**Booking Software & Building Controlling System**

Data Exchange Protocol

*Nordic Standard*

2019-10-21

Level 2

Subversion 1

*Document version 1*

**Companies that conforms to this document.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Country | Date | Contact | Color |
| Regler Och Webbteknik Sverige AB | Sweden | 2016-09-21 | Robin Andersson  [robin@rows.se](mailto:robin@rows.se) | Yellow |
| Agrando Sweden AB | Sweden | 2016-09-22 | Rune Hansen [rune.hansen@agrando.no](mailto:rune.hansen@agrando.no) | Blue |
| LJ System AB | Sweden | 2016-09-23 | Niklas Falemar  [niklas@ljsystem.se](mailto:niklas@ljsystem.se) | Orange |
| Eniac Data AB | Sweden | 2016-09-23 | David Fröjmark  [david.frojmark@eniac.se](mailto:david.frojmark@eniac.se) | Green |
| JEFF Electronics AB | Sweden | 2016-09-26 | Björn Granbom  [bjorn@jeff.se](mailto:bjorn@jeff.se) | Red |
| Mild Media AB | Sweden | 2016-09-27 | Albin Nystedt | Purple |
| M&V Software Oy | Finland | 2016-10-31 | Martin Wikström  [martin@mvs.fi](mailto:no_reply@apple.com) | Brown |
| Diaprint Automatik A/S | Denmark | 2019-10-21 | Steen Spaten [steen@diaprint.dk](http://www.apple.com/se) | Turquoise |

**Introduction**

The goal with this standard is to simplify and unify the data exchanges between booking administration softwares and building controlling systems.

**Abbreviations and explanations**

**Booking administration software:** is referred to an administration software where a user has the ability to schedule the use of one or more resources.

In this document Booking Administration Software will be shorted to BAS.

**Building controlling systems:** is refereed to systems that in one way or another controls a building and/or its surroundings.

In this document Building Controlling Systems will be shorted to BCS.

**GUID/UUID:** will be refereed to UUID in this document. The similarities between Microsoft’s version (GUID) and Unix version (UUID) makes it unnecessary to specify witch one can be used. When “UUID” is stated in this document it implies both GUID and UUID.

**Data Transfer**

The data are to be transferred with HTTP or HTTPS as a POST method.

The URL shall always be the same regardless of what method is invoked.

All transfers are initiated by the BCS.

Thus all requests will be made by a BCS and all responses comes from a BAS.

BAS is a holder of the API and the BCS is a user of the API.

HTTP Keep-Alive are preferred.

**Request/response payload**

The payload data must be formatted as valid **JSON**.

The JSON object must be passed as the body of the POST request.

All variable names are **case sensitive**.

The order of variables is **not predetermined** and can be in any order within the JSON object.

All “id” values must have the format as a UUID string, but the generation does not need to follow the UUID nor the GUID standard.

Every request must include a **clientID** which is a unique identification of the BCS system so that the BAS can filter unwelcome requests.

Every request must also include a **clientToken**.

The purpose of the token is to verify the BCS and it’s **clientID**.

The token is a hash generated by using **HMAC-SHA1** expressed as hex string.

The hash message is a string containing **systemTime, clientID** and the current requested method.

Which is then **HMAC-SHA1** hashed together with a **clientKey** and expressed as a hex string called **clientToken**.

The clientKey must be sent to the HMAC routine as a UUID string.

The **clientKey** is a UUID generated by the BAS and given to BCS (through email or other form of external communication) together with the **clientID** before any methods can be called.

All BCS can use the same **clientID** between different BAS.

However all BCS must have a unique **clientKey** between different BAS.

**Generation of the clientToken.**

string **clientKey** = "[UUID]";

string **message** = "[systemTime] + [clientID] + [method]"

byte **clientHash**[20] = HMAC-SHA1(clientKey, message)

string **clientToken** = byte\_to\_hexstring(clientHash)

|  |  |  |  |
| --- | --- | --- | --- |
| message | | | |
| Name | Type | Description | Example |
| systemTime | String | UInt64 epoch SystemTime in seconds. If the time deviates more than +-10 minutes from BAS clock, the request must be ignored by BAS. | 1475226019 |
| clientID | String | BCS clientID expressed as a UUID string. | 9818d49a-005d-4a83-93b3-9de04a6a5225 |
| method | String | Current invoked method.  This must be case sensitive. | GetCustomerData |
| Example: 14752260199818d49a-005d-4a83-93b3-9de04a6a5225GetCustomerData | | | |

The calculated clientHash is then expressed as hex string in the request called clientToken.

The result should match the following: (from ANSI C example)

**systemTime**: "1475226019"

**clientID**: "9818d49a-005d-4a83-93b3-9de04a6a5225"

**method**: "GetCustomerData"

**clientKey**: "5878b222-9781-4e1b-936f-ef9ccad60518"

**message**: "14752260199818d49a-005d-4a83-93b3-9de04a6a5225GetCustomerData"

**clientToken**: "9085856495ee242be7b2b6228517d6778a00de4d"

**Fetching customer data**

*This method requests information about the customer and their resources*.

REQUEST

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Note** |
| method | Requested method. Every request must have this.  The value is case sensitive. |  |
| client | Information about the BCS client. |  |
| client.api | Request version number as signed integer. This is to ensure that future changes that break backward compatibility does not destroy existing connections.  The value is a string divided into three parts with dotts.  **Part 1:** API Level implementation.  **Part 2:** Method version. Is only to be changed if backward compatibility is broken.  **Part 3:** Documentation standard version.  The BAS must return status.code = 460 or 461 if the version of part 1 or 2 is unknown. (See status.code section below)  Part 3 should not create any errors. |  |
| client.id | A unique identification of the BCS system. |  |
| client.time | Epoch time used to create clientToken. (UTC) |  |
| client.token | The resulting hash expressed in hex. |  |
| payload | Object containing data for the method. |  |
| payload.customers | A list of requested BAS customers (UUID). The customer id is determined by BAS and handed to BCS prior to installation. |  |

**Example request:**

{

“method”: “GetCustomerData”,

“client”: {

“api”: “1.1.12”,

“id”: “9818d49a-005d-4a83-93b3-9de04a6a5225”,

“time”: 1475226019,

“token”: “a43074f5bfa356d3be7dda51ef2e4016292743bd”

},

“payload” : {

"customers": ["e3941203-37c8-4aaf-a10c-a46100ccb787"]

}

}

RESPONSE

|  |  |  |
| --- | --- | --- |
| Name | Description | Note |
| status | Must exist in all responses. |  |
| status.code | Signed integer. Based on the HTTP standard codes (<https://en.wikipedia.org/wiki/List_of_HTTP_status_codes>). Future additions to the codes should if possible be based on it’s HTTP counterpart or selected so that it doesn’t conflict with any known HTTP code.  Success  **200** = OK (All is good and data returned as requested.)  **201** = Created (Data received and processed.)  **202** = Accepted (Data received and are still being processed.)  **204** = No Content (Request OK but no data returned.)  **299** = OK Deprecated (Same as 200 except the method will be removed in next level. Alternative method might be suggested in status.msg.)  Client Error  **400** = Bad Request (Invalid JSON in request.)  **401** = Unauthorized (Unknown clientID or invalid token.)  **405** = Method Not Allowed (Requested method does not exist.)  **409** = Conflict (Sent data already exist.)  **410** = Gone (Method does not exist anymore. Must have returned 299 in previous level.)  **418** = I'm a teapot  **460** = Level Version Out Of Range (Requested level version not yet implemented.)  **461** = Method Version Out Of Range (Requested method version not yet implemented.)  Server Error  **500** = Internal Server Error (Fatal error in BAS. Further debug error code may be added to status.msg.)  **OBS:** These codes, even though based on the HTTP response codes, should **not** be set in the response header as the HTTP code. The HTTP code should always be 200. |  |
| status.msg | Description of the warning or error. This should only be used by humans for debugging and not by the parser. If status.code 500 is returned, the message may include a code used by the BAS for debugging. |  |
| server | Object containing information about BAS server. |  |
| server.api | The highest version supported by BAS.  The BCS should note this and conform to older version if BCS is more updated than the BAS. |  |
| server.time | Server epoch time. (UTC) |  |
| payload | Object containing data for the method. |  |
| payload.customers | A list of requested customers. |  |
| payload.customers[x].id | Customer id. |  |
| payload.customers[x].name | The name of the customer within BAS. |  |
| payload.customers[x].resources | A list of resources this customer has created. |  |
| payload.customers[x].resources[y].id | A unique id for the resource.  The ID must be formated as a UUID. The generation of the ID is not specified. |  |
| payload.customers[x].resources. [y].name | The name of the resource in BAS.  A rename of a resource in BAS should not alter the id.  **Optional:** if BCS wants to rename its resource/zone name automatically.  **Optional:** if the BAS has hierarchic architecture and wants to display the full path. It should in that case be separated with a / |  |
| payload.customers[x].resources.[y].type | The type of the resource. Should be considered a ENUM and can only be any of the following:  **heat** = The resource returned is a location (must commonly a church) with a system regulating the heating. **bell** = The resource returned is one or multiple bells with a system controlling when the should ring. |  |

**Example response:**

{

"status": {

"code": 200,

"msg": "ok"

},

“server”: {

“api”: “1.1.12”,

“time”: 1475226019

},

“payload” : {

"customers": [{

"id": "e3941203-37c8-4aaf-a10c-a46100ccb787",

"name": "Customer name",

"resources": [{

"id": ”5817c100-d599-4f2e-9c25-07e7a64075a0",

"type": "heat",

"name": "Location/resource one"

}, {

"id": "fe77c299-980e-49a2-82a5-4f42a4cadf34",

"type": "heat",

"name": "Location/resource two"

}, {

"id": "eeefa7a1-add0-4d76-967b-27263921d329",

"type": "bell",

"name": "Bell/resource three"

}]

}]

}

}

**Fetching resource data**

*Fetching list of booked instances.*

REQUEST

|  |  |  |
| --- | --- | --- |
| Name | Description | Note |
| method | See method “GetCustomerData” for more info. |  |
| client | See method “GetCustomerData” for more info. |  |
| client.api | See method “GetCustomerData” for more info. |  |
| client.id | See method “GetCustomerData” for more info. |  |
| client.time | See method “GetCustomerData” for more info. |  |
| client.token | See method “GetCustomerData” for more info. |  |
| payload | Object containing data for the method. |  |
| payload.dateFormat | Specifies what type of format the date and times must have in this request. Both the request and response must conform to selected format.  “string” = Dates and times is to be represented as string in the format “yyyy-mm-dd hh:MM:ss” in GMT timezone.  “epoch” = Dates and times is to be represented as Unix epoch times. The number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT) It must interpreted as UInt64 |  |
| payload.start | Specifies from what date and time (GMT time) the bookings should start.  Every booking that has it’s start and/or end date **on or after** this time.  If dateFormat is set to “epoch” this value must also conform to epoch standard and vise versa. |  |
| payload.end | Specifies from what date and time (GMT time) the bookings should end.  Every booking that has it’s start and/or end date **before** this time.  If dateFormat is set to “epoch” this value must also conform to epoch standard and vise versa. |  |
| payload.resources | A list of requested resources.  The resources id is requested with method: “GetCustomerData” in beforehand. |  |

**Example request:**

{

“method”: “ GetResourceData”,

“client”: {

“api”: “1.1.12”,

“id”: “9818d49a-005d-4a83-93b3-9de04a6a5225”,

“time”: 1475226019,

“token”: “a43074f5bfa356d3be7dda51ef2e4016292743bd”

},

“payload”: {

”dateFormat”: ”string”

"start": "2015-05-01 00:00:00",

"end": "2016-05-01 00:00:00",

"resources": [

"5817c100-d599-4f2e-9c25-07e7a64075a0",

"fe77c299-980e-49a2-82a5-4f42a4cadf34",

"eeefa7a1-add0-4d76-967b-27263921d329"

]

}

}

RESPONSE

|  |  |  |
| --- | --- | --- |
| Name | Description | Note |
| status | See method “GetCustomerData” for more info. |  |
| status.code | See method “GetCustomerData” for more info. |  |
| status.msg | See method “GetCustomerData” for more info. |  |
| server | See method “GetCustomerData” for more info. |  |
| server.api | See method “GetCustomerData” for more info. |  |
| server.time | See method “GetCustomerData” for more info. |  |
| payload | Object containing data for the method. |  |
| payload.list | The list of requested bookings.  Booked start time < Requested end time  and Booked end time > Requested start time |  |
| payload.list[x].resource | Current resource id. |  |
| payload.list[x].id | Unique id for this specific booking instance.  No duplicates!  This is an “instance id”, not booking id.  If the booking is a repeated booking “I.e every Sunday” every Sunday must produce its own unique id. This is to accommodate the future function to let BCS report back information on past booking instances to the BAS. |  |
| payload.list[x].start | The start (GMT time) of the booking.  Expressed as selected dateFormat. |  |
| payload.list[x].end | The end (GMT time) of the booking.  Expressed as selected dateFormat. |  |
| payload.list[x].created | The date and time (GMT time) current booking was made in the BAS.  Expressed as selected dateFormat. |  |
| payload.list[x].signature | A signature/name on the person who made the current booking. |  |
| payload.list[x].heat | Indicates what type of heat/temperature the booker wants on this particular booking.  In the BAS this should be a dropdown or similar presented to the booker when the booking is made or the the template is defined.  All values smaller than zero is predefined temperatures in BCS by the facility manager.  **-3** = Cleaning temperature. Predefined temperature for cleaning staff. (Defined in BCS system)  **-2** = No heat. The temperature should be the same as if there was no booking at this time. The humidity protection should be active. This value is meant to be used when the resource is booked but no people will be there.  **-1** = No heat. The temperature should be the same as if there was no booking at this time. But the humidity protection should be disabled. This value is meant to be used when the resource is booked and there will be people there, but the temperature is irrelevant.  **0** = Standard heat. No change. The predefined booked temperature is to be used.  **> 0** = Selected temperature. I.e value of 21 means that the desired temperature for this booking is 21°C  **null** = The resource doesn’t have any heating data. |  |
| payload.list[x].program\_id | An ID string of what bell program should run at the given start time.  **string** = The ID of the program that should run.  **null** = The resource doesn’t have any program data. |  |
| payload.list[x].title | The title of this booking. |  |

**Example response:**

{

"status": {

"code": 200,

"msg": "ok"

},

“server”: {

“api”: “1.1.12”,

“time”: 1475226019

},

“payload”: {

"list": [{

"resource": "5817c100-d599-4f2e-9c25-07e7a64075a0",

"id": "afff431b-2835-45a1-9c5e-a100d746ea0c",

"start": "2015-05-05 11:30:00",

"end": "2015-05-05 12:00:00",

”created”: ”2015-05-01 11:00:00”,

”signature”: ”Eva Andersson”,

"heat": 0,

”program\_id”: null,

"title": "Booking with standard heat temp"

}, {

"resource": " fe77c299-980e-49a2-82a5-4f42a4cadf34",

"id": "99c42508-0e9c-4eef-af19-d7dbb48f9b27",

"start": "2015-04-30 18:30:00",

"end": "2015-05-02 19:30:00",

”created”: ”2015-04-01 11:00:00”,

”signature”: ”Eva Andersson”,

"heat": -2,

”program\_id”: null,

"title": "Do not use the resource."

}, {

"resource": " eeefa7a1-add0-4d76-967b-27263921d329",

"id": "bfeb2312-070d-441f-b863-37bd40dd8886",

"start": "2015-04-30 18:30:00",

"end": "2015-05-02 19:30:00",

”created”: ”2015-04-01 11:00:00”,

”signature”: ”Eva Andersson”,

"heat": null,

”program\_id”: "63c43a36-ebab-4229-8b85-2966421bcfb7",

"title": "Ring the bells."

}]

}

}

Document version notes

Notes for v001:

**1.** LJ System

Increased the level from 1 to 2. This because suggested changes to the standard introduces new data fields not regarding heating.  
  
**2.** LJ System  
Added Diaprint Automation A/S to the list of conforming companies.  
  
**3.** LJ System  
Clarified that the **clientKey** is generated by BAS and given to BCS and not the other way around.  
  
**4.** LJ System  
Added the field **payload.customers[x].resources[y].type** to the **GetCustomerData** method. Adding the possibility for BCS to filter out the resource types they are interested in.  
  
**5.** LJ System  
Changed the field **payload.list[x].heat** on the method **GetResourceData** to be nullable. This is for when the resource type is **bell** and no heating data is available.  
  
**6.** LJ System  
Added the field **payload.list[x].program\_id** to the method **GetResourceData**. If the resource type is **bell** this field contains the ID of the program to run at the given start time. When the resource type isn’t **bell** the field should be **null**.  
  
**7.** LJ System  
Cleared all the document version notes regarding level 1.  
  
**8.** LJ System  
Added new document version notes describing the changes made for level 2.