the HTTP request header, the GotAPI Auth Server SHALL assume that the application is an OS-specific native application or a Hybrid Native/Web App and the value is the origin of the application. If both of the X-GotAPI-Origin and Origin exist in the HTTP request request header, the GotAPI Auth Server SHALL take the value of the X-GotAPI-Origin as the origin. If only Origin header exists in the request header, the GotAPI Server SHALL assume that the application is a web application running on a web browser. If neither the Origin header nor the X-GotAPI-Origin header exists, the GotAPI Server SHALL return the error code as defined by the definition of the JSON format of the response below.

If the origin is accepted as defined in the section “8.2.1 Grant”, the GotAPI Auth Server SHALL ask the user if the application may use the requested scope as described in the section “7.2.2.2 GotAPI-2”. If the origin is not accepted, the GotAPI Server SHALL send a response immediately without asking the user anything.

The GotAPI Auth Server SHALL respond as follows:

Definition of the response

	Definitions
MIME-Type	application/json
HTTP status	200 OK

Definition of the JSON format of the response

Name	Sub name	Type	Definition of value	Mandatory/Optional
result		Number	If the requested scope is authorized by the user through the GotAPI Auth Server, the value is 0, otherwise an integer other than 0, which indicates an error code. This specification doesn't define error codes.	Mandatory
accessToken		String	The access token which the GotAPI Auth Server created for the authorized scope requested by the application. If the scope were not authorized, this value must be an empty string.	Mandatory
errorCode		String	If the requested scope was not authorized or an error occurred, this value is set to an integer other than 0. Otherwise, this value must be 0. This specification doesn't define error codes.	Mandatory
errorMessage		String	If the requested scope was not authorized or an error occurred, this value is set to a human-readable letter string describing the error. Otherwise, this value must be an empty string.	Mandatory
hmac		String	An HMAC generated for the counter measure against the Server spoofing attack described in the section "7.4.3.2.1 HMAC server authentication using trusted Application ID for the Server spoofing attack".	Mandatory if the application provides a key to the GotAPI Auth Server

Example of the response when the requested scope was authorized successfully

```
{

    "result": 0,

    "accessToken": "9876543210",

    "errorCode": 0,

    "errorMessage": "" ,

    "hmac": "0123.....xyz"

}
```

Example of the response when the requested scope was not authorized

```
{

    "result":1,

    "accessToken": "",

    "errorCode": 1,

    "errorMessage": "The requested functions are not available." ,

    "hmac": "0123.....xyz"

}
```

The GotAPI Auth Server MAY append additional name-value pairs in the JSON data as needed.

8.3 GotAPI Server

8.3.1 Service Discovery API on the GotAPI-1 Interface

This API provides the information about what services the GotAPI Server can supply. This API is available only if the application authorization on the GotAPI-2 Interface is completed.

Definition of the request

	Definitions
Method	HTTP GET (REST)
Request URL	http://127.0.0.1:4035/gotapi/servicediscovery https://127.0.0.1:4036/gotapi/servicediscovery
Parameters	None

Definition of the request parameters

Parameter name	Definition of value
accessToken	The access token obtained from the GotAPI Auth Server through the GotAPI-2 Interface.

Example of the request URL

```
http://127.0.0.1:4035/gotapi/servicediscovery?accessToken=0987654321
```

When the GotAPI Server receives the request, it SHALL run the Plug-In Discovery procedure described in the section “7.2.2.4.1 Plug-In Discovery” and the section “8.3.4 Plug-In discovery on the GotAPI-4 Interface”.

When the GotAPI Server completes the Plug-In Discovery procedure, it SHALL respond as follows:

Definition of the response

	Definitions
MIME-Type	application/json
HTTP status	200 OK

Definition of the JSON format of the response

Name	Sub name	Type	Definition of value	Mandatory/Optional
result		Number	If success, the value is 0, otherwise an integer other than 0, which indicates an error code. This specification doesn't define error codes.	Mandatory
product		String	The name of the GotAPI Server (e.g. "ABConnect")	Mandatory
version		String	The version of the GotAPI Server (e.g. "1.0").	Mandatory
services		Array	The list of the services. If none of services were found, an empty array is returned.	Mandatory
	serviceId	String	The service identifier. If this service represents the external device which is connected through the relevant Plug-In, this value must be the device identifier (deviceId).	Mandatory
	name	String	The name of the service. If this service represents the external device which is connected through the relevant Plug-In, this value must be the name of the external device.	Mandatory
	manufacturer	String	The manufacturer of the service. If this service represents the external device which is connected through the relevant Plug-In, this value must be the manufacturer of the external device. Otherwise, this value must be the Plug-In provider name.	Optional
	version	String	The version of the service. If this service represents the external device which is connected through the relevant Plug-In, this value must be the version of the external device. Otherwise, this value must be the Plug-In version.	Optional
	type	String	The type of the service. If this service represents the external device which is connected through the relevant Plug-In, this value represents the type of the network used to connect to the external device. The value must be any one of "WiFi", "BLE", "NFC", or "Bluetooth".	Optional
	online	Boolean	If the service is available at the time, this value is true. Otherwise false.	Mandatory
hmac		String	An HMAC generated for the counter measure against the GotAPI Server spoofing attack described in the	Mandatory if the application provides

	section "7.4.3.2.1 HMAC server authentication using trusted Application ID for the GotAPI Server spoofing attack".	a key to the GotAPI Server

The following example shows that the GotAPI Server has discovered two services. The first one is a service provided by an internal enabler Plug-In, which provides CPU information of the local device. The second one is a service provided by an external device.

Example of the response

```
{
    "result":0,
    "product": "ABConnect",
    "version": "1.0",
    "services":[
        {
            "id":"deviceinfo.plugin1.example.org",
            "name":"Local Device Information Analyzer",
            "manufacturer": "ABC Software Inc.",
            "version": "2.3",
            "type":"cpuInfo",
            "online":true
        },
        {
            "id":"device1.plug-in2.example.org",
            "name":"Smart watch DC01A",
            "manufacturer": "ABC Electric Inc.",
            "version": "3.0",
            "type":"WiFi",
            "online":true
        }
    ],
    "hmac": "0123.....xyz"
}
```

The GotAPI Server MAY append additional name-value pairs in the JSON data as needed.

8.3.2 Service Information API on the GotAPI-1 Interface

This API provides the detailed information of the service provided by internal capabilities of the host device and external devices connected through the relevant Plug-In. This API is available only if the application authorization on the GotAPI-2 Interface is completed.

Definition of the request

	Definitions
Method	HTTP GET (REST)
Request URL	http://127.0.0.1:4035/gotapi/serviceinformation https://127.0.0.1:4036/gotapi/serviceinformation

Definition of the request parameters

Parameter name	Definition of value
serviceId	The identifier of the targeted service. This value is available from the Service Discovery API on the GotAPI-1 Interface.
accessToken	The access token obtained from the GotAPI Auth Server through the GotAPI-2 Interface.

Example of the request URL

```
http://127.0.0.1:4035/gotapi/serviceinformation?serviceId=abcdefg123&accessToken=0987654321
```

When the GotAPI Server receives the request, it SHALL respond as follows:

Definition of the response

	Definitions
MIME-Type	application/json
HTTP status	200 OK

Definition of the JSON format of the response

Name	Sub name	Type	Definition of value	Mandatory/Optional
result		Number	If success, the value is 0, otherwise an integer greater than 0, which indicates an error code. This specification doesn't define error codes.	Mandatory
product		String	The name of the GotAPI Server (e.g. "ABConnect")	Mandatory
version		String	The version of the GotAPI Server (e.g. "1.0").	Mandatory
connect		Object		Mandatory
	wifi	Boolean	If the external device are available through WiFi, the value is true, otherwise false. If the external device doesn't support the WiFi connection, this name-value pair must not exist.	Mandatory if the external device supports the WiFi connection.
	bluetooth	Boolean	If the external device are available through Bluetooth, the value is true, otherwise false. If the external device doesn't support the Bluetooth connection, this name-value pair must not exist.	Mandatory if the external device supports the Bluetooth connection.
	nfc	Boolean	If the external device are available through NFC, the value is true, otherwise false. If the external device doesn't support the NFC connection, this name-value pair must not exist.	Mandatory if the external device supports the NFC connection.
	ble	Boolean	If the external device are available through BLE, the value is true, otherwise false.	Mandatory if the external device

		If the external device doesn't support the BLE connection, this name-value pair must not exist.	supports the BLE connection.
supports	Array	The list of the available API names. This specification doesn't define the names.	Mandatory
hmac	String	An HMAC generated for the counter measure against the GotAPI Server spoofing attack described in the section "7.4.3.2.1 HMAC server authentication using trusted Application ID for the GotAPI Server spoofing attack".	Mandatory if the application provide a key to the GotAPI Server

Example of the response

```
{
    "result":0,
    "product": "ABConnect",
    "version":"1.0",
    "connect":[
        "wifi":true
    ],
    "supports":[
        "system",
        "battery",
        "vibration"
    ],
    "hmac": "0123.....xyz "
}
```

The GotAPI Server MAY append additional name-value pairs in the JSON data as needed.

8.3.3 Common data set of responses on the GotAPI-1 Interface

The GotAPI Server SHALL include the following data in all responses on the GotAPI-1Interface. The data set SHALL not be used for other purposes.

Common data set for the response on the GotAPI-1 Interface

Name	Sub name	Type	Definition of value	Mandatory/Optional
product		String	The name of the GotAPI Server (e.g. "ABConnect")	Mandatory
version		String	The version of the GotAPI Server (e.g. "1.0").	Mandatory
hmac		String	An HMAC generated for the counter measure against the GotAPI Server spoofing attack described in the section "7.4.3.2.1 HMAC server authentication using trusted Application ID for the GotAPI Server spoofing attack".	Mandatory if the application provide a key to the GotAPI Server

8.3.4 Plug-In discovery on the GotAPI-4 Interface

When the application requests the Service Discovery described in the section “8.3.1 Service Discovery API on the GotAPI-1 Interface”, the GotAPI Server SHALL find the installed Plug-Ins and obtain information from each, and, then, ask the Plug-Ins about what devices it provides access to are available at the time. After all the Plug-Ins have responded, the GotAPI Server SHALL return the results of the Plug-In discovery to the application. See the section “7.2.2.4.1 Plug-In Discovery” for the detailed architecture.

Using the GotAPI-4 Interface, the GotAPI Server sends the data object for the Plug-In discovery request to all the installed Plug-Ins as defined blow:

Definition of the data object for the Plug-In discovery request

Name	Sub name	Type	Definition of value	Mandatory/Optional
receiver		String	The address of the GotAPI Server application used by Plug-Ins. Generally, it is the application ID recognized by the OS.	Mandatory
requestCode		int	A request code identifying the request. This value could be any number but must MUST be an integer greater than 0, and unique for each open request, to ensure responses can be correlated.	Mandatory
api		String	The value must be "gotapi".	Mandatory
profile		String	The value must be "networkServiceDiscovery".	Mandatory
attribute		String	The value must be "getNetworkServices".	Mandatory

The GotAPI Server MAY append additional data in the data object as needed.

This data object is sent to the Plug-Ins in an OS specific mechanism, .e.g., Intents for Android.

Requirements for OS-specific request channel and data container

OS	Description
Android	The GotAPI Server must use Explicit Intents for the request. The data object must be mapped to the Extra directly.

Example of the data object of the Android Implicit Intents

Name	Sub name	Example of value	Note
Action		"org.deviceconnect.action.GET"	This value is defined by the GotAPI Server application.
Component		"org.example.plugin"	This value is the package name of the Plug-In application.
Extra			
	receiver	"org.deviceconnect"	This value is the package name of the GotAPI Server.
	requestCode	1	
	api	"gotapi"	
	profile	"serviceDiscovery"	
	attribute	"services"	
	config	*"additional parameters"*	*This name-value pair is an additional data which is not defined by this specification.*

When the Plug-In receives the Plug-In discovery request from the GotAPI Server, the Plug-In SHALL determine whether the corresponding devices are available at the time, and, then, SHALL send the data object of the Plug-In discovery response to the GotAPI Server as defined below:

Definition of the data object for the Plug-In discovery response

Name	Sub name	Type	Definition of value	Mandatory/Optional
requestCode		int	The request code coming from the GotAPI Server.	Mandatory
result		int	If success, the value is 0, otherwise an integer other than 0, which indicates an error code. This specification doesn't define error codes.	Mandatory
services		Array		Mandatory
	serviceId	String	The service identifier representing the external device or internal service. This id must be a string which is the concatenation of the identifier of the Plug-In and the identifier of the device. For example, if the identifier of the Plug-In is "org.example.plugin" and the identifier of the external device is "12345", the id could be "com.example.plugin.12345".	Mandatory
	name	String	The name of the device.	Mandatory
	manufacturer	String	The manufacturer of the service. If this service represents the external device which is connected through the relevant Plug-In, this value must be the manufacturer of the external device. Otherwise, this value must be the Plug-In provider name.	Optional
	version	String	The version of the service. If this service represents the external device which is connected through the relevant Plug-In, this value must be the version of the external device. Otherwise, this value must be the Plug-In version.	Optional
	type	String	The type of the network used to connect to the device. The value must be any one of "WiFi", "BLE", "NFC", or "Bluetooth".	Optional
	online	Boolean	True if the device is online and available, False otherwise.	Mandatory
	scopes	Array	The list of profiles that the application wants to use. (e.g. ["file", "notification", "vibration"]). A profile is a set of functions supported by the Plug-In. This specification doesn't define the profile names. The profile names are defined by Plug-Ins.	Mandatory

The Plug-In MAY append additional data in the data object as needed.

This data object is sent to the Plug-Ins in an OS specific mechanism, .e.g., Intents for Android.

Requirements for OS-specific response channel and data container

OS	Description
Android	The GotAPI Server must use Explicit Intents for the response. The data object must be mapped to the Extra directly.

Example of the data object of the Android Explicit Intents

Name	Sub name	Example of value	Note
Action		"org.deviceconnect.action.RESPONSE"	This value is defined by the GotAPI Server application.
Component		"org.deviceconnect"	This value is the package name of the GotAPI Server application.
Extra			
	requestCode	1	
	result	0	

	services	*[Array Object]*	This value is an example. Note that this is "not" a JSON string. This value must be an Array object whose content is the same as the following JSON example:
			`[`
			`  {`
			`    "id": "org.example.plugin.12345",`
			`    "name": "Smart Watch S23P",`
			`    "manufacturer": "ABC Electric Inc.",`
			`    "version": "3.0",`
			`    "type": "Bluetooth",`
			`    "online": true,`
			`    "scopes": [`
			`      "notification",`
			`      "vibration"`
			`    ]`
			`  },`
			`  ...`
			`]`

	config	*"additional parameters"*	*This name-value pair is an additional data which is not defined by this specification.*

When the GotAPI Server receives the responses from the Plug-Ins through the Plug-In discovery, it SHALL create a mapping table which associates the service identifier with the relevant Plug-In.

The application doesn't care about what services are associated with what Plug-Ins. When the application sends a request on the GotAPI-1 Interface, it specifies only the service identifier. When the GotAPI Server receives a request with a serviceId on the GotAPI-1 Interface from the application, the GotAPI Server SHALL determine the relevant Plug-In from the mapping table.

As described in the table "Definition of the data object for the Plug-In discovery response", the id of each service (external device or internal service) is a string which is the concatenation of the identifier of the Plug-In and the identifier of the service. That is, the id of the service should be unique in a host device. But every Plug-Ins may not necessarily follow the requirement. If multiple Plug-Ins report a same service id, the in the GotAPI Server SHALL take the first reported service id and the others SHALL be ignored.

8.3.5 Plug-In approval on the GotAPI-4 Interface

As described in the section “7.2.2.4.3 Approval”, when an application requests an API access to a device through the corresponding Plug-In for the first time, the GotAPI Server SHALL obtain a permission for the API accesses from the Plug-In. This procedure is composed of the two steps:

(1) The GotAPI Server requests registration of the application to the Plug-In.

(2) The GotAPI Server requests an access token to the Plug-In

This section defines the data object of the requests and responses for each step.

8.3.5.1 Request for registration of application

When an application requests an API access to a device through the corresponding Plug-In for the first time, the GotAPI Server SHALL send an app registration request and get a clientId from the Plug-In.

To get a clientId from the Plug-In, the GotAPI Server SHALL sends the data object to the Plug-In as defined blow:

Definition of the data object for the app registration request

Name	Sub name	Type	Definition of value	Mandatory/Optional
receiver		String	The address of the GotAPI Server application used by Plug-Ins. Generally, it is the application ID recognized by the OS, such as a package name.	Mandatory
requestCode		int	A request code identifying the request. This value could be any number but must be an integer greater than 0.	Mandatory
api		String	The value must be "gotapi".	Mandatory
profile		String	The value must be "authorization".	Mandatory
attribute		String	The value must be "createClient".	Mandatory
package		String	The identifier of the application. If the application is a web application running on a web browser, this value is the origin of the application. If the application is an OS-specific native application or a Hybrid Native/Web App, this value is the application ID recognized by the OS, such as a package name.	Mandatory

The GotAPI Server MAY append additional data in the data object as needed.

This data object is sent to the Plug-Ins in an OS specific mechanism, .e.g., Intents for Android.

Requirements for OS-specific request channel and data container

OS	Description
Android	The GotAPI Server must use Explicit Intents for the request. The data object must be mapped to the Extra directly.

Example of the data object of the Android Explicit Intents

Name	Sub name	Example of value	Note
Action		"org.deviceconnect.action.GET"	This value is defined by the GotAPI Server application.
Component		"org.example.plugin"	This value is the package name of the

			Plug-In application.
Extra			
	receiver	"org.deviceconnect"	This value is the package name of the GotAPI Server application.
	requestCode	1	
	api	"gotapi"	
	profile	"authorization"	
	attribute	"createClient"	
	package	"http://example.com"	
	config	*"additional parameters"*	*This name-value pair is an additional data which is not defined by this specification.*

When the Plug-In receives the application registration request from the GotAPI Server, the Plug-In SHALL create a clientId for the application. The Plug-In MAY have a white-list of applications whose origin or application ID has been approved to access the Plug-In. If the application isn't found in the white-list, the Plug-In MAY deny this request. Note that the details of the approval of the origin of the applications is out of the scope this specification.

If the Plug-In accepts this request, the Plug-In SHALL send the data object to the GotAPI Server as defined below:

Definition of the data object for the app registration response

Name	Sub name	Type	Definition of value	Mandatory/Optional
requestCode		int	The request code coming from the GotAPI Server.	Mandatory
result		int	If success, the value is 0, otherwise an integer greater than 0, which indicates an error code. This specification doesn't define error codes.	Mandatory
clientId		String	The identifier of the application, which is generated by the Plug-In.	Mandatory

The Plug-In MAY append additional data in the data object as needed.

This data object is sent to the Plug-Ins in an OS specific mechanism, .e.g., Intents for Android.

OS-specific response channel and data container

OS	Description
Android	The GotAPI Server must use Explicit Intents for the response. The data object must be mapped to the Extra directly.

Example of the data object of the Android Explicit Intents

Name	Sub name	Example of value	Note
Action		"org.deviceconnect.action.RESPONSE"	This value is defined by the GotAPI Server application.
Component		"org.deviceconnect"	This value is the package name of the GotAPI Server application.
Extra			

	requestCode	1	
	result	0	
	clientId	"123ABC"	
	config	*"additional parameters"*	*This name-value pair is an additional data which is not defined by this specification.*

8.3.5.2 Request for an access token

Immediately after the GotAPI Server has received a clientId from the Plug-In, the GotAPI Server SHALL send the data object to the Plug-In in order to get an access token as defined below:

Definition of the data object for the access token request

Name	Sub name	Type	Definition of value	Mandatory/Optional
receiver		String	The address of the GotAPI Server application used by Plug-Ins. Generally, it is the application ID recognized by the OS, such as a package name.	Mandatory
requestCode		int	A request code identifying the request. This value could be any number but must be an integer greater than 0.	Mandatory
serviceId		String	The identifier of the targeted Service. This value is provided by the application over the GotAPI-1 Interface.	Mandatory
api		String	The value must be "gotapi".	Mandatory
profile		String	The value must be "authorization".	Mandatory
attribute		String	The value must be "requestAccessToken".	Mandatory
package		String	The identifier of the application. If the application is a web application running on a web browser, this value is the origin of the application. If the application is an OS-specific native application or a Hybrid Native/Web App, this value is the application ID recognized by the OS, such as a package name.	Mandatory
clientId		String	The identifier of the application, which is generated by the Plug-In.	Mandatory

The GotAPI Server MAY append additional data in the data object as needed.

This data object is sent to the Plug-Ins in an OS specific mechanism, .e.g., Intents for Android.

Requirements for OS-specific request channel and data container

OS	Description
Android	The GotAPI Server must use Explicit Intents for the request. The data object must be mapped to the Extra directly.

Example of the data object of the Android Explicit Intents

Name	Sub name	Example of value	Note
Action		"org.deviceconnect.action.GET"	This value is defined by the GotAPI Server application.

Component		"org.example.plugin"	This value is the package name of the Plug-In application.
Extra			
	receiver	"org.deviceconnect"	This value is the package name of the GotAPI Server application.
	requestCode	1	
	serviceId	"device1.localhost.deviceconnect.org"	
	api	"gotapi"	
	profile	"authorization"	
	attribute	"requestAccessToken"	
	package	"http://example.com"	
	clientId	"123ABC"	
	config	*"additional parameters"*	*This name-value pair is an additional data which is not defined by this specification.*

When the Plug-In receives the access token request from the GotAPI Server, the Plug-In SHALL ask the user if the user permits the application to accesses the Plug-In. For example, the Plug-In MAY show a yes/no confirmation dialog box to the user.

If the user confirms permission for the application to access the Plug-In, the Plug-In SHALL create an access token, and send the data object to the GotAPI Server as defined below:

Definition of the data object for the access token response

Name	Sub name	Type	Definition of value	Mandatory/Optional
requestCode		int	The request code coming from the GotAPI Server.	Mandatory
result		int	If success, the value is 0, otherwise an integer greater than 0, which indicates an error code. This specification doesn't define error codes.	Mandatory
accessToken		String	The access token for the application, which is generated by the Plug-In.	Mandatory
expire		long	The unix time representing the expiration date of the access token.	Mandatory

The Plug-In MAY append additional data in the data object as needed.

This data object is sent to the Plug-Ins in an OS specific mechanism, .e.g., Intents for Android.

Requirements for OS-specific response channel and data container

OS	Description
Android	The GotAPI Server must use Explicit Intents for the response. The data object must be mapped to the Extra directly.

Example of the data object of the Android Intents

Name	Sub name	Example of value	Note

Action		"org.deviceconnect.action.RESPONSE"	This value is defined by the GotAPI Server application.
Component		"org.deviceconnect"	This value is the package name of the GotAPI Server application.
Extra			
	requestCode	1	
	result	0	
	accessToken	"0123456789abcdef"	
	expire	1413423117	
	config	*"additional parameters"*	*This name-value pair is an additional data which is not defined by this specification.*

8.3.6 Common data set on the GotAPI-4 Interface

As described in the section "7.2.2.4.4 Data Forwarding", once a connection between the GotAPI Server and the targeted Plug-In is established (i.e., GotAPI-4 Plug-In Discovery and GotAPI-4 Approval have been successfully completed), the application can communicate with the targeted Plug-In. The GotAPI Server passes the data transferred between the application and the Plug-In transparently without any modification. But when the GotAPI Server communicates with the Plug-In on the GotAPI-4 interface, some information is to be added for the purpose of interoperability. This section defines the common data set.

The data names defined in the table below are reserved as the common data set for the request from the GotAPI Server to the Plug-In. When the GotAPI Server sends a request to the Plug-In, it SHALL send the data object defined in the table below. The GotAPI Server SHALL not use the data names defined in the table below for other purposes.

Definition of the common data set for the request from the GotAPI Server to the Plug-In

Name	Sub name	Type	Definition of value	Mandatory/Optional
receiver		String	The address of the GotAPI Server application used by Plug-Ins. Generally, it is the application ID recognized by the OS, such as a package name.	Mandatory
requestCode		int	A request code identifying the request. This value could be any number but must be an integer greater than 0.	Mandatory
serviceId		String	The identifier of the targeted Service. This value is provided by the application over the GotAPI-1 Interface.	Mandatory
api		String	The value must be "gotapi".	Mandatory
profile		String	Plug-In specific function name. This specification doesn't define the name.	Mandatory
attribute		String	Plug-In specific attribute name. This specification doesn't define the name.	Mandatory
clientId		String	The identifier of the application, which is generated by the Plug-In.	Mandatory
accessToken		String	The access token for the application, which is generated by the Plug-In.	Mandatory

The GotAPI Server MAY append additional data in the data object as needed.

This data object is sent to the Plug-Ins in an OS specific mechanism, .e.g., Intents for Android.

Requirements for OS-specific request channel and data container

OS	Description
Android	The GotAPI Server must use Explicit Intents for the request. The data object must be mapped to the Extra directly.

Example of the data object of the Android Intents

Name	Sub name	Example of value	Note
Action		"org.deviceconnect.action.GET"	This value is defined by the GotAPI Server application.
Component		"org.example.plugin"	This value is the package name of the Plug-In application.
Extra			
	receiver	"org.deviceconnect"	This value is the package name of the GotAPI Server application.
	requestCode	1	
	serviceId	"device1.localhost.deviceconnect.org"	
	api	"gotapi"	
	profile	"vibration"	
	attribute	"vibrate"	
	clientId	"123ABC"	
	accessToken	"0123456789abcdef"	
	config	*"additional parameters"*	*This name-value pair is an additional data which is not defined by this specification.*

The data names defined in the table below are reserved as the common data set for the response from the Plug-In to the GotAPI Server. When the Plug-In sends a response to the GotAPI Server, it SHALL send the data object defined in the table below. The Plug-In SHALL not use the data names defined in the table below for other purposes.

Definition of the common data set for the response from the Plug-In to the GotAPI Server

Name	Sub name	Type	Definition of value	Mandatory/Optional
requestCode		Number	The request code coming from the GotAPI Server.	Mandatory
result		Number	If success, the value is 0, otherwise an integer greater than 0, which indicates an error code. This specification doesn't define error codes.	Mandatory

The Plug-In MAY append additional data in the data object as needed.

This data object is sent to the Plug-Ins in an OS specific mechanism, .e.g., Intents for Android.

Requirements for OS-specific response channel and data container

OS	Description
Android	The GotAPI Server must use Explicit Intents for the request.

	The data object must be mapped to the Extra directly.

Example of the data object of the Android Intents

Name		Example of value	Note
Action		"org.deviceconnect.action.RESPONSE"	This value is defined by the GotAPI Server application.
Component		"org.deviceconnect"	This value is the package name of the GotAPI Server application.
Extra			
	requestCode	1	
	result	0	
	config	*"additional parameters"*	*This name-value pair is an additional data which is not defined by this specification.*

The Plug-In MAY append additional name-value pairs in the data object as needed.

This data object is sent to the Plug-Ins in an OS specific mechanism, .e.g., Intents for Android.

9. Release Information

9.1 Supporting File Document Listing

Doc Ref	Permanent Document Reference	Description
Supporting File		

Table 7: Listing of Supporting Documents in GotAPI 1.0 Release

9.2 OMNA Considerations

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
n/a	n/a	No prior version

A.2 Draft/Candidate Version 1.0 History

Document Identifier	Date	Sections	Description
Draft Versions OMA-ER-GotAPI-V1_0	04 Feb 2014	All	Initial baseline document.
	03 Jun 2014		Updates for agreed CRs: OMA-CD-GotAPI-2014-0002-CR_GotAPI_ER_Updates OMA-CD-GotAPI-2014-0003-CR_Architecture OMA-CD-GotAPI-2014-0008-CR_GotAPI_URI_Scheme_and_Availability_API OMA-CD-GotAPI-2014-0007-CR_Multiple_GotAPI_Server_Support
	12 Aug 2014	Various	Updates for agreed CRs: OMA-CD-GotAPI-2014-0012R01-CR_Security_etc OMA-CD-GotAPI-2014-0013R01-CR_Technical_Outline
	25 Aug 2014	6.1, 7.3.2.1.1, 7.3.2.4	Updates for agreed CRs: OMA-CD-GotAPI-2014-0025R01-CR_AD_description_of_Temporary_Server_Feed__for_GotAPI_1 OMA-CD-GotAPI-2014-0026R01-CR_Merged_Plug_In_CRs__0015R01_0020_0021_and_0022_
	06 Nov 2014	Various	Updates for agreed CRs: OMA-CD-GotAPI-2014-0028-CR_New_architecture_for_the_Plug_In_Discovery OMA-CD-GotAPI-2014-0030R03-CR_Section_8.6_GotAPI_Server OMA-CD-GotAPI-2014-0031R01-CR_Adding_GotAPI_Server_spoofing_attack_to_7.4.3 OMA-CD-GotAPI-2014-0034R01-CR_Section_7.3.2.2_GotAPI_2 OMA-CD-GotAPI-2014-0035R01-CR_8.7_GotAPI_Authorization_Server
	20 Nov 2014	Various	Updates for agreed CRs: OMA-CD-GotAPI-2014-0037-CR_CONR_1_Scope OMA-CD-GotAPI-2014-0038-CR_CONR_2_References OMA-CD-GotAPI-2014-0039-CR_CONR_5_GotAPI_Enabler_release_description OMA-CD-GotAPI-2014-0040-CR_CONR_6_Requirements OMA-CD-GotAPI-2014-0041-CR_CONR_7_Dependencies OMA-CD-GotAPI-2014-0042-CR_CONR_7_Architectural_Diagram OMA-CD-GotAPI-2014-0043-CR_CONR_7_GotAPI_Server OMA-CD-GotAPI-2014-0044-CR_CONR_7_GotAPI_Authorization_Server OMA-CD-GotAPI-2014-0045-CR_CONR_7_GotAPI_1 OMA-CD-GotAPI-2014-0046-CR_CONR_7_GotAPI_2 OMA-CD-GotAPI-2014-0047-CR_CONR_7_GotAPI_3 OMA-CD-GotAPI-2014-0048-CR_CONR_7_GotAPI_4 OMA-CD-GotAPI-2014-0049-CR_CONR_7_Confidentialty_and_Integrity OMA-CD-GotAPI-2014-0050-CR_CONR_7_Immunity_form_Attack OMA-CD-GotAPI-2014-0051-CR_CONR_8_Technical_Specifications
	04 Dec 2014	Various	Updates for agreed CRs: OMA-CD-GotAPI-2014-0052-CR_Hybrid_Apps OMA-CD-GotAPI-2014-0053-CR_Concept_of_Service_provided_by_Plug_In OMA-CD-GotAPI-2014-0054R03-CR_Various_Protocol_over_GotAPI_1

Document Identifier	Date	Sections	Description
	04 Jan 2015	Various	Updates for agreed CRs: OMA-CD-GotAPI-2014-0057-CR_EventSource OMA-CD-GotAPI-2014-0058-CR_Editorial_in_Introduction OMA-CD-GotAPI-2014-0059-CR_Terminology_Clean_Up
	06 Jan 2015	4	Update for agreed CR: OMA-CD-GotAPI-2015-0001-CR_ER_Intro_Diagram OMA-CD-GotAPI-2015-0002-CR_getNetworkServices
Candidate Version OMA-ER-GotAPI-V1_0	10 Feb 2015	n/a	Status changed to Candidate by TP TP Ref # OMA-TP-2015-0035R02-INP_GotAPI_V1_0_ERP_and_ETR_for_Candidate_approval

Appendix B. Call Flows (Informative)

This is a placeholder to be populated, as required.

Appendix C. GotAPI Enabler Deployment Considerations

This is a placeholder, to be populated as required.

Appendix D. Plug-In Discovery Mechanisms for Android

In order for the GotAPI Server to discover the installed Extension Plug-Ins, there are at least the following two mechanisms available on the Android platform.

1. **GotAPI Server initiated search mechanism**

   When the GotAPI Server is invoked, the GotAPI Server searches installed Extension Plug-Ins using Android specific methods.

   (1) Android supports the getInstalledApplications() method of the PackageManager class, which provides the list of the installed applications. [ANDROID_INSTALLED_APPS]

   (2) When the GotAPI Server gets the list of the installed applications on the Android device, the GotAPI Server can find Extension Plug-In applications from the list. To identify which are the Extension Plug-Ins, the GotAPI Server can read the AndroidManifest.xml of each application, and determine if the application is an Extension Plug-In or not. [ANDROID_APP_MANIFEST]

   Note: This specification does not define the way how to determine if an application is an Extension Plug-In or not. It is left to implementers of the GotAPI Server. Implementers of Extension Plug-Ins must follow the rules that are defined by each implementer of the GotAPI Server.

   (3) The GotAPI Server must keep the list of the installed Extension Plug-In up-to-date during the GotAPI Server is running. To do so, the GotAPI Server must keep receiving events fired when new native applications are installed. Android supports the Broadcast Intent whose Action is "android.intent.action.PACKAGE_ADDED". [ANDROID_PACKAGE_ADDED]

   (4) If the GotAPI Server adopts this mechanism, the GotAPI doesn't need to store the list in a persistent storage, because the GotAPI Server is able to create a complete list of the installed Extension Plug-Ins whenever the GotAPI Server is invoked and keep the list up-to-date during the GotAPI Server is running.

   [References]

   [ANDROID_INSTALLED_APPS]

   http://developer.android.com/reference/android/content/pm/PackageManager.html#getInstalledApplications(int)

   [ANDROID_PACKAGE_ADDED]

   http://developer.android.com/reference/android/content/Intent.html#ACTION_PACKAGE_ADDED

   [ANDROID_APP_MANIFEST]

   http://developer.android.com/guide/topics/manifest/manifest-intro.html

2. **Plug-In initiated registration mechanism**

   (1) After an Extension Plug-In is installed and invoked, the Extension Plug-In invokes the GotAPI Server and sends a request for registration to the GotAPI Server. For Android, the Extension Plug-In uses Explicit Intents to talk to the GotAPI Server. The Extension Plug-In must know the package name of the GotAPI Server application and the provider of the GotAPI Server implementation must ensure that the package name of the GotAPI Server implementation is hardcoded in the Extension Plug-In.

   (2) The GotAPI Server polls all the registered Extension Plug-Ins periodically in order to determine if each Extension Plug-In is still installed.

   (3) If the GotAPI Server adopts this mechanism, the GotAPI has to store the list in a persistent storage and keep it up-to-date, because the GotAPI Server doesn't know the complete list when the GotAPI Server is invoked.