



BSI Standards Publication

Open Data Communication in Building Automation, Controls and Building Management — Control Network Protocol

Part 10: Web Services for Control Networking Protocol Specification

National foreword

This British Standard is the UK implementation of EN 14908-10:2025.

The UK participation in its preparation was entrusted to Technical Committee RHE/16, Performance requirements for control systems.

A list of organizations represented on this committee can be obtained on request to its committee manager.

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

© The British Standards Institution 2025
Published by BSI Standards Limited 2025

ISBN 978 0 539 25945 2

ICS 35.240.67; 91.140.01; 97.120

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 April 2025.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------

English Version

Open Data Communication in Building Automation,
Controls and Building Management - Control Network
Protocol - Part 10: Web Services for Control Networking
Protocol Specification

Communication de données ouverte dans le domaine
de l'automatisation, du contrôle et de la gestion des
bâtiments - Protocole de contrôle réseau - Partie 10 :
Services Web pour la spécification du protocole de
contrôle réseau

Firmenneutrale Datenkommunikation für die
Gebäudeautomation und Gebäudemanagement -
Gebäude-Netzwerk-Protokoll - Teil 10: Spezifikation
der Webdienste für das Kontrollnetzwerkprotokoll

This European Standard was approved by CEN on 17 February 2025.

CEN members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration. Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the CEN-CENELEC Management Centre or to any CEN member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CEN member into its own language and notified to the CEN-CENELEC Management Centre has the same status as the official versions.

CEN members are the national standards bodies of Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Republic of North Macedonia, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Türkiye and United Kingdom.



EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels

Contents

Page

European foreword	9
Introduction	10
1 Scope.....	12
2 Normative references.....	12
3 Terms and definitions	12
4 IAP API FUNDAMENTALS.....	13
4.1 IAP API Overview	13
4.2 IAP/MQ Fundamentals.....	14
4.2.1 IAP/MQ Overview	14
4.2.2 MQTT in IAP	14
4.2.3 Supported MQTT Version	14
4.2.4 IAP/MQ Topics	15
4.2.5 Persistent Clients vs Transient Clients	15
4.2.6 Client Expiration	15
4.2.7 Message Payload	15
4.2.8 CMS Connectivity.....	16
4.2.9 An IAP Site	16
4.3 Partial Object Assignment	16
4.3.1 Definition	16
4.3.2 Edge Servers and Partial Object Assignment.....	17
4.3.3 Timestamps in IAP.....	18
4.3.4 Most Recently Used (MRU) Timestamp	18
4.4 IAP/MQ Topic Syntax.....	19
4.4.1 Syntax elements.....	19
4.4.2 Device Syntax.....	23
4.4.3 Wildcards.....	25
4.5 IAP/REST Fundamentals	25
4.5.1 REST Overview.....	25
4.5.2 REST in IAP	26
4.5.3 Making IAP/REST Calls	26
4.6 IAP/REST Syntax	26
4.7 Queries and Parameters.....	27
4.8 Path Parameters.....	29
4.9 Query Parameters.....	30
4.10 IAP/WS Fundamentals.....	31
4.11 Data Log WebSocket Queries and Parameters.....	32
4.11.1 Data Log WebSocket Overview	32
4.11.2 Key.....	32
4.11.3 Operator	32
4.11.4 Value	32
4.11.5 Regular Expression Encoding.....	32
5 IAP/MQ API	33
5.1 Overview	33
5.2 About	33
5.2.1 Definition	33

5.2.2	Example.....	34
5.2.3	Properties.....	35
5.3	Alarm.....	39
5.3.1	Definition.....	39
5.3.2	Example.....	39
5.3.3	Alarm Configuration.....	41
5.3.4	Alarm Status	41
5.3.5	Examples.....	41
5.3.6	Alarm Properties	41
5.3.7	Alarm Actions.....	43
5.3.8	Alarm Conditions	44
5.3.9	Manage Alarms	47
5.4	Connections	47
5.4.1	Definition.....	47
5.4.2	Connection Implementation.....	47
5.4.3	Connection with Datapoint Presets	48
5.4.4	Connection Objects.....	49
5.4.5	Connection Do Actions.....	54
5.4.6	Connection Types	57
5.4.7	Type Translation.....	58
5.5	Data Type Definitions.....	62
5.5.1	Definition.....	62
5.5.2	Data Type References.....	62
5.5.3	About IAP Data Types.....	63
5.5.4	Data Type General Information.....	63
5.5.5	Data Type Presentation and Formatting.....	65
5.6	Device.....	66
5.6.1	Definition.....	66
5.6.2	Device Objects.....	66
5.6.3	Device Configuration	67
5.6.4	Device Do Action	68
5.6.5	Device Interface	78
5.6.6	Device Status	79
5.6.7	Dynamic Interfaces	81
5.6.8	Implementation Detail Object.....	83
5.7	Discovery	85
5.8	Discovery and Provisioning Details	87
5.8.1	Device Discovery.....	87
5.8.2	Segment Discovery	91
5.8.3	Segment Provisioning	91
5.8.4	Segment Discovery Message	93
5.9	Events.....	95
5.9.1	Definition.....	95
5.9.2	Data Events	97
5.9.3	Join Events.....	97
5.9.4	Tracing.....	98
5.9.5	Event Object Properties	98
5.9.6	Created Event.....	101
5.9.7	Deleted Event	103
5.9.8	Updated Event.....	104
5.10	Group.....	105
5.10.1	Definition.....	105
5.10.2	Group Characteristics	106

5.10.3 Group Examples.....	106
5.10.4 IAP Groups.....	108
5.10.5 self.add.....	110
5.10.6 self.create.....	111
5.10.7 self.delete.....	112
5.10.8 self.provision.....	112
5.10.9 self.remove.....	113
5.11 Handle Allocation	114
5.11.1 Definition	114
5.11.2 Handle Allocation Service	115
5.11.3 Handle Request.....	115
5.11.4 Handle Response.....	116
5.11.5 Example.....	116
5.12 Interface Blocks.....	116
5.12.1 Definition	116
5.12.2 Interface Topics.....	117
5.12.3 Block Object Properties.....	117
5.12.4 Datapoint Objects	119
5.12.5 Datapoint Presets	125
5.12.6 Datapoint Localization.....	133
5.12.7 Monitor Preference Object	137
5.12.8 Working with Datapoint Values	141
5.12.9 Working with Unions.....	145
5.13 License Services.....	146
5.13.1 License Service Overview.....	146
5.13.2 Actions	146
5.13.3 Capacity	153
5.13.4 Cloud.....	154
5.13.5 Configuration.....	156
5.13.6 License Management	157
5.13.7 Status.....	159
5.14 Load Do Action	161
5.14.1 Definition	161
5.14.2 Outer Image File	161
5.14.3 Load Procedure	162
5.14.4 The Load Object.....	163
5.14.5 Inner Image File Example	163
5.14.6 Load Action and Manifest Properties	164
5.15 On-Demand Monitoring.....	169
5.15.1 Definition	169
5.15.2 On-Demand Datapoint Monitoring Request.....	170
5.15.3 On-Demand Datapoint Monitoring Response.....	171
5.15.4 On-Demand Monitoring Service Collaboration.....	171
5.15.5 Item Poll Request.....	172
5.15.6 Item Poll Response.....	172
5.16 Query	173
5.16.1 Definition	173
5.16.2 Query Request Properties	173
5.16.3 Query Response.....	175
5.16.4 Filtering.....	176
5.17 Schedule Services.....	177
5.17.1 Schedule Services Overview.....	177
5.17.2 Schedule Algorithm.....	178

5.17.3	Schedule Object.....	179
5.17.4	Calendar Object.....	182
5.17.5	Calendar Status Object.....	183
5.17.6	Dates in Schedules and Calendars.....	185
5.17.7	Specifying Dates.....	185
5.17.8	Weekly Schedule.....	189
5.17.9	Exception Schedule.....	190
5.18	Segment Configuration.....	192
5.18.1	Definition.....	192
5.18.2	Segment Configuration Properties.....	193
5.18.3	Location Object.....	197
5.19	Segment Do Actions.....	199
5.19.1	Definition.....	199
5.19.2	Action Objects.....	199
5.20	Segment Status.....	206
5.20.1	Definition.....	206
5.20.2	Example.....	206
5.20.3	Properties.....	206
6	IAP/REST API.....	207
6.1	IAP/REST Overview.....	207
6.2	Access.....	207
6.2.1	URI Definition.....	207
6.2.2	Query Parameters.....	208
6.3	Alarms.....	208
6.3.1	URI Definition.....	208
6.3.2	Query Parameters.....	222
6.4	Authentication (Login/Logout).....	222
6.4.1	URI Definition.....	222
6.4.2	Query Parameters.....	224
6.4.3	Example.....	224
6.5	Capabilities.....	225
6.5.1	URI Definition.....	225
6.5.2	Query Parameters.....	227
6.6	Connection.....	228
6.6.1	URI Definition.....	228
6.6.2	Parameters.....	228
6.6.3	Query Parameters.....	228
6.6.4	Examples.....	228
6.7	Context.....	228
6.7.1	Context Overview.....	228
6.7.2	Context - Contextual Entity Relationship.....	229
6.7.3	Device Assignment.....	229
6.7.4	URI Definition.....	229
6.8	Customers.....	236
6.8.1	URI Definition.....	236
6.8.2	Query Parameters.....	239
6.8.3	Examples.....	239
6.9	Datapoint Categories.....	240
6.9.1	URI Definition.....	240
6.9.2	Queries.....	241
6.10	Datapoint Default Values.....	243
6.10.1	URI Definition.....	243

6.10.2	Queries.....	245
6.10.3	Example	247
6.11	Datapoint Favorites	247
6.11.1	URI Definition.....	247
6.11.2	Query Parameters.....	250
6.12	Datapoint Logs	250
6.12.1	Overview	250
6.12.2	URI Definition.....	251
6.12.3	Queries.....	256
6.12.4	Logging Configuration	256
6.12.5	Datapoint Log Value.....	256
6.13	Datapoint Monitor	259
6.13.1	Overview	259
6.13.2	URI Definition.....	259
6.13.3	Queries.....	262
6.14	Datapoint Overrides	264
6.14.1	Overview	264
6.14.2	URI Definition.....	264
6.14.3	Queries.....	266
6.15	Datapoint Override State	268
6.15.1	URI Definition.....	268
6.15.2	Queries.....	269
6.16	Datapoint Override Value	270
6.16.1	Overview	270
6.16.2	IAP/MQ Mapping.....	270
6.16.3	URI Definition.....	271
6.16.4	Queries.....	272
6.16.5	Example	273
6.17	Datapoint Properties	273
6.17.1	URI Definition.....	273
6.17.2	Queries.....	274
6.17.3	Examples	276
6.18	Datapoint Tags.....	278
6.18.1	Overview	278
6.18.2	URI Definition.....	278
6.18.3	Queries.....	279
6.19	Datapoint Type	281
6.19.1	Overview	281
6.19.2	IAP/MQ Mapping.....	281
6.19.3	URI Definition.....	281
6.19.4	Queries.....	284
6.20	Datapoint Value	285
6.20.1	URI Definition.....	285
6.20.2	Queries.....	286
6.20.3	Examples	288
6.21	Device Capabilities	290
6.21.1	URI Definition.....	290
6.21.2	Query Parameters.....	295
6.22	Device Types.....	295
6.23	Devices.....	307
6.23.1	URI Definition.....	307
6.23.2	Query Parameters.....	337
6.23.3	Example	338

6.24	Diagnostics	341
6.24.1	URI Definition	341
6.24.2	Query Parameters	341
6.25	DLA Files	341
6.25.1	URI Definition	341
6.25.2	Queries	345
6.26	Drivers	348
6.27	Floorplans	351
6.27.1	URI Definition	351
6.27.2	Query Parameters	356
6.28	Geozone	356
6.28.1	URI Definition	356
6.28.2	Query Parameters	365
6.29	Groups	365
6.29.1	URI Definition	365
6.29.2	Group Queries	374
6.30	Log Destination	377
6.30.1	URI Definition	377
6.30.2	Query Parameters	377
6.31	Log Events	377
6.31.1	Log Events Overview	377
6.31.2	URI Definition	378
6.31.3	Query Parameters	381
6.31.4	Event Log Path Parameters	381
6.31.5	Examples	383
6.32	Log State	385
6.32.1	URI Definition	385
6.32.2	Query Parameters	386
6.33	Log Status	386
6.33.1	URI Definition	386
6.33.2	Query Parameters	387
6.33.3	Example	387
6.34	On-Demand	387
6.35	Permissions	388
6.35.1	URI Definition	388
6.35.2	Query Parameters	389
6.35.3	Example Response	389
6.36	Presets	389
6.36.1	URI Definition	389
6.36.2	Query Parameters	391
6.37	Protocols	391
6.38	Reports	395
6.38.1	URI Definition	395
6.38.2	Query Parameters	405
6.39	Roles	405
6.39.1	URI Definition	405
6.39.2	Query Parameters	406
6.40	Schedule Service	406
6.40.1	URI Definition	406
6.40.2	Schedule Service Query and Path Parameters	413
6.40.3	Schedule Path Parameters	413
6.40.4	Schedule Service Examples	414
6.41	SMTP Configuration	418

6.41.1	URI Definition.....	418
6.41.2	Query Parameters.....	419
6.42	Transformation Service.....	420
6.43	Trend Log.....	421
6.44	Users.....	424
6.44.1	URI Definition.....	424
6.44.2	Query Parameters.....	429
6.44.3	Examples.....	429
6.45	Version.....	430
6.45.1	URI Definition.....	430
6.45.2	Query Parameters.....	431
6.45.3	Example.....	431
6.46	Zones.....	432
6.46.1	URI Definition.....	432
6.46.2	Query Parameters.....	437
6.46.3	Zone Path Parameters.....	437
7	IAP/WS API.....	438
7.1	IAP/WS Overview.....	438
7.2	General Purpose WebSocket.....	438
7.3	Data Log WebSocket.....	438
7.4	Examples.....	439
7.4.1	Example 1.....	439
7.4.2	Example 2.....	440
	Bibliography.....	441

European foreword

This document (EN 14908-10:2025) has been prepared by Technical Committee CEN/TC 247 "*Building Automation, Controls and Building Management*", the secretariat of which is held by SNV.

This European Standard shall be given the status of a national standard, either by publication of an identical text or by endorsement, at the latest by October 2025, and conflicting national standards shall be withdrawn at the latest by October 2025.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CEN shall not be held responsible for identifying any or all such patent rights.

This document is part of the EN 14908 series, which consists of the following parts:

- EN 14908-1, *Open Data Communication in Building Automation, Controls and Building Management — Control Network Protocol — Part 1: Protocol Stack*
- EN 14908-2, *Open Data Communication in Building Automation, Controls and Building Management — Control Network Protocol — Part 2: Twisted Pair Communication*
- EN 14908-3, *Open Data Communication in Building Automation, Controls and Building Management — Control Network Protocol — Part 3: Power Line Channel Specification*
- EN 14908-4, *Open Data Communication in Building Automation, Controls and Building Management — Control Network Protocol — Part 4: IP Communication*
- EN 14908-5, *Open Data Communication in Building Automation, Controls and Building Management Implementation Guideline — Control Network Protocol — Part 5: Implementation*
- EN 14908-6, *Open Data Communication in Building Automation, Controls and Building Management — Control Network Protocol — Part 6: Application Elements*
- EN 14908-7, *Open communication in building automation, controls and building management — Control Network Protocol — Part 7: Communication via internet protocols*
- EN 14908-8, *Open Data Communication in Building Automation, Controls and Building Management — Control Network Protocol — Part 8: Communication using Broadband over Power Line Networks – with internet protocols*
- EN 14908-9, *Open Data Communication in Building Automation, Controls and Building Management — Control Network Protocol — Part 9: Wireless Communication in ISM bands*

Any feedback and questions on this document should be directed to the users' national standards body. A complete listing of these bodies can be found on the CEN website.

According to the CEN-CENELEC Internal Regulations, the national standards organisations of the following countries are bound to implement this European Standard: Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Republic of North Macedonia, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Türkiye and the United Kingdom.

Introduction

This document specifies the Internet of Thing IoT Access Protocol (IAP) as a standard web services application protocol based on MQTT (Message Queuing Telemetry Transport), and REST (Representational State Transfer) and WebSockets over HTTP or HTTPS transport protocols.

MQTT, REST, and WebSockets are widely used today for the IoT, but they are low-level protocols with no standardization in the application data and service requests sent using the protocols. IAP delivers that standardization for control and automation networks.

IAP provides secure and open access to the most complex building automation and control systems, enabling integration of existing controls and automation systems with emerging analytics and AI applications that transform operations from a closed single-protocol loop to an intelligent multi-protocol system.

EN 14908-10 IoT Access Protocol (IAP) web services are an open and extensible standard designed for residential, commercial, and industrial control and automation applications using the EN 14908-1 control network protocol and related protocols to provision and manage IoT devices, to access and update data from the devices, and to aggregate data from diverse devices and protocols for delivery to external applications and services.

IAP web services are typically implemented on a central gateway or edge server that communicates with multiple sensor, actuator, and controller edge devices using one or more drivers such as EN 14908-1, and also interfaces with one or more enterprise and cloud services or applications. The following figure 1 illustrates a gateway or edge server that uses IAP to communicate with edge devices and cloud services.

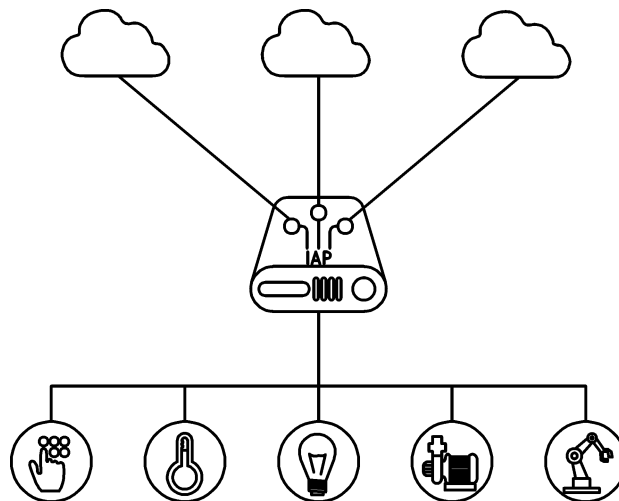


Figure 1 — Edge Server

This document is for software developers and web-page authors. It documents three APIs: *IAP/MQ*, which uses MQTT (Message Queuing Telemetry Transport) as the transport protocol, *IAP/REST* (*REpresentational State Transfer*), which uses REST over HTTP or HTTPS, and *IAP/WS* (WebSockets), which uses WebSockets over HTTP or HTTPS. These APIs are typically implemented on a gateway or edge server that interfaces with edge devices such as energy meters, motion sensors, temperature sensors, HVAC controllers, water heaters/boilers, lighting controllers, refrigeration controllers, variable-frequency drives, motion controllers, and switches. The gateway or edge server typically communicates with the edge devices using a variety of building controls communication protocols including EN 14908-1 control network protocol, BACnet (ISO 16484-5), Modbus, EnOcean, and LoRaWAN.

This document consists of the following four main sections:

- IAP API Fundamentals;
- IAP/MQ API;
- IAP/REST API;
- IAP/WS API.

1 Scope

This document specifies an open and extensible standard for residential, commercial, and industrial control and automation applications using the EN 14908-1 control network protocol and related protocols (EN 14908-2 to EN 14908-9) to provision and manage IoT devices, to access and update data from the devices, and to aggregate data from diverse devices and protocols for delivery to external applications and services.

The web services as specified in this document are typically implemented on a central gateway or edge server that communicates with multiple sensor, actuator, and controller edge devices using one or more drivers, such as EN 14908-1, and typically also interfaces with one or more enterprise and cloud services or applications.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

EN 14908-5, *Open Data Communication in Building Automation, Controls and Building Management Implementation Guideline — Control Network Protocol — Part 5: Implementation*

EN 14908-6, *Open Data Communication in Building Automation, Controls and Building Management Implementation Guideline — Control Network Protocol — Part 6: Application elements*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <https://www.iso.org/obp/>
- IEC Electropedia: available at <https://www.electropedia.org/>

3.1

Message Queuing Telemetry Transport

MQTT

lightweight, publish-subscribe, machine to machine network protocol for Message queue/Message queuing service [1]

3.2

REpresentational State Transfer

REST

software architectural style that describes a uniform interface between physically separate components, often across the Internet in a client-server architecture

3.3

Internet Access Protocol

IAP

open and extensible protocol designed for the Internet of Things to provision and manage devices, and to access and update data from the devices

4 IAP API FUNDAMENTALS

4.1 IAP API Overview

The IoT Access Protocol (IAP) is an open and extensible protocol designed for the IoT to provision and manage devices based on the EN 14908-x protocol suite, to access and update data from the devices, and to aggregate data from diverse devices for delivery to external applications and services such as data analytics and machine learning services. IAP may also be used to support other non- EN 14908-x devices through the methodologies defined in this document.

IAP provides an information model and a data model for the IoT that enables the construction of digital twins of the physical IoT devices in a system. IAP enables these digital twins to be organized into contexts. This organization simplifies integration of the IoT devices to interact with each other and with external services for system monitoring, supervisory control, data analytics, and machine learning.

System monitoring, control, data analytics, machine learning, and AI systems require knowledge of the meaning of the data received from or sent to IoT devices, as well as providing a means to specify the relationships of the data on multiple devices. For example, an IoT device may have a temperature sensor. If the sensor data is exposed as a number with no additional semantics by an edge server, it is difficult for a monitoring and control application to use the temperature value without additional information from the device manufacturer, and it is difficult for an analytics or machine learning application to analyze the value over time without the additional information. A first level of meaning is provided by IAP data types which specify data encoding, size, scaling, units, and usage for a datapoint. Hundreds of standard IAP data types are defined by EN 14908-6, and IAP defines a standard protocol for defining new data types when an appropriate standard data type is not available. For example, the value received from a temperature sensor may be specified using a standard temperature IAP data type that specifies the value is a temperature value expressed in Celsius units.

While an open standard for defining and specifying data types is essential for Industrial IoT systems, it is not sufficient for supporting an open model for IoT system monitoring, control, analytics, and machine learning applications. For example, a space comfort controller in a building may require multiple temperature sensors to monitor space temperature, supply air temperature, and outdoor air temperature. The data type for these sensors is not sufficient to monitor, control, or analyze the operation of the space comfort controller since all three values may be reported as Celsius temperature values. An open method to differentiate the three types of temperature values is also essential for Industrial IoT systems. IAP provides an open method to create a digital twin of a logical subsystem in an IoT device. The subsystem digital twins are exposed as IAP blocks, where a block is a collection of datapoints with specified meaning. IAP blocks are defined by IAP profiles. More than one hundred standard IAP profiles are defined by EN 14908-6. These profiles were developed by industry-specific Industrial IoT working groups to ensure the profiles met the requirements of the specific applications enabled by them.

For example, the space comfort controller may be modelled as a block defined by the IAP standard space comfort controller profile. The IAP space comfort controller profile defines standard inputs for space temperature, source air temperature, and outdoor air temperature. Monitoring, control, and analytics applications may differentiate the three temperature values based on the standard model of the digital twin specified by the IAP profile.

To simplify system integration, IAP extends the digital twin model to the full device. A device is a physical entity with a communications interface. There are two types of devices--edge devices and edge servers. Edge devices typically have one or more sensors, actuators, and controllers. The digital twin for a device contains one or more blocks, each with one or more datapoints. Edge servers provide a unified interface to one or more edge devices. IAP provides device-level operations such as provisioning, testing, and replacement.

To reduce the time required to integrate a system, IAP adds context information for each device, and includes services to leverage the context information when integrating a system. A context may be a

region, campus, building, floor, room, or other area that contains a device. Contexts are organized hierarchically so that a campus contains buildings and areas, a building contains floors, and a floor contains rooms and areas. IAP operations may be specified by templates that may be associated with contexts. For example, IAP supports connection templates that specify how specific datapoints in a context may be connected to other datapoints in the context. This enables the integrator to connect all the temperature sensors in an area to a controller in the area that calculates the average temperature reported by the sensors. The integrator may apply hundreds of area contexts to the connection template to create hundreds of independent connections for each of the areas as defined by the connection template, all with one simple operation.

This standard defines three APIs: IAP/MQ, which uses MQTT (Message Queuing Telemetry Transport) as the transport protocol, IAP/REST (REpresentational State Transfer), which uses HTTP and HTTPS, and IAP/WS API, which uses WebSockets. These APIs are defined in the following sections:

- IAP/MQ Fundamentals;
- IAP/MQ Topic Syntax;
- IAP/REST Fundamentals;
- IAP/REST Syntax;
- Queries and Parameters;
- IAP/WS Fundamentals;
- Data Log WebSocket Queries and Parameters.

4.2 IAP/MQ Fundamentals

4.2.1 IAP/MQ Overview

IAP/MQ is the IAP API based on MQTT version 3.1.1. MQTT is an ISO standard (ISO/IEC PRF 20922) publish-subscribe-based messaging protocol. It works on top of the TCP/IP protocol. It is a lightweight protocol that is suitable for implementing applications, services, and drivers on a wide range of devices and servers including small code footprint edge devices, edge servers, and cloud hosts. IAP/MQ is used to connect services and drivers within an edge server and is used to connect edge servers to each other and cloud applications and services

4.2.2 MQTT in IAP

The IoT Access Protocol (IAP) uses an MQTT message broker such as Mosquitto or ActiveMQ, known as the IAP message broker. An IAP system may include central management software (CMS) that provides a user interface for managing the system operation. It includes an IAP message broker, known as the CMS message broker or the CMS broker. IAP edge servers typically include a local message broker, sometimes referred to as a segment message broker.

IAP does not require a specific broker, specific MQTT client library, or implementation. The chosen broker and clients shall support the appropriate versions of MQTT as indicated in the section 4.2.3 Supported MQTT Version, below.

4.2.3 Supported MQTT Version

All MQTT components of an IAP system shall support MQTT protocol version 3.1.1. MQTT version 3.1 or earlier shall not be supported.

4.2.4 IAP/MQ Topics

All IAP topics use a specific format as described in **4.4 IAP/MQ Topic Syntax**.

4.2.5 Persistent Clients vs Transient Clients

An IAP client shall implement a persistent or a transient MQTT client. Most IAP clients are persistent clients.

For a persistent client, the broker maintains events while the client is unavailable, whereas events are lost to a transient client while the transient client is disconnected.

For example, a client subscribes to a certain set of topics in the IAP/MQ event channel. Then this client disconnects, and later re-connections. Publications made to the subscribed topics during the client's downtime will be delivered to the client if it is a persistent client. A transient client will not see publications that occurred during its downtime.

But a side effect of a persistent client that never reconnects, is that the client may accumulate many messages that the broker keeps on that client's behalf. This issue is addressed in **4.2.6 Client Expiration**.

MQTT clients implement persistence using a constant and unique client ID, and the MQTT property `cleanSession=False` when connecting to the IAP message broker. Persistent clients shall specify a unique MQTT client ID and shall re-use the same client ID in every session. An MQTT client identifier is between 1 and 23 characters long, UTF-8 encoded, and shall consist of letters a-z, A-Z, digits 0-9. A client ID may be constructed from a combination of the MACID and a sequence number, or completely at random, and is often allocated by agreement among components.

Temporary tools, such as mobile servicing and diagnostic tools, may connect as transient clients, using a unique client ID and the MQTT property `cleanSession=True` when connecting. This includes tools that do not expect to re-connect to the same system regularly or do not catch-up with the history of events and changes that occurred between connections.

Transient clients shall specify a unique MQTT client ID but may allocate a new client ID for every session, if necessary. A transient client ID begins with a unique identifier, which has the same format as a persistent client ID.

4.2.6 Client Expiration

A typical IAP message broker supports persistent client expiration at an interval of 2 hours. When a persistent client is disconnected for a duration exceeding this interval, the persistent client's session is automatically cleared to free resources in the broker.

4.2.7 Message Payload

The payload for IAP/MQ messages shall be JSON encoded. However, for readability, many of the examples in this document are shown in Javascript format, which eliminates the use double-quotes around keys, as shown in the following examples.

Javascript

```
{
  value: 1234,
  name: "John Denver"
}
```

JSON

```
{
  "value": 1234,
  "name": "John Denver"
}
```

4.2.8 CMS Connectivity

Connectivity between the CMS broker and individual segment brokers is not always present. Some systems may be designed to connect on a schedule and for limited duration only, other systems may simply use unreliable connections. Tools shall not assume that the edge server is present at the time the tool makes a change. Tools may detect the presence or absence of connectivity and indicate the state in the user interface. However, a tool shall not expect to receive immediate confirmation for a change through the feedback channel.

IAP tools shall publish changes to the request channel as necessary and shall respond to updates received through the feedback channel as these updates become available. Updates received from the feedback channel shall report the current system state at the time of the update, which may differ from the state that was requested with the request channel. This difference may be caused by an error processing the request or by a state change initiated by another client to the same system.

For example, a user interface tool may relay a human operator's requests to turn a light on at 50 % brightness. When the tool makes that request, it may reflect the new state of on at 50 % as a tentative state, as yet unconfirmed. When the corresponding feedback is received, the tool may then update its user interface to show the confirmed state, which may be on at 50 % or any other state.

4.2.9 An IAP Site

An IAP-based system consists of one or more sites. A site is typically a physical location, often providing shared Internet access to components within the site. Each site in an IAP-based system includes one or more segments, with one edge server controlling each segment.

4.3 Partial Object Assignment**4.3.1 Definition**

IAP/MQ supports partial object assignment where appropriate. For example, a write request may request to a change affecting one datapoint within a device's interface block, leaving other properties and datapoints within the same block unchanged.

Some objects do not support partial write operations. For example, a datapoint value is considered an atomic value as described in **5.12.8.8**

Values are Atomic and shall only be modified whole. For information on datapoint values, device interface blocks, or device objects—see **5.12 Interface Blocks**.

Where partial assignments are supported, empty assignments are also supported. An empty assignment is essentially an assignment without a value change, triggering the renewed publication of the corresponding feedback item. The edge server decides whether such an empty assignment also affects the edge device.

Field and object assignments shall be compatible, according to the following rules:

- Some properties may be read only. Tools may write these properties if the value does not change.

- Some properties may be write only. Such a property is used explicitly in an assignment request within the request channel but is not explicitly reported within the feedback channel.
- Some objects and operations require a minimum set of properties. For example, the device `do` topic accepts an `action` object. This object always requires an `action` property, which specifies the desired action (provision, delete, etc.), and the selected action typically requires further mandatory arguments.
- These operations fail unless the minimum required details are provided. Failure is reported with an error diagnostic.
- Some clients may be unaware of all properties received with an object. Clients shall not act on object types and topics they don't recognize, but a user interface client may list this data in some generic representation for expert users.
- Privileged clients may add properties to standard objects. Such clients shall include this property when publishing and all clients shall ignore it unless it is known and understood.
- The feedback channel shall always report the complete object, even though objects support partial assignment in the request channel.

4.3.2 Edge Servers and Partial Object Assignment

Wherever partial object assignment is supported, edge servers shall also support request topics that reach into an object. For example, the `glp/0/bw/rq/dev/lon/1/if/light/0` topic addresses a particular interface block named "light" within a device. This request topic accepts assignment of a complete or a partial light interface block object.

The following shorthand notation illustrates the assignment of a partial object to this topic, affecting only the `desc` property within that block:

```
glp/0/bw/rq/dev/lon/1/if/light/0 ::= { desc: "an example description" }
```

The following example yields the same result, but this time including the partial object in the topic:

```
glp/0/bw/rq/dev/lon/1/if/light/0/desc ::= "an example description"
```

Topics may address items at any depth within the addressed object. For example, the `light` interface block may implement an `sp` datapoint. The Datapoint Object (see **5.12.4 Datapoint Objects**) supports a `monitor` object to control datapoint monitoring, which in turn supports a `rate` property (see **5.12.7 Monitor Preference Object**) to specify the frequency at which a new actual value may be obtained.

The following alternative forms all produce the same result (setting the rate to 17) using partial object assignments:

```
glp/0/bw/rq/dev/lon/1/if/light/0 ::= { sp: { monitor: { rate: 17 } } }
glp/0/bw/rq/dev/lon/1/if/light/0/sp ::= { monitor: { rate: 17 } }
glp/0/bw/rq/dev/lon/1/if/light/0/sp/monitor ::= { rate: 17 }
glp/0/bw/rq/dev/lon/1/if/light/0/sp/monitor/rate ::= 17
```

4.3.3 Timestamps in IAP

The standard timestamp format in IAP is the MRU timestamp (described below). In addition to this timestamp, data events may also use the `ts` format. The `ts` timestamp is a simplified extended ISO 8601 timestamp in the `YYYY-MM-DDTHH:mm:ss.sssZ` format, always in UTC.

The MRU timestamp (describe below) uses the same format as the `ts` format:

`YYYY-MM-DD HH:mm:ss.sss ZZZ`

Where:

- `YYYY` is the four-digit year,
- `MM` is the two-digit month,
- `DD` is the two-digit day of the month,
- `HH` is the hour of the day, always using the 24-hour clock,
- `mm` the number of minutes in the hour,
- `ss` the number of seconds in the minute,
- `sss` the number of milliseconds within the second,
- `ZZZ` is the name of the time zone for wall time or UTC.

For information on configuring the time zone, see the `time` object described in **5.18 Segment Configuration**.

All numbers in a timestamp are padded with 0 to the left to meet the specified number of digits, if necessary. Number groups are separated with a single hyphen, colon, period, or a single space character.

EXAMPLES

```
2018-11-30 16:33:05.257 BST
2019-01-09 15:02:18.321 UTC
```

4.3.4 Most Recently Used (MRU) Timestamp

All messages published in the feedback channel shall include a most recently used (`mru`) timestamp, using the standard IAP timestamp format. Messages posted in other channels may include an MRU timestamp. All MRU timestamps are UTC timestamps.

Most recipients of messages ignore the MRU, but recipients may use the MRU, for example, to prune stale objects which may have resulted from the deletion of items. MRU timestamps are always ignored when assigned through the request channel.

Deletion of resources generally leaves one item behind, a status object which informs clients that reconnect in the future that the resource in question has been deleted. These items are typically pruned on a low frequency.

4.4 IAP/MQ Topic Syntax

4.4.1 Syntax elements

The syntax for topics is as follows:

```
glp/{IAP_Ver}/{SID}/{Channel}/{Resource_Type}[/{Resource_Detail}][/{Property_Path}]
```

EXAMPLE

```
glp/0/s1/fb/dev/lon/2/if/light/1
```

Table 1 describes the IAP/MQ syntax levels.

Table 1 — IAP/MQ Topic Syntax

Level	Description
glp	IAP topics always begin with <code>glp</code> , where <code>glp</code> is a constant keyword. A topic shall not begin with a slash.
<i>IAP_Ver</i>	The topic shall specify the IAP version. This standard describes version 0 (zero). All IAP implementations shall support version 0 of the protocol. The number provided here indicates the highest protocol version supported. IAP supports all versions up to and including the maximum protocol version indicated with <code>glp/0/{SID}/about/version</code> .
<i>SID</i>	The SID is the segment identifier, which is the logical address of the device that controls a segment. In IAP, that device is either an edge server, or, an IAP-native edge device that is the only device on that segment. Such an edge device may implement other object types, such as alarms or connections for itself, and uses the standard topic layout. An SID consists of the letters A-Z, a-z, digits 0-9, and special characters period (.), hyphen (-), and underscore (_). Special characters shall not be used as the first character of a SID. A valid SID matches the following regular expression: <pre>[A-Za-z0-9][A-Za-z0-9._-]*</pre> The SID is assigned to the edge server when the segment is provisioned. A regular SID has the format shown above. A different address is used in place of the SID in certain specific circumstances, such as before the segment is provisioned, or to indicate a local address for the edge server. For information about SIDs before an edge server is provisioned, see 5.8.2 Segment Discovery. A local address for the edge server may be used for communication between IAP components within a segment. Topics using "." for the SID are called local topics. Local topics are not relayed to the CMS or other segments through bridges. Topics using local addressing never traverse connections between the CMS and a remote edge server. A period identifier used outside the segment represents a segment-independent address. And like local topics, segment-independent topics are

Level	Description
	not relayed across bridge connections. A period identifier is used for certain services, such as the logging service described in 6.11 Datapoint Logs. <code>glp/0/. /=logger/event</code> The = sign indicates that the topic is addressing a service, instead of an object. Changing a segment identifier requires that the segment be deprovisioned first, then re-provisioned with the new segment identifier. Reserved SIDs: — Beginning an SID with one of the following characters is reserved for use within the edge server: %, &, \$ — The SID shall not begin with # or + to avoid confusion with MQTT topic wildcards. — The SID shall not contain a slash (/).
<i>Channel</i>	IAP groups topics and message handling attributes into directional communication channels. Additional channels may exist within an edge server, but clients shall not react to channel types that they are unaware of. When necessary, clients shall process any unknown channel types verbatim. IAP includes the following channels, with the following identifiers: <code>rq</code> Request Channel. Tools use the request channel to publish messages requesting value or state changes. Messages in the request channel flow from the tool or CMS to the edge server, and within the segment the messages flow to the edge processor or to another service. The request flows to the component responsible for acting on it. All publishing to the request channel use MQTT properties QoS 2, retain=False. <code>fb</code> Feedback Channel. The feedback channel provides the system state from the edge processor or another component within the edge server to all clients. All publishing to the feedback channel uses MQTT properties QoS=1, retain=True. The <code>fb</code> channel is retained so that new clients receive the current system state when they subscribe to the feedback channel. A client may subscribe to the entire feedback channel, or only to selected topics and topic groups within the feedback channel. All feedback channel objects include a timestamp of the most recent update for that object. This is provided in the <code>mru</code> property. <code>ev</code> Event Channel. The event channel provides diagnostics, such as errors, warnings, alarms, and similar information to all clients. All publishing to the event channel use MQTT properties QoS=1, retain=False. A new client will not receive events that occurred in the past, and neither will a disconnected non-persistent client. However, a disconnected persistent MQTT client will receive events missed while it was disconnected.

Level	Description
	A persistent MQTT client is one that connects with the MQTT "cleanSession" attribute set to False. bin Binary Channel. Topics within the binary channel address a binary blob of data. There are no objects or properties within objects in the binary channel, and binary channel topics do not provide any hierarchy within the channel. An IAP/MQ binary channel topic matches the following filter: <pre>glp/0/+bin/+/#</pre> An IAP/MQ binary channel topic typically takes the form: <pre>glp/0/{SID}/bin/{Handle}</pre> Additional topic elements are possible, such as: <pre>glp/0/{SID}/bin/{Handle}/a/b/</pre> Unlike the request, feedback, or event channel, a binary channel topic, such as <code>glp/0/{SID}/bin/{Handle}/a/b</code>, is independent and has no relation to <code>glp/0/{SID}/bin/{Handle}/a</code>. Blobs of data within the binary channel are limited in size and are usually considered short-lived. A use-case is that of embedded transfer of small binary images through IAP/MQ for use with the load action described in 5.14 Load Do Action. IAP/MQ infrastructure components shall support at least three simultaneous publications up to 128 KB, each within the binary channel. Publications exceeding the size limit, or exceeding the maximum number of simultaneous publications, shall be refused with standard MQTT message codes, using no PUBACK with MQTT 3.1.1, or a negative acknowledgment with MQTT 5. Simultaneous publications are those that are published with <code>retain=true</code> and are not yet deleted, plus those published with <code>retain=false</code> and not yet delivered to all subscribers. Items are published into the binary channel using <code>QoS=1</code> or <code>2</code>, <code>retain=true</code>. The consumer of that data, such as the component executing a load action, is expected to remove that data by publishing <code>null</code> with <code>retain=true</code> to the same topic.

Level	Description
<i>Resource_Type</i>	IAP supports a variety of resource types such as devices, groups, schedules, etc. The supported resource types include those in the following list. • about – Segment Information • alarm – Alarms • app – Apps • cfg – Segment Configuration • con – Connections • dev – Devices (see 4.4.2 Device Syntax below) • grp – Groups • res – Resource Data Types • sev – Schedule Services • sts – Segment state • =system – Handle request; the = sign is used to indicate that the topic is addressing a service, instead of an object.
<i>Resource_Detail</i>	The general syntax for the <i>Resource Detail</i> is: <pre>[{Driver_ID}]/ {Handle}/{Object_Path}</pre> Depending on the resource type, <i>Resource Detail</i> may include a <i>Driver_ID</i> (such as for dev resources), and the <i>Object_Path</i> element may be multiple levels deep, depending on the object being specified. Possible topic syntax for a dev Resource and grp Resource is shown below: For a dev resource: <pre>glp/0/{SID}/{Channel}/dev/{Driver_ID}/ {Handle}/{Object_Path}</pre> For a grp resource: <pre>glp/0/{SID}/{Channel}/grp/{Handle}/{Object_Path}</pre> See 4.4.2 Device Syntax below for more syntax information.
<i>Property_Path</i>	Many resources allow topics to address a property within the selected object. This includes nested properties, which are separated with a single forward slash. Consider the following example: <pre>glp/0/s1/rq/dev/lon/a/if/LightCntrl/0/nviLampValue/ value</pre> Where — glp/0/s1 is the segment address; — .../rq addresses the request channel within the segment; — .../dev/lon/a addresses a device object (.../Resource_Type/Driver_ID/Handle);

Level	Description
	— <code>.../if/LightCntl/0</code> addresses an interface block within the device (<code>.../Object_Path</code>); <code>.../nviLampValue/value</code> is a property path within that interface block object.

4.4.2 Device Syntax

`.../dev / {Driver_ID} / {Handle} / {Object_Path} [ / {Property_Path} ]`

Following the `dev` resource element, device topics may include the elements listed in Table 2:

Table 2 — Device Syntax

Topic Element	Description
<i>Driver_ID</i>	Topics separate device objects according to their driver. When required, the driver identifier follows the resource type identifier. Edge servers may support multiple drivers. Each supported protocol shall be identified with a unique driver ID. The following driver IDs are specified by this document – any additional protocols shall use different implementation-specific driver IDs. • bacnet – BACnet (ISO 16484-5) • enocan – EnOcean (ISO/IEC 14543-3-10) • iap – IAP/MQ (this standard) • lon – CNP and CNP/IP (EN 14908-1 and EN 14908-7) • lorawan – LoRaWAN (ITU-T Y. 4480) • modbus – Modbus
<i>Handle</i>	Handles are identifiers of addressable items used with IAP topics. The handle is a unique identifier within its scope. For example, every device object has a unique handle within its driver and within its segment. A handle is a case-sensitive string at least one character long and consists of letters A-Z and a-z, digits 0-9, and special characters period (.), hyphen (-), and underscore (_). A period may be used to separate parts of a handle but shall not be used as the first character. The following characters are reserved, and shall not be used: =, :, & The handle is assigned by the tool that creates the primary item. For example, a tool may create a CNP device and assign handle "1" or "2" as shown in the example below. <pre>glp/0/s1/rq/dev/lon/1/ glp/0/s1/rq/dev/lon/2/</pre>

Topic Element	Description
	Other handles are intrinsic to the principal item. For example, a device may have two <code>light</code> interface blocks, which appear with interface block address <code>light/1</code> and <code>light/2</code>, respectively. Tools allocate handles when creating resources through the request channel. Tools are responsible for allocating a unique handle. For more information, see 5.11 Handle Allocation.
<i>Object_Path</i>	<i>Object_Path</i> refers to the object addressed within the device. Depending on the specific object, the <i>Object_Path</i> may be multiple levels deep and may contain additional handles. Each device supports a configuration object with the <code>cfg</code> level in the request and the feedback channel. In the request channel it is used to support modifications to the object configuration. In the feedback channel it always provides the complete and correct configuration object. The following examples show the <code>cfg</code> level in the feedback and request channels. EXAMPLE 1 <pre>glp/0/s1/rq/dev/lon/2/cfg glp/0/s1/fb/dev/lon/2/cfg</pre> Where <code>lon</code> is the <i>Driver ID</i>, <code>2</code> is the <i>Handle</i>, <code>cfg</code> is the <i>Object Path</i>. Other endpoints within a device are datapoints in the device's interface object (<code>if</code>) defined in 5.12 Interface Blocks, which contains sets of blocks of functionality. For example, one device may support two lights and one occupancy sensor, addressed with these topics: EXAMPLE 2 <pre>glp/0/s1/fb/dev/lon/2/if/light/0 glp/0/s1/fb/dev/lon/2/if/light/1 glp/0/s1/fb/dev/lon/2/if/occ/0</pre> Where <code>lon</code> is the <i>Driver ID</i>, <code>2</code> is the <i>Handle</i>, <code>if/light/0</code>, <code>if/light/1</code>, <code>if/occ/0</code> are the <i>Object_Paths</i>. Most object types support a status object (<code>sts</code>) in the feedback channel, and a <code>do</code> endpoint in the request channel used to invoke methods on the object. Many properties and nested properties within an addressable object may be addressed with the object path, but exceptions exist: a datapoint value, which is within most interface blocks, shall only be written in its entirety, and shall not be written only in parts. Therefore, a datapoint value may be addressed and written with an object path alone only if the datapoint implements a scalar type. Datapoints implementing complex types shall be written with an object path that ends at the <code>value</code> property within the datapoint object.

Topic Element	Description
	For example: value 1234 may be published to a scalar datapoint with a topic such as <code>glp/0/1/rq/dev/lon/1/if/light/0/sp/value</code>. A datapoint that implements a complex type such as a structure with <code>r, g, b</code> values, shall be written in its entirety, for example by writing <code>{r: 0.5, g: 0.7, b: 0.83}</code> to <code>glp/0/1/rq/dev/lon/1/if/light/0/sp/value</code>

4.4.3 Wildcards

The following wildcards may be used in the topic syntax for MQTT subscriptions: # and +

Clients may receive messages by creating subscriptions. A subscription may be to an explicit topic, in which case only messages to that topic will be received. Or a subscription may include wildcards. Table 3 describes the two wildcards.

Table 3 — Wildcards

Wildcard	Description
#	Represents all remaining levels of hierarchy. It shall be the final character in a subscription.
+	Represents a single level of hierarchy in a subscription.

4.5 IAP/REST Fundamentals

4.5.1 REST Overview

REST is an architectural style that allows applications to communicate effectively over HTTP by using HTTP methods (such as GET, PUT, and DELETE) to manage resources. REST is not a specification. However, it does have a number of constraints that together form the key guidelines for REST. The key constraints leveraged by IAP/REST are:

- *client-server model*, which separates the responsibilities of the client application from those of the server. Clients access a resource of the server using the resource's URI.
- *stateless*, which requires that the server does not store any client-specific information about the last HTTP request made by the client. Each request made by the client shall provide all the information required to handle the request, either provided as part of the URI, query parameters, body, or headers. The server's response shall include the information regarding the state, either included in the headers, response body, or status.
- *uniform interface*, which requires each resource to be identified using a single, unique URI, and all clients access all resources in the same manner.
- *layered system*, which allows you the option to use a layered system architecture, where the client uses the same interface when it is connected directly to the server or to an intermediary, such as a proxy or gateway.

REST components communicate by transferring a representation of a resource in a format as described by this standard, selected dynamically based on the capabilities or desires of the recipient and the nature of the resource.

4.5.2 REST in IAP

IAP/REST is an IoT access protocol based on REST. IAP/REST defines a standard set of resource types that are required for managing, monitoring, and controlling IoT devices. Resource types include devices, datapoints, alarms, logs, and schedules. Many IAP/REST resources represent a physical entity. For example, an IAP/REST device resource represents a physical IoT device. Some resources are contained within other resources. For example, an IAP/REST device resource may contain a number of IAP/REST datapoints, where each datapoint resource represents a datapoint on the physical IoT device defined by the IAP/REST device resource.

4.5.3 Making IAP/REST Calls

A client application shall authenticate with the IAP/REST server prior to making any IAP/REST requests from the server as described in **6.3.3 Authentication (Login/Logout)**.

4.6 IAP/REST Syntax

The syntax for IAP resources is as follows:

```
/iap/{Resource_Type} [/ {Resource_Detail}] [? {Query_Parameters}]
```

The syntax notation uses bold to indicate items that shall be entered as shown, and terms in italics within parentheses to represent variables that shall be replaced with an appropriate value. IAP is case-sensitive, so case shall be as shown in the syntax descriptions.

EXAMPLE

```
GET /iap/floorplans/floors/{id}
```

Table 4 summarizes the IAP/REST syntax.

Table 4 — IAP/REST Syntax

Segment	Description
iap	IAP/REST URIs always begin with /iap, where iap is a constant keyword.
Resource_Type	IAP/REST supports a variety of resource types such as devices, alarms, datapoints, etc. The supported resource types include those in the following list: - access – Geozone - alarms – Alarms - auth – Authentication - customers – Customers - devs – Devices, Device Capabilities, Datapoints - diagnostics – Diagnostics - floorplan – Floorplans - generateKey – Customers - gz – Geozone - log – Log Destination, Log Events

Segment	Description
	• <code>permission</code> – Permissions • <code>reports</code> – Reports • <code>role</code> – Roles • <code>scevent</code> – Schedule Events • <code>sch</code> – Schedules • <code>user</code> – Users • <code>version</code> – Version
<i>Resource_Detail</i>	Depending on the Resource Type, the <i>Resource Detail</i> may be zero or multiple levels deep, depending on the resource being specified. The following examples show the syntax for a <code>user</code> resource without any <i>Resource Detail</i>, and a <code>device</code> resource with two <i>Resource Detail</i> levels: For a <code>user</code> resource: <pre>GET /iap/user</pre> For a <code>devs</code> resource: <pre>GET /iap/devs/floor/{id}</pre> Resource details may also include item reference fields. For example, the Datapoint Categories resource uses the following URI: <pre>/iap/devs/{id}/if/{block_name}/{block_index}/{datapoint_name}/cat</pre> The item reference fields in this URI are <i>id</i>, <i>block_name</i>, <i>block_index</i>, and <i>datapoint_name</i>. Item reference fields may contain path parameters as described in 4.7 Queries and Parameters.
<i>Query_Parameters</i>	Query parameters may be added to the end of the URI, following a '?' character. Multiple parameters may be specified by separating each parameter with an '&' character. For more information see 4.9 Query Parameters.

4.7 Queries and Parameters

A query specifies a selection rule for a specified property. A query may be specified within a path element of the path component, or within a query component.

— A query specified in a path element is called a path parameter. See **4.8**

— **Path** Parameters.

— A query specified in a query component is called a query parameter. See **4.9**

— **Query** Parameters.

A query in IAP/REST is a key-value pair along with an operator that specifies the relationship between the key and the value. The syntax for a query is $\{key\}\{operator\}\{value\}$. Multiple key-value pairs may be specified, separated by an "&" character as a logical "AND", combining the parameters.

$\{key\}$, $\{operator\}$, and $\{value\}$ have the following descriptions:

- *key* – Identifies a property of the IAP/REST resources. When included in a path element, the *key* specifies a property within the IAP/REST object specified by the path element.
- *operator* – Specifies the relation between the *key* and *value*. Table 5 lists the available operators.

Table 5 — Queries and Parameters – Operators

Operator	Definition
==	Exact match
!=	Exact mismatch (for a fully specified string)
=+	Regular expression match, case-insensitive
!+	Regular expression mismatch, case-insensitive
<	Less than (for numbers and timestamps)
<=	Less than or equal (for numbers and timestamps)
=>	Equal or greater than (for numbers and timestamps)
>	Greater than (for numbers and timestamps)

- *value* – Specifies the value of the property to be used to identify selected resources. The value may be:
 - An absolute value for scalar properties.
 - A regular expression for the regular expression match (=+) or regular expression mismatch (!+) operators.

For the regular expression match (=+) and regular expression mismatch (!+) operators, the regular expression syntax is defined as a subset of Perl 5 syntax supported by `java.util.regex` as described in the `java.util.regex` documentation provided in Reference [2].

Regular expressions shall not include characters not specified for use in URI paths. Any such characters shall be percent encoded as described in Reference [3]. For example, Table 6 lists characters and their encoding that are reserved in Reference [4], along with additional special characters that also require percent encoding:

Table 6 — Queries and Parameters – Special Characters

Character	Encoding	Special Characters	Encoding
!	%21	newline	%0A or %0D or %0D%0A
#	%23	space	%20
\$	%24	"	%22
&	%26	%	%25
'	%27	–	%2D
(	%28	.	%2E
)	%29	<	%3C
*	%2A	>	%3E
+	%2B	\	%5C
,	%2C	^	%5E
/	%2F	_	%5F
:	%3A	`	%60
;	%3B	{	%7B
=	%3D		%7C
?	%3F	}	%7D
@	%40	~	%7E
[	%5B		
]	%5D		

Examples

This is a simple request for a named datapoint `sp` within a device with ID `myDevice`:

```
GET iap/devs/myDevice/if/Light/0/sp
```

This addresses all datapoints named `sp` within any device whose `type` property matches a regular expression, and whose block type (the profile) is reported as a particular and specific value. All matching blocks are considered regardless of their handle.

```
GET iap/devs/*+type=+^90000146164[0-4]85/if/*+type==0000000000000000-0%2FSFPTopenLoopActuator%2F/*+sp
```

4.8 Path Parameters

A path parameter may be included in any item reference field for the path component. The item reference fields are the `id`, `block_name`, `block_index`, `datapoint_name`, and `prop_name` values in the following path:

```
/iap/devs/{id}/if/{block_name}/{block_index}/{datapoint_name}/value
[/{prop_name}]
```

Path parameters may be specified in the following alternate forms:

- A single value that identifies a single object that shall be identified by the path element.
- A single asterisk (*), identifying all objects that shall be identified by the path element.
- A single asterisk followed by a plus sign (*+) followed by one or more key-value pairs ({key}{operator}{value}), each adjacent pair separated by an ampersand (&).

EXAMPLE

Request

```
GET /iap/devs/8/if/Lamp/0/nviValue/value
```

Response

```
[ {
  "deviceId" : 8,
  "blockName" : "Lamp",
  "blockIndex" : "0",
  "datapointName" : "nviValue",
  "value" : {
    "value" : 0,
    "state" : 0
  }
} ]
```

4.9 Query Parameters

Query parameters shall be added to the end of the URI, following a '?' character. If more than one query parameter is specified, then each parameter shall be separated with an '&' character.

Each query parameter has the form *<name>=<value>*, where each value shall be specified as a single string value, a simple wildcard by specifying a single asterisk ("*") as the value, or a Perl regular expression, terminated with a semicolon and enclosed in quotes ("*|<Perl regular expression>|*").

Table 7 describes the optional query parameters:

Table 7— Query Parameters

Query Parameter	Description
<code>max_age={max_seconds}</code>	For queries that return a datapoint value, <code>max_age</code> specifies the maximum datapoint age for any returned datapoint values. <code>max_age</code> forces a poll request for the datapoint, if the last datapoint value is older than the time specified by <code>max_seconds</code> . The value shall not be specified as a regular expression.
<code>pg={page}</code>	Specifies the collection of resources found on the page specified by <code>page</code> . Null means one (1). If a snapshot number is not provided with the <code>xs</code> key, the CMS creates a new snapshot of the resources identified by the request path and returns the snapshot number to the client. If a valid snapshot number is provided with the <code>xs</code> key, the page is fetched from the specified snapshot. If an invalid snapshot number is provided with the <code>xs</code> key, an error response is returned. If no page number is specified with the <code>pg</code> parameter, all resources identified by the path parameter are returned.
<code>sz={size}</code>	Specifies the number of resources per page to be used for page requests. The size shall be specified as an integer greater than zero. Null means no limit. If a <code>pg</code> parameter is specified, but the page size is not specified with the <code>xs</code> parameter, the default shall be 50. For example: 100. Resources may be skipped or duplicated if the requesting application changes the <code>xs</code> value between subsequent requests for different pages.
<code>xs={snapshot_num}</code>	Specifies the snapshot number to use when a page is requested using the <code>pg</code> parameter. When the first of a series of paginated requests is made, the requesting client shall not include a snapshot number. When a page is requested using the <code>pg</code> parameter and a snapshot number is not provided with the <code>xs</code> parameter, the requested resources are gathered, sorted in ascending order according to their ID, a new snapshot of the resources is created, and then the snapshot number is returned to the client. When a paginated response is requested with the <code>pg</code> parameter, if a valid snapshot number is provided with the <code>xs</code> parameter, the CMS fetches the page from the specified snapshot. If an invalid snapshot number is provided with the <code>xs</code> parameter, the CMS returns an error response.

4.10 IAP/WS Fundamentals

The IAP/WS API uses WebSockets to support two-way communication between client apps and the IAP server. Unlike REST, which requires the client to poll and then wait for a response, the WebSocket API

allows a client to get real-time updates from the IAP server. To use a WebSocket, the client shall issue a WebSocket request to a server, the server shall open a WebSocket, and then the server shall send one or more WebSocket updates to the client.

Clients may use WebSockets to get datapoint updates and to get data logs from an IAP/WS server.

4.11 Data Log WebSocket Queries and Parameters

4.11.1 Data Log WebSocket Overview

Only data log WebSockets use queries and parameters.

That is, the general purpose WebSocket is started by `"/iap/ws"` and does not use queries or parameters. Once a client starts a general WebSocket, the client may use REST API requests to request on-demand GET requests.

A query specifies a selection rule for a specified object or objects. A query parameter shall be specified and added to the end of a URI following a `?` character.

The remainder of this section applies to data log WebSockets.

4.11.2 Key

The *key* identifies a property of the IAP/REST resources. When included in a path element, the *key* specifies a property within the IAP/REST object specified by the path element.

4.11.3 Operator

The *operator* specifies the relation between the *key* and *value*.

4.11.4 Value

The *value* specifies the value of the property to be used to identify selected resources. The value shall be one of the following:

- An absolute value for scalar properties.
- A regular expression for the regular expression match (`=+`) or regular expression mismatch (`!+`) operators.

For the regular expression match (`=+`) and regular expression mismatch (`!+`) operators, the regular expression syntax is defined as a subset of Perl 5 syntax supported by `java.util.regex` as described in the `java.util.regex` documentation provided in Reference [2].

4.11.5 Regular Expression Encoding

Special characters specified in this section that are in locations specified in this section shall be encoded before sending the IAP/WS request to the IAP server.

Encoding in IAP/WS is not the same as IAP/REST.

Device, block and datapoint names shall not be encoded except for spaces. This is different from the REST API which does require special characters in Device, block and datapoint to be encoded.

Regular expressions shall not include characters not specified for URI paths. Any such characters shall be percent encoded as described in Reference [3]. For example, the following characters from Table 8 that are reserved in RFC 3986 shall be encoded as shown below each character:

Table 8 — Regular Expressions - Using Percent Encoding

!	#	\$	&	'	(	)	*	+	,	/	:	;	=	?	@	[	]
%21	%23	%24	%26	%27	%28	%29	%2A	%2B	%2C	%2F	%3A	%3B	%3D	%3F	%40	%5B	%5D

Following are additional special characters that require percent encoding as shown in Table 9:

Table 9 — Special Characters Requiring Percent Encoding

newline	space	"	%	-	.	<	>	\	^	_	`	{		}	~
%0A or %0D or %0D%0A	%20	%22	%25	%2D	%2E	%3C	%3E	%5C	%5E	%5F	%60	%7B	%7C	%7D	%7E

For example, to specify XIF names in a request, instead of instance names, add `*+xifName==` encoded to `%2A%2BxifName==`

```
&dpName=*+xifName=={datapoint XIF name} encoded
to &dpName=%2A%2BxifName=={datapoint XIF name}
```

```
Example: &dpName=*+xifName==nvoLuxLevel encoded
to &dpName=%2A%2BxifName==nvoLuxLevel
```

To specify datapoint instance names without using exact matches add `*+name==` encoded to `%2A%2Bname==`

```
&dpName=*+name=={datapoint instance name} encoded
to &dpName=%2A%2Bname=={datapoint instance name}
```

```
Example: &dpName=*+name==nvoLightLevel encoded
to &dpName=%2A%2Bname==nvoLightLevel
```

The `&value` parameter specifies timespans. The time span request shall be encoded.

Equal sign "=" in name parameters is not encoded but it is in `&value` parameters.

5 IAP/MQ API

5.1 Overview

IAP/MQ is the IAP API based on MQTT. MQTT is an ISO standard (ISO/IEC PRF 20922) publish-subscribe-based messaging protocol. It works on top of the TCP/IP protocol. It is a lightweight protocol that is suitable for implementing applications, services, and drivers on a wide range of devices and servers including small code footprint edge devices, edge servers, and cloud hosts. IAP/MQ is used to connect services and drivers within an edge server and is used to connect edge servers to each other and cloud applications and services.

This section defines the IAP/MQ API topics, which allow access to the commonly used resources.

For information on the IAP/MQ topic syntax, see **4.2.4 IAP/MQ Topics**.

5.2 About

5.2.1 Definition

```
glp/0/{SID}/fb/about
```

The About topic shall inform clients of the IAP server's make and model, capabilities, and limitations. This topic is updated in the feedback channel once when the edge server starts. The topic returns all the available information.

5.2.2 Example

```
{
  "capabilities":["lon-routing-rni", "lon-routing-router", "arbitrary-alarms", "device-grouping", "scheduler", "history", "pagination-snapshots"],
  "limitations":{"devices":20,"protocols":["lon","modbus"]},
  "components":[
    {"name":"ltx",
      "desc":"CNP Translator",
      "license":"Copyright (c) 2020 XYZ Corporation. All rights reserved.",
      "manufacturer":" XYZ Corporation",
      "version":"1.14.050"},
    {"name":"lte",
      "desc":"CNP Engine",
      "license":"Copyright (c) 2020 XYZ Corporation. All rights reserved.",
      "manufacturer":"XYZ Corporation",
      "version":"2.40-386"},
    {"name":"Modbus protocol engine",
      "desc":"Modbus protocol engine ",
      "license":"Copyright (c) 2020 XYZ Corporation. All rights reserved.",
      "manufacturer":"XYZ Corporation",
      "version":"v1.00.004"}
  ],
  "protocol":{"lon":{"devices":32768},"modbus":{"devices":20}},
  "copyright":"Copyright (c) XYZ Corporation. All rights reserved",
  "discovery":"satm",
  "manufacturer":"XYZ Corporation",
  "model":"XYZ Segment Controller",
  "product":"XYZSC",
  "version":"2.40.414",
  "glpVersion":0,
  "languages":["en"],
  "mru":"2019-02-08 15:12:38.600 UTC"
}
```

5.2.3 Properties

Table 10 lists the About object properties:

Table 10 — About Properties

Field Name	Type	Description
copyright	string	(Optional) Reports the copyright information. EXAMPLE 1 "Copyright (C) 2024 XYZ Corporation"
discovery	string	Reports the supported discovery method. Possible values: none Discovery of devices is not supported. This method is specified if the driver supported by this segment does not support discovery and requires explicit device creation and management, or if the devices within this segment are always present and not subject to discovery. Always-present devices include virtual devices representing hardwired I/O, for example. auto Devices are automatically discovered and created in the hierarchy. This is suitable for drivers that prevent discovery crosstalk and ambiguity. For example, an edge server operating on a strictly private media may declare <i>auto</i> discovery mode.
manufacturer	string	(Optional) Reports the manufacturer. EXAMPLE 2 "XYZ Corporation"
model	string	(Optional) Reports the model information. EXAMPLE 3 "ABC 3000"
product	string	(Optional) Reports the product name. EXAMPLE 4 "ABC by XYZ Corporation Smart Gateway"
version	string	(Optional) Reports the version of the product. EXAMPLE 5 "2.08.034"
glpVersion	number	(Optional) The version of IAP. EXAMPLE 6 "0"
capabilities	List	(Optional) Reports a list of capabilities for an edge server. An edge server may support none, all, or a subset of these capabilities. Reporting the available capabilities enables tools and services to adjust behavior accordingly. An edge server's capabilities do not change unless it is re-provisioned, repaired, replaced, or new firmware is loaded into it. Clients shall consider the following capabilities unsupported in an edge server unless they are included in the capabilities list for the edge server:

Field Name	Type	Description
		• <code>arbitrary-alarms</code> – Indicates that the edge server supports an alarm manager service that allows you to define conditions that raise and cancel alarms. See 5.3 Alarm. • <code>bridged-connections</code> – Indicates that the edge server supports point-to-point connections with infrastructure support. See 5.4 • <code>Connections</code>. • <code>bulk-load</code> – Indicates that at least one protocol supported by this edge server supports the simultaneous loading of multiple instances of edge devices with the same type. See 5.14 Load Do Action. • <code>device-grouping</code> – Indicates that the edge server may take advantage of the device grouping feature. See the provision action in 5.6.4 Device Do Action. • <code>Scheduler</code> – Indicates that the segment may manage schedules and scheduled events locally. When this capability is not declared, the CMS shall manage the schedule within itself and shall activate the related jobs using a CMS-side scheduler. See 5.17 Schedule Services. • <code>History</code> – Indicates that a history of logged event records is supported and may be retrieved. See 5.9 Events. • <code>pagination-snapshots</code> – Indicates that pagination snapshots are supported for retrieval of logged event records See 5.16 • <code>Query</code>. • <code>lon-routing-rni</code> – Indicates support for EN 14908-1 remote network interfaces (RNIs). • <code>lon-routing-router</code> – Indicates support for EN 14908-1 routers. • <code>mixed-protocol-groups</code> – Indicates support for group objects with member lists that may contain devices, or items within devices, that use different drivers.
<code>limitations</code>	Object	(Optional) Reports limitations to features supported with this edge server. Each limitation is reported as a resource name/limit pair, with resource names exactly as listed below, and the corresponding limit value as discussed with each resource below. Some limitations are intrinsic and never change or change only under the same conditions which may also cause a change to capabilities (such as an edge server's capabilities, that change

Field Name	Type	Description
		whenever the controller is re-provisioned, repaired, replaced, or new firmware is loaded into it), some limitations may be configurable through a non-IAP interface and may change when the edge server resets. Limitations shall not change without publication of the corresponding <code>About</code> object. The following limitations are defined. The limitations are in reference to IAP resources. For example, resources in reference to "groups" are in reference to "IAP group objects", which may have a direct relationship to driver groups or groups exposed in the CMS. • <code>Devices</code> – <code>N</code> (where <code>N</code> represents the number of edge devices allowed.) • <code>devices-per-group</code> – <code>N</code> (where <code>N</code> represents the number of edge devices allowed per group.) • <code>Groups</code> – <code>N</code> (where <code>N</code> represents the number of groups allowed.) • <code>discoverDevices</code> – When <code>discoverDevices</code> is set to <code>True</code>, at least one of the drivers supports the <code>discoverDevices</code> action. When <code>discoverDevices</code> is either missing or set to <code>False</code>, then that device does not support discovery. See 5.8.1 Device Discovery. • <code>groups-per-device</code> <code>N</code> – <code>N</code> represents the number of groups that an edge device may be a member of. • <code>Protocols</code> – Supported drivers specified by driver ID. The driver IDs listed in 4.4.2 Device Syntax are specified by this standard. Other driver IDs may be reported.
<code>languages</code>	<code>list</code>	(Optional) List of supported languages for localized diagnostics.

Field Name	Type	Description
components	List of component objects	(Optional) Edge servers report only components that may be individually updated or have been added, as may be the case with custom modules added by a client. Each component object shall provide the following fields identified as required, and may provide the following fields identified as optional. The list of reported components may be empty, as may be the case with a monolithic edge server architecture. Each component object shall provide the following fields identified as required, and may provide the following fields identified as optional: • <code>name</code> : String (Required) – The component's name. EXAMPLE 1 <code>"GLP Toolkit for Node.js"</code> • <code>desc</code> : String (Optional) – Component's description. EXAMPLE 2 <code>"A toolkit of common services"</code> • <code>license</code> : String (Required) – Reports a copyright notice, as shown, for proprietary software and licenses. Other components may report or refer to a different license, such as a URL to the MIT license if that component was governed by the MIT license. EXAMPLE 3 <code>"Copyright 2024 XYZ Inc."</code> • <code>manufacturer</code> : String (Required) – The manufacturer's name. EXAMPLE 4 <code>"XYZ Inc."</code> • <code>uuid</code> : String (Optional) – Contains an optional component identifier. When provided, this shall be a version 4 UUID, according to Reference [5]. Compatible values for example are available from Reference [6]. <code>uuid</code> may be used by a client that relies on close communication with a specialized user interface component or any other case where a component shall be uniquely identifiable by some other component. EXAMPLE 5 <code>"d193ab8c-bd98-4931-876b-23f5723861e5"</code> • <code>version</code> : String (Optional) – The version of the component. EXAMPLE 6 <code>"1.02.034"</code>

Field Name	Type	Description
protocol	Protocol limitation Object	The <i>protocol limitations object</i> contains limitation objects as described under limitations above, for each of the device protocols supported within the segment. This allows for the declaration of protocol-specific limitations. The global limitations are compiled from the individual protocol limitations. For example, the <code>global limitations.devices</code> property reports the total number of devices supported across all protocols while the <code>global limitations.devices-per-group</code> reports the minimum number of devices within a group supported by any of the supported protocols. Tools may use global and protocol-specific limitations to prevent erroneous configurations and requests. The limitations declared in this manner are guidelines only. For example, system or driver resource limitations may prevent the addition of a given device to a group, even though the group contains fewer than the declared <code>devices-per-group</code> limit. EXAMPLE 7 <pre>{lon: {devices: 32768, ...}, ... }</pre>
mru	timestamp	The timestamp of the most recent update for this object. All feedback channel objects include the <code>mru</code> property to provide a timestamp of the most recent update for that object.

5.3 Alarm

5.3.1 Definition

```
glp/0/{SID}/ (rq | fb) /alarm/{Handle}/{option}
```

The **alarm** topic allows you to define conditions that will trigger an alarm, as well as an optional action to be triggered by the alarm, such as publishing an alarm notification within IAP, sending an email or SMS notification, or executing a job object.

IAP includes an alarm manager service that monitors IAP for all alarm conditions and, when the alarm conditions are observed, it triggers the event reporting and logging services, as well as any alarm actions that are defined for the alarm. See **5.3.7 Alarm Actions**.

5.3.2 Example

A user may define an alarm to be triggered whenever a particular datapoint reports power consumption that exceeds 750 watts, and to send an email to a technician when the alarm is triggered. When the IAP alarm manager detects the alarm condition to be true, the alarm manager uses the event reporting and logging services to report the event, and also triggers the action defined for the alarm (sending an email to a technician).

The triggered actions may be configured, and their messages may be customized using a selection of macros. These macros are described under the `message` property in Table 13 below.

IAP identifies four states for alarms, and whenever an alarm changes state, the alarm manager reports the change through the event and event logging services. Table 11 list the alarm states.

Table 11 — Alarm States

Active	An alarm is triggered because the conditions defined for an alarm have been met.
Inactive	An alarm is cancelled because either the conditions to cancel an alarm have been met, or the conditions defined to trigger the alarm are no longer being met.
Acknowledged	A human operator responds to an alarm by performing an action through the CMS (such as clearing the alarm). Whether an alarm requires acknowledgment is configurable. An alarm that does not require explicit acknowledgment is always considered acknowledged.
Unacknowledged	A triggered alarm that is configured to require acknowledgment, has not been acknowledged by a human operator.

The alarm's state (active or inactive, and acknowledged or unacknowledged) is reflected in the events channel for the selected category. See **5.9 Events**.

Multiple conditions may be defined for an alarm. An alarm may become active when one condition is met, such as a power reading exceeding 750 W, and may return to an inactive state when another condition is met (such as the power reading dropping back below 500 W). Setting a second condition is optional; if none is provided, the one condition determines when the alarm becomes active and when it returns to inactive. In this case, with one condition set, the alarm would become inactive when the power reading drops below 750 W.

Deactivating an alarm never requires acknowledgment. But for an alarm that requires explicit acknowledgement, the alarm does not clear until it is acknowledged. An alarm that does not require explicit acknowledgment is always considered acknowledged, therefore, it is automatically cleared once the deactivation condition is met.

Alarm Lifecycle Example:

Consider an alarm that activates when a temperature reading exceeds 80 °C, deactivates once the temperature falls below 70 °C, and is configured to require explicit acknowledgment.

- 1) When the temperature exceeds 80 °C, the alarm becomes active and in an unacknowledged state, awaiting acknowledgment. The alarm manager begins tracking this alarm and reports the change in alarm state to the event and logging service.
- 2) When the temperature drops below 70 °C, the alarm becomes inactive, but remains unacknowledged. This new state is reported to the event and logging service.

When the CMS submits explicit acknowledgment of the alarm, it remains inactive, but becomes acknowledged. This new state is reported to the event and logging service. The alarm manager stops tracking the alarm.

5.3.3 Alarm Configuration

The alarm's current configuration is reported in the following topic with the properties listed in the [Alarm Properties](#) table described in **5.3.6 Alarm Properties**.

```
glp/0/{SID}/fb/alarm/{Handle}/cfg
```

5.3.4 Alarm Status

The alarm definition status is reported in the following topic:

```
glp/0/{SID}/fb/alarm/{Handle}/sts
```

Alarm status is reported with the **state** property described in Table 12.

Table 12 — Alarm Status Property

Field Name	Type	Description
state	string	Possible values: "enabled", "disabled", or "deleted".

5.3.5 Examples

Example of configuration object published on glp/0/lsg/fb/alarm/k/cfg

```
{
  "desc": null,
  "enabled": true,
  "cat": "test",
  "ackd": true,
  "activation": "gt(\"glp/0/lsg/fb/dev/lon/0/if/light/0/sp/value/h\", 10)",
  "deactivation": "eq(\"glp/0/lsg/fb/dev/lon/0/if/light/0/sp/value/h\", 30)",
  "message": null,
  "urgent": false,
  "actions": null,
  "severity": null,
  "mru": "2019-03-22 06:29:05.774 UTC"
}
```

Example of status object published on glp/0/lsg/fb/alarm/k/sts

```
{
  "state": "enabled",
  "mru": "2019-03-22 06:29:03.845 UTC"
}
```

5.3.6 Alarm Properties

Each alarm supports the properties listed in Table 13.

Table 13 — Alarm Properties

Field Name	Type	Description
desc	String	(Optional) A description of the alarm.
enabled	Boolean	(Required) Indicates whether the alarm is enabled.
cat	String	(Optional) Alarm category to use when the alarm is reported through the event and event logging service. The available categories are <code>alarm</code> and <code>data</code> . You may also define your own category. The default is <code>alarm</code> .
ackd	Boolean	(Optional) Defaults to True.
activation	String	(Required) Alarm condition expression to activate the alarm. See 5.3.8 Alarm Conditions .
deactivation	String	(Optional) Alarm condition expression to deactivate the alarm. See 5.3.8 Alarm Conditions .
message	String	(Optional) Alarm message, defaults to the activation alarm condition. The alarm message is used when reporting the alarm to the event and logging service. It is also sent with SMS actions (clipped when necessary) and used for the body of an email action. The message shall be configured using a simple template language with placeholders enclosed in braces: • <code>{topic}</code> – feedback topic under scrutiny, according to the alarm condition. • <code>{alarm}</code> – Alarm's feedback topic. • <code>{condition}</code> – Condition that triggered the activation. • <code>{date}</code>, <code>{time}</code> – Current date and time. • <code>{sid}</code> – Segment identifier. The default value is "alarm".
urgent	Boolean	(Optional) Alarm urgency, defaults to False. The alarm urgency is reported to the event reporting service. See 5.9 Events.
actions	Object	(Optional) List of actions. Possible values: "sms", "email", and "job". See 5.3.7 Alarm Actions. EXAMPLE <pre>{sms: { ... }, email: { ... } ...}</pre>
severity	String	(Optional) string describing the alarm severity in user-defined categories, such as: — "minor fault" — "major fault" — "critical fault"

Field Name	Type	Description
		The string shall contain 1 to 60 characters. The severity, if available, is included with the Event object that reports the alarm. See 5.9.5 Event Object Properties .

5.3.7 Alarm Actions

An alarm manager may invoke actions when an alarm activates. There are three basic Alarm actions: `sms`, `email`, and `job`. Each action is identified with its keyword and one or more action-specific attributes. Multiple actions may be configured for an alarm. All configured actions occur when the alarm activates.

Specific implementations may support other alarm action types, such as Slack or X notifications. Actions include template text which may be parameterized with several macros, such as the alarm source, and the condition which was met to trigger the alarm. The available actions are listed in each product's documentation.

Table 14 lists the basic Alarm actions.

Table 14 — Alarm Actions

Action	Description
<code>sms</code>	Sends an SMS text message to the specified cell phone. The <code>sms</code> action has one property: <code>to</code>, which contains an email address for use with an Email-to-SMS Gateway service, or a comma-separated list of such addresses to send the alarm to. Because SMS messages are sent using Email-to-SMS gateway services, a correct SMTP service configuration shall be specified. See the <code>SMTP</code> property in the Segment Configuration Properties table described in 5.18.2 Segment Configuration Properties. The SMS message body contains the alarm message (see the <code>message</code> property in the Alarm Properties table described in 5.3.6 Alarm Properties), truncated to fit the SMS format, if necessary. SMS services require that the edge server has Internet access. EXAMPLE 1 <pre> sms: { to: "4081234567@sms.mycellco.com" }</pre>
<code>email</code>	Sends an email message to the specified email account. The <code>email</code> action has two properties: — <code>to</code>, which accepts a comma-separated list of destination email addresses. — <code>subject</code>, which contains the subject for the message. The message body is constructed from the alarm message (described in the <code>message</code> property in the Alarm Properties table described in 5.3.6 Alarm Properties). The same macros that are available for the message body may also be used for the message subject. Emails are sent through the SMTP server configured with the segment (see the <code>SMTP</code> property in the Segment Configuration Properties table described in 5.18.2 Segment Configuration Properties). Email services require that the edge server has Internet access. EXAMPLE 2 <pre> email: { to: "supervisor@xyz.com, abc@gmail.com",</pre>

Action	Description
	<pre>subject: "Alarm: {condition}" }</pre>
job	Runs the specified job. The <code>job</code> action has three properties: — <code>activate</code> — <code>deactivate</code> — <code>acknowledge</code> Each property may specify an IAP job to be executed when the alarm is activated, deactivated, or acknowledged, respectively. Jobs are referenced with their request channel topic. EXAMPLE 3 <pre>job: { activate: "glp/0/downtown/rq/job/123" }</pre>

5.3.8 Alarm Conditions

5.3.8.1 Definition

```
eq(topic, value)
gt(topic, value)
lt(topic, value)

updated(topic, interval, unit)
changed(topic, interval, unit)
changed(topic, interval, unit, threshold)

recent(topic)
```

IAP provides a number of functions that allow you to establish the desired conditions for your alarms. The alarm condition is expressed in JavaScript. A valid JavaScript construct shall be used to combine or evaluate the results of these functions, or to construct arguments to these functions. The *topic* is typically, but not necessarily, a feedback channel topic.

5.3.8.2 Comparison Functions

```
eq(topic, value)
gt(topic, value)
lt(topic, value)
```

There are three comparison functions `eq` (equals), `gt` (greater than), and `lt` (less than). These functions compare the most recent value received for the specified *topic* with the specified *value*, and return a Boolean result. The *topic* is typically, but not necessarily, a feedback channel topic.

The alarm condition is expressed in JavaScript. JavaScript expressions that use the standard JavaScript operators shall be used to combine or evaluate the results of these functions, or to construct arguments to these functions.

When the alarm manager encounters the *topic* for the first time, it separates it into the IAP resource *topic* and the property selector parts, and subscribes to the IAP resource *topic* to track changes. When a newly published value is received on that topic, all alarm conditions that are based on that topic are evaluated. A resource topic is a topic that appears in one of the IAP channels, and the property selector is a slash-separated list of properties within that topic.

Topic examples:

This topic refers to an IAP resource:

```
glp/0/dt/fb/dev/cm/0/if/light/0
```

This topic refers to a property within the same IAP resource:

```
glp/0/dt/fb/dev/cm/0/if/light/0/sp/value/b
```

In the three comparison functions (*eq*, *gt*, *lt*), *value* accepts any value compatible with the *topic*: a string, number, Boolean constant, list, or object.

- Null shall only be equal to itself (null eq null is always True).
- Null shall always be less than any other value (except for null).
- Two numbers are compared using standard arithmetic.
- Two strings are compared using lexicographic sorting.
- Two lists are compared with their member count; the list with more members is "greater than" the list with fewer members.
- Objects shall only be compared for equality (or non-equality); ordered comparisons such as "less than" or "greater than" yield a runtime error when applied to an object.
- Comparison between items of a different type may be possible if the automatic data conversion rules described in **5.4.7 Type Translation** yield a value pair that may be compared. For example, number 17 may be compared with string "18", and 17 is less than "18".
- Standard JavaScript operators `!`, `&&` or `||`, among others, are available and supported to provide logic inversion and combination:

```
!gt("glp/0/dt/fb/dev/cm/0/if/light/0/sp/value/b", 0.5)
```

5.3.8.3 Updated and Changed Functions

```
updated(topic, interval, unit)
changed(topic, interval, unit)
changed(topic, interval, unit, threshold)
```

These functions detect whether a value has been updated or changed within a certain interval:

- `updated()` yields true if the value referenced by *topic* has been published, regardless of its value, less than the amount of time specified by *interval*.

- `changed()` without a *threshold* yields true if the value referenced by *topic* has been published and changed from the previous value less than the amount of time specified by *interval*. The alternative form uses an explicit *threshold* value to control the amount by which the value shall change.

When a newly published value is received on that topic, all alarm conditions that depend on the topic are evaluated. The alarm manager repeatedly evaluates the alarm conditions that use `updated()` or `changed()` functions as per the *unit* specified: every millisecond if unit is 'ms', every second if unit is 's', and so forth. That way, the alarm manager does not evaluate alarm conditions more than necessary, and the user has control over the accuracy of the time expression: a condition with a timer of 60s is evaluated once per second, while a condition with a timer of 60 000 ms is evaluated 1000 times per second.

A specific alarm manager implementation may restrict high-frequency evaluations, for example, to a maximum rate of 10 per second.

Table 15 lists the `updated()` and `changed()` function parameters.

Table 15 — Updated and Changed Functions Parameters

<i>topic</i>	(Required) See the description of <i>topic</i> for the comparison functions (<code>eq</code> , <code>gt</code> , and <code>lt</code>), described in 5.3.8.2 Comparison Functions .
<i>interval</i>	(Required) A number expressed as multiples of the duration specified with <i>unit</i> .
<i>unit</i>	(Required) Shall be one of the following: ms milliseconds s seconds h hours d days
<i>threshold</i>	(Optional) Defines the unsigned threshold by which the value shall change within the duration specified by <i>interval</i> and <i>unit</i> .

5.3.8.4 Recent Function

`recent(topic)`

The `recent()` function returns the most recent value associated with *topic*, or null, if none is known. The definition of *topic* is the same as for the comparison functions (`eq`, `gt`, `lt`) described in **5.3.8.2 Comparison Functions**.

5.3.8.5 Executing Alarm Conditions

Executing a user-defined alarm condition code shall be restricted to calling the well-defined functions above, and combinations with built-in operators. Any syntax errors or attempts to access functionality that is not specified by the alarm condition sandbox shall be reported as runtime errors.

5.3.8.6 System Alarm Conditions

Specific implementations may also support special topics in addition to topics referencing process values. Those special topics may take the following form: `glp/0/./=alarm/...`

An alarm manager may also expose system resources such as the CPU temperature, overall memory usage, or an individual process' memory consumption, allowing for alarm conditions like the following CPU temperature monitor: `gt("glp/0/./=alarm/system/cpu/temperature", 85)` The available system topics are listed with each product's documentation.

5.3.9 Manage Alarms

```
glp/0/{SID}/rq/alarm/{Handle}/do
```

Alarms are managed by publishing the actions listed in Table 16 on the alarm `do` topic.

Table 16 — Manage Alarms Actions

Action	Description
create	Creates an alarm. Arguments are listed in the Alarm Properties table described in 5.3.6 Alarm Properties .
delete	Deletes an existing alarm. No arguments shall be specified for this action.
update	Updates an alarm. Arguments are listed in the Alarm Properties table described in 5.3.6 Alarm Properties .

EXAMPLE

```
glp/0/S5tVx3x/rq/alarm/h.1/do
{ action: "create" }
```

5.4 Connections

5.4.1 Definition

```
glp/0/{SID}/(rq|fb)/con/{Handle}/{object}
```

Datapoint connections are defined under the `glp/0/SID/(rq|fb)/con` topic. An individual connection shall be defined with a handle indicated with `{Handle}`. Connection objects support direct data flow from source datapoints to destination datapoints, ranging from true peer-to-peer connections within a given driver to assisted connections including type and driver translation.

IAP supports datapoint connections of one or more data sources, typically a sensor such as a temperature sensor, a traffic counter, or a call button to one or more data destinations. Destinations are physical outputs such as fan controls, lighting controls, or logical entities such as digital controller algorithms. For example, an occupancy sensor may drive the occupancy state of 5 different VAV controllers and lights.

See **5.4.7 Type Translation** for information on type conversions between source and destination types.

5.4.2 Connection Implementation

- Connections may be implemented as true peer-to-peer connections within participants of the same driver, if that protocol has facilities for peer-to-peer connections and required resources remaining. A true peer-to-peer connection allows for data flow across the connection, initiated by the data source as new data becomes available and not requiring a facilitator.

- Connections shall be implemented as point-to-point connections when protocol-specific peer-to-peer connections are not possible. Point-to-point connections may span multiple device protocols or segments and involve data type translation and require a facilitator. The facilitator may be actively looking for data as may be the case in a master/slave fieldbus or may do little more than relay data from one system to another when such data becomes available. The facilitator may also transform the source data to meet the destination's data type. See **5.4.6 Connection Types** for more information about peer-to-peer and point-to-point connections.
- Connections shall be created and provisioned.
- A single connection may have multiple sources and multiple destinations, and shall have at least one source and one destination. These are collectively known as connection endpoints.
- The principal type of a connection endpoint is a datapoint value, referenced by a feedback or request channel topic to a datapoint's *value* property. Feedback channel references shall be used for source endpoints, request channel references shall be used for destination endpoints.

5.4.3 Connection with Datapoint Presets

Connections with datapoint presets described in **5.12.5 Datapoint Presets** and localization transformations described in **5.12.6 Datapoint Localization** typically require a facilitator to do the transformation from IAP to local value, and from local datapoint value to preset value even if the connection is between data points implemented using the same driver within the same segment, and if that driver supports true peer-to-peer connections. The facilitator may also transform the preset source data to meet the destination's data type. The facilitator may decide to skip the data from the source if the destination datapoint only expects a preset value, and the preset value does not map to any of the destination presets.

The **Presets** and **Localization** options are used to connect elements of source properties to destination properties. They use the same transformation rules, syntax, and error events like `map`. (See **5.4.7 Type Translation** for more information.)

- Unlike `map`, presets and localization may be used in sources and destinations.
- There is only one presets/localization for all sources, but all destinations may have different presets/localization.
- Multiple sources with different presets/localization may use a different connection rule.
- The transformation shall be applied at the source, destination, or both.
- Data arrives at segment S1, and S1 is responsible for applying a suitable localization/preset transformation before being sent to segment S2. Then the destination segment S2 is responsible for applying a suitable preset/localization transformation.
- When the presets are defined in the destinations, the `map` option is ignored. The destination presets will be used to transform the source according to the destination presets.
- See **presets** and **localization** described in **5.4.4.3 Connection Configuration Properties** for more information.