PT Citra Tubindo, Tbk.

PostgreSQL Extension for SATO Printer

Programming Reference

CV Datatrans Informatika

2019

Table of Contents

Chapter 1. Configuration

PostgreSQL Extension for SATO Printer is a set of database object to drive TCP/IP accessible SATO printer. The object consists of table, PLPGSQL and C user defined function.

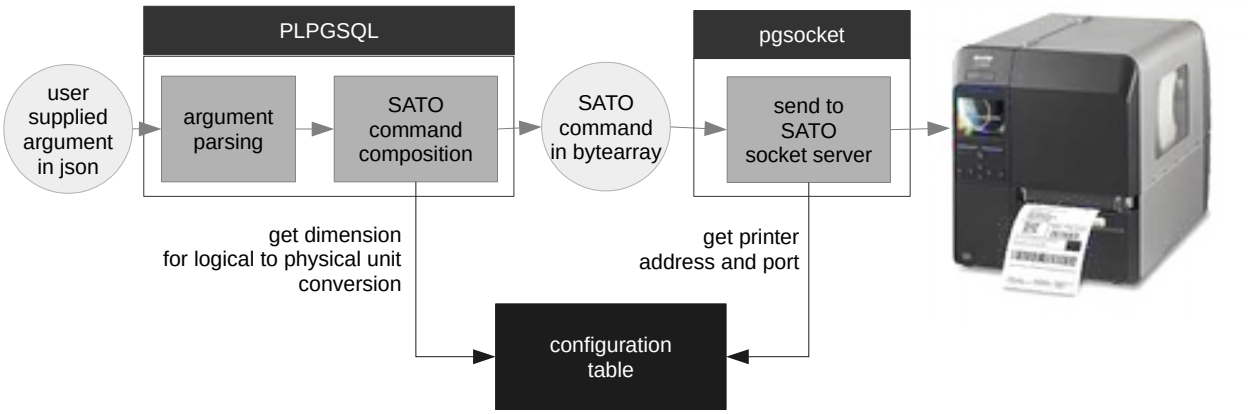

Figure 1: Function Diagram

User invokes function by supplying printing commands as argument in json array. The function written in PLPGSQL parses the json array and convert it into SATO ESC commands. A configuration table provides factors for conversion from logical unit (millimeter as supplied by user) into physical unit (dot as requested by SATO). The resulted SATO ESC command (in bytearray) is sent to SATO printer by invoking function provided by pgsocket.

1.1. Installation

1.1.1. PostgreSQL Server

- Minimum version 8.4.
- Linux operating system.
- Use package manager or compiled from source.

1.1.2. pgsocket

It is a PostgreSQL extension to send to and get response from TCP socket server. Get **pgsocket** source from https://github.com/AbdulYadi/pgsocket. Use git clone or download zip file and unzip it.

```
git clone https://github.com/AbdulYadi/pgsocket.git
```

- Check Makefile, ensure **pg_config** path in PG_CONFIG entry is valid (you may change the path according to your specific PostgreSQL installation).

```
MODULE_big = pgsocket
OBJS = pgsocket.o
EXTENSION = pgsocket
DATA = pgsocket--1.0.sql

PG_CONFIG = pg_config
```

```
PGXS := $(shell $(PG_CONFIG) --pgxs)

include $(PGXS)
```

- Compile with regular user privilege

```
make
```

- Install with root privilege

```
su - root
make install
```

On successful installation there will be 2 user defined functions:

1. public.pgsocketsend(text, integer, integer, bytea): send data using TCP socket to specified host address and port without reading the response.
2. public.pgsocketsendrcvstxetx(text, integer, integer, integer, bytea): send data using TCP socket to specified host address and port and read response from the host prefixed by STX and terminated by ETX.

1.1.3. pgreadfile

It is a PostgreSQL extension to read file in binary format. Get **pgreadfile** source from https://github.com/AbdulYadi/pgreadfile. Use git clone or download zip file and unzip it.

```
git clone https://github.com/AbdulYadi/pgreadfile.git
```

- Check Makefile, ensure **pg_config** path in PG_CONFIG entry is valid (you may change the path according to your specific PostgreSQL installation).

```
MODULE_big = pgreadfile
OBJS = pgreadfile.o
EXTENSION = pgreadfile
DATA = pgreadfile--1.0.sql

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)

include $(PGXS)
```

- Compile with regular user privilege

```
make
```

- Install with root privilege

```
su - root
make install
```

On successful installation there will be 1 user defined functions: public.pgreadfile(text): read file as specified by text argument.

1.1.4. pgsato

Get **pgsato** source from https://github.com/AbdulYadi/pgsato. Use git clone or download zip file and unzip it.

```
git clone https://github.com/AbdulYadi/pgsato.git
```

Install it by executing commands in **sato.sql** file:

```
<path-to-psql>psql -U <user> -h <host> -p <port> -f sato.sql -d <dbname>
```

Replace <path-to-psql>, <user>, <host>, <port>, <dbname> with your specific setting.

On successful installation, there will be:

1. Table sato.server.
2. Public function:
    2.1. sato.status(IN t_server text, OUT i_printerstatus smallint, OUT t_printerstatus text, OUT i_bufferstatus smallint, OUT t_bufferstatus text, OUT i_ribbonstatus smallint, OUT t_ribbonstatus text, OUT i_mediastatus smallint, OUT t_mediastatus text, OUT i_error smallint, OUT t_error text, OUT i_batterystatus smallint, OUT t_batterystatus text, OUT i_remainprint integer) RETURNS SETOF record
    2.2. sato.bytea_import(p_path text) RETURNS bytea
    2.3. sato.print(IN t_server text, VARIADIC j_jobs json[], OUT b_rfidwritesuccess boolean, OUT t_err text, OUT t_epc text, OUT t_tid text) RETURNS SETOF record
3. Private function:
    3.1. sato.status_do(IN t_host text, IN i_port smallint, OUT i_printerstatus smallint, OUT t_printerstatus text, OUT i_bufferstatus smallint, OUT t_bufferstatus text, OUT i_ribbonstatus smallint, OUT t_ribbonstatus text, OUT i_mediastatus smallint, OUT t_mediastatus text, OUT i_error smallint, OUT t_error text, OUT i_batterystatus smallint, OUT t_batterystatus text, OUT i_remainprint integer) RETURNS SETOF record
    3.2. sato.code2d_dm(j_arg json) RETURNS text
    3.3. sato.code2d_qr(j_arg json) RETURNS text
    3.4. sato.control_qty(i_qty integer) RETURNS text
    3.5. sato.font_text(i_dpi smallint, j_arg json) RETURNS text
    3.6. sato.graphic_bmp(t_hex text) RETURNS text
    3.7. sato.helper_mmdot(i_mm numeric, i_dpi smallint) RETURNS integer
    3.8. sato.intel_backfeed(i_dpi smallint, i_min integer, i_max integer, j_arg json) RETURNS text
    3.9. sato.intel_feed(i_dpi smallint, i_min integer, i_max integer, j_arg json) RETURNS text
    3.10. sato.position_horz(i_dpi smallint, i_max integer, i_pos numeric) RETURNS text
    3.11. sato.position_origin(i_dpi smallint, i_vertmax integer, i_horzmax integer, j_arg json) RETURNS text
    3.12. sato.position_vert(i_dpi smallint, i_max integer, i_pos numeric) RETURNS text
    3.13. sato.rfid_uhfwrite(j_arg json) RETURNS text
    3.14. sato.system_mediasize(i_dpi smallint, i_maxlabelheight integer, i_maxlabelwidth integer, j_arg json) RETURNS text

Chapter 2. Configuration Table

SATO printing configuration is defined in **sato.server** table with following fields:

1. **id** text NOT NULL: unique printer identifier, e.g. sato1.
2. **host** text NOT NULL: host address in name or IP address, e.g. satoserver or 10.109.129.31.
3. **port** smallint NOT NULL: port number, typically 9100.
4. **dpi** smallint NOT NULL: printer head density in dpi[1] unit. CL4NX printer has density of 203 dpi, for example.
5. **maxlabelheight** integer NOT NULL: maximum printing label length in dot unit. CL4NX with 203 dpi has maximum length of 20000 dots, for example.
6. **maxlabelwidth** integer NOT NULL: maximum printing label width in dot unit. CL4NX with 203 dpi has maximum width of 832 dots, for example.
7. **minfeed** integer NOT NULL: minimum printing label forward feed in dot unit. CL4NX with 203 dpi has minimum feed of 48 dots, for example.
8. **maxfeed** integer NOT NULL: maximum printing label forward feed in dot unit. CL4NX with 203 dpi has maximum feed of 1600 dots, for example.
9. **minbackfeed** integer NOT NULL: minimum printing label backward feed in dot unit. CL4NX with 203 dpi has minimum feed of 48 dots, for example.
10. **maxbackfeed** integer NOT NULL: maximum printing label backward feed in dot unit. CL4NX with 203 dpi has minimum feed of 480 dots, for example.
11. **maxvert** integer NOT NULL: maximum printing vertical position in dot unit. CL4NX with 203 dpi has maximum vertical position of 20000 dots, for example.
12. **maxhorz** integer NOT NULL: maximum printing horizontal position in dot unit. CL4NX with 203 dpi has maximum horizontal position of 832 dots, for example.

Example: create configuration record for SATO CL4NX 203 dpi with id sato1, ip address 10.109.129.31 and port 9100:

```
INSERT INTO sato.server (id, host, port, dpi, maxlabelheight, maxlabelwidth,
minfeed, maxfeed, minbackfeed, maxbackfeed, maxvert, maxhorz) VALUES
('sato1', '10.109.129.31', 9100, 203, 20000, 832, 48, 1600, 48, 480, 20000,
832);
```

1 dpi (dot per inch) is a measurement of printer resolution indicating how many ink dots the printer can place in one square inch.

Chapter 3. Public Function

Public function is the one executable by public. Any user successfully login to PostgreSQL server can execute this function.

3.1. sato.status

sato.status(IN t_server text, OUT i_printerstatus smallint, OUT t_printerstatus text, OUT i_bufferstatus smallint, OUT t_bufferstatus text, OUT i_ribbonstatus smallint, OUT t_ribbonstatus text, OUT i_mediastatus smallint, OUT t_mediastatus text, OUT i_error smallint, OUT t_error text, OUT i_batterystatus smallint, OUT t_batterystatus text, OUT i_remainprint integer) RETURNS SETOF record

Get printer status.

Input Parameter

t_server A text to specify unique printer identifier[2].

Output Parameter

i_printerstatus A smallint indicating printer status code.

t_printerstatus A text explaining printer status code.

Printer Status	Description
0	Standby
1	Waiting for dispensing
2	Analyzing
3	Printing
4	Offline
5	Error

i_bufferstatus A smallint indicating buffer status code.

t_bufferstatus A text explaining buffer status code.

Buffer Status	Description
0	Buffer available
1	Buffer near full
2	Buffer full

i_ribbonstatus A smallint indicating ribbon status code.

t_ribbonstatus A text explaining ribbon status code.

Ribbon Status	Description
0	Ribbon present
1	Ribbon near end

2 See Configuration Table on page 4.

Ribbon Status	Description
2	No Ribbon
3	Direct thermal model

i_mediastatus A smallint indicating media status code.

t_mediastatus A text explaining media status code.

Media Status	Description
0	Media present
2	No media

i_error A smallint indicating error code.

t_error A text explaining error code.

Error	Description
0	Online
1	Offline
2	Machine error
3	Memory error
4	Program error
5	Setting information error (FLASH-ROM error)
6	Setting information error (EE-PROM error)
7	Download error
8	Parity error
9	Over run
10	Framing error
11	LAN timeout error
12	Buffer error
13	Head open
14	Paper end
15	Ribbon end
16	Media error
17	Sensor error
18	Printhead error
19	Cover open error
20	Memory/Card type error
21	Memory/Card read/write error
22	Memory/Card full error
23	Memory/Card no battery error
24	Ribbon saver error
25	Cutter error
26	Cutter sensor error
27	Stacker full error
28	Command error
29	Sensor error at Power-On

Error	Description
30	RFID tag error
31	Interface card error
32	Rewinder error
33	Other error
34	RFID control error
35	Head density error
36	Kanji data error
37	Calendar error
38	Item No error
39	BCC error
40	Cutter cover open error
41	Ribbon rewind non-lock error
42	Communication timeout error
43	Lid latch open error
44	No media error at Power-On
45	SD card access error
46	SD card full error
47	Head lift error
48	Head overheat error
49	SNTP time correction error
50	CRC error
51	Cutter motor error
52	WLAN module error
53	Scanner reading error
54	Scanner checking error
55	Scanner connection error
56	Bluetooth module error
57	EAP authentication error (EAP failed)
58	EAP authentication error (time out)
59	Battery error
60	Low battery error
61	Low battery error (charging)
62	Battery not installed error
63	Battery temperature error
64	Battery deterioration error
65	Motor temperature error
66	Inside chassis temperature error
67	Jam error
68	SIPL field full error
69	Power off error when charging

Error	Description
70	WLAN module error
71	Option mismatch error
72	Battery deterioration error (notice)
73	Battery deterioration error (warning)
74	Power off error
75	Non RFID warning error
76	Barcode reader connection error
77	Barcode reading error
78	Barcode verification error
79	Barcode reading error (verification start position abnormally)

i_batterystatus A smallint indicating battery status code.

t_batterystatus A text explaining battery status code.

Battery Status	Description
0	Normal
1	Battery near end
2	Battery error

i_remainprint An integer indicating remaining number of print.

Returns

SETOF record output is retrieved through output arguments, e.g.
SELECT * FROM sato.status('sato1');

Throws

printer server ... is not found: the unique printer identifier is not registered in configuration table.

3.2. sato.print

sato.print(IN t_server text, VARIADIC j_jobs json[], OUT b_rfidwritesuccess boolean, OUT t_err text, OUT t_epc text, OUT t_tid text) RETURNS SETOF record

Send print command.

Input Parameter

t_server A text to specify unique printer identifier[3].

j_jobs Json variadic to specify a series of print commands.

Output Parameter

b_rfidwritesuccess A boolean indicating RFID recording successfulness. Only relevant if j_jobs contains RFID recording command[4] with validation turned on.

3 See Configuration Table on page 4.

4 See Error: Reference source not found on page Error: Reference source not found.

t_err	A text explaining RFID recording successfulness. Only relevant if j_jobs contains RFID recording command with validation turned on.
t_epc	A text indicating data read from EPC area. Only relevant if j_jobs contains RFID recording with validation turned on. The text can be compared with the one specified in RFID recording command for verification.
t_tid	A text indicating data read from TID area. Only relevant if j_jobs contains RFID recording command with validation turned on.

Returns

SETOF record — output is retrieved through output arguments, e.g.

```
SELECT * FROM sato.print('sato1',
'{ "cmd":"qty","arg":1}',
'{ "cmd":"text",
        "arg":{"body":"12345678",
                "font":{"pitch":0.1,
                        "file":"SATO0.ttf",
                        "style":1,"wd":2.0,"ht":3.0
                }
        }
}'
);
```

Throws

printer server ... is not found: the unique printer identifier is not registered in configuration table.

invalid command ...: invalid value for "cmd" json key. Please refer to sato.print command below.

printer error ...: printer is not ready.

3.2.1. sato.print command

Print command is in json format with two keys: **cmd** and **arg**: {"cmd":..., "arg":....}.

y

Set vertical position for printing.

cmd	y
arg	decimal number specifying position in mm unit. With dpi 203 and maxvert 20000 dots[5], valid range is 0 to 2502.46 mm.
throws	"vertical position is out of range": position is outside valid range.
example	{"cmd":"y", "arg":2.0}

x

Set horizontal position for printing.

5 See Configuration Table on page 4.

cmd	x
arg	decimal number specifying position in mm unit. With dpi 203 and maxhorz 832 dots[6], valid range is 0 to 104.10 mm.
throws	"horizontal position is out of range": position is outside valid range.
example	{"cmd":"x","arg":2.0}

origin

Set vertical and horizontal position for printing.

cmd	origin
arg	json with y and x keys for vertical and horizontal position, respectively. Values are decimal numbers in mm unit. Valid values for y and x are the same as explained above.
throws	"invalid origin command argument": arg in command json is null. "invalid origin command argument, y is not found": y is null. "invalid origin command argument, x is not found": x is null. "vertical position is out of range": y is outside valid range. "horizontal position is out of range": x is outside valid range.
example	{"cmd":"origin","arg":{"y":2.0,"x":8.0}}

mediasize

Set printing media size.

cmd	mediasize
arg	json with ht and wd keys for height and width, respectively. Values are decimal numbers in mm unit. With dpi 203, maxlabelheight 20000 dots and maxlabelwidth 832 dots[7], valid range is 0 to 2502.46 mm and 0 to 104.10 mm for height and width, respectively.
throws	"invalid mediasize command argument": arg in command json is null. "invalid mediasize command argument, ht is not found": ht is null. "invalid mediasize command argument, ht is out range": ht is outside valid range. "invalid mediasize command argument, wd is not found": wd is null. "invalid mediasize command argument, wd is out range": wd is outside valid range.
example	{"cmd":"origin","arg":{"ht":42.0,"wd":64.0}}

text

Print text.

cmd	text
arg	json with keys: body: text to be printed. font: json with keys: pitch: decimal specifying font gap in mm. Maximum physical dimension is 99

6 See Configuration Table on page 4.

7 See Configuration Table on page 4.

dots. With dpi 203[8], valid range is 0 to 12.38 mm.

file: text specifying filename for font set. Use built-in font filename.

Built-in font filename for CL4NX/CL6NX model is:

Filename	Description
SATOCGSleek.ttf	Sato CG Sleek
SATOCGStream.ttf	Sato CG Stream
SATOOCRA.ttf	Sato OCR-A
SATO0.ttf	Sato 0
SATOALPHABC.ttf	Sato Alpha Bold Condensed
SATOBETABI.ttf	Sato Beta Bold Italic
SATOFOLIOB.ttf	Sato Folio Bold
SATOFUTURAMC.ttf	Sato Futura Medium Condensed
SATOGAMMA.ttf	Sato Gamma
SATOOCRB.ttf	Sato OCR-B
SATOSANS.ttf	Sato Sans
SATOSERIF.ttf	Sato Serif
SATOSYM.ttf	Sato Symbol Set
SATOVICA.ttf	Sato Vica
SATOWING.ttf	Sato Wingbats

style: integer specifying style, 0: standard, 1: bold, 2:italic, 3:bold+italic.

wd: decimal specifying font width in mm. Minimum physical dimension is 20 dots. With dpi 203[9], it will be set to minimum of 2.50 mm automatically if less than the limit.

ht: decimal specifying font height in mm. Minimum physical dimension is 20 dots. With dpi 203[10], it will be set to minimum of 2.50 mm automatically if less than the limit.

throws "invalid text command argument": arg in command json is null.
"invalid text command argument, body is not found": text body is null.
"invalid text command argument, font is not found": font is null.
"invalid text command argument, font pitch is not found": font pitch is null.
"invalid text command argument, font pitch is out of range": font pitch is outside valid range.
"invalid text command argument, font file is not found": font file is null.
"invalid text command argument, font style is not found": font style is null.
"invalid text command argument, font wd is not found": font wd is null.
"invalid text command argument, font ht is not found": font ht is null.

example
```
{ "cmd":"text",
    "arg":{"body":"1234ABCD",
            "font":{"pitch":0.1,
```

8 See Configuration Table on page 4.
9 See Configuration Table on page 4.
10 See Configuration Table on page 4.

"file":"SATO0.ttf",
"style":1,
"wd":2.0,
"ht":3.0
}
}
}

qr

Print QR code.

cmd	qr
arg	json with keys: correctlevel: single character specifying correction level code: L, M, Q, or H (7%, 15%, 25% or 30%, repectively). cellsize: integer specifying size of one side of cell in dot, valid range is 0 to 99. body: text to be encoded.
throws	"invalid qr command argument": arg in command json is null. "invalid qr command argument, correction level is not found": correctlevel is null. "invalid qr command argument, correction level is out of range": correctlevel is outside valid set L, M, Q, H. "invalid qr command argument, cell size is not found": cellsize is null. "invalid qr command argument, cell size is out of range": cellsize is out of valid range. "invalid qr command argument, body is not found": body is null.
example	{ "cmd":"qr", "arg":{"correctlevel":"Q", "cellsize":4, "body":"1234ABCD" } }

dm

Print Datamatrix code.

cmd	dm
arg	json with keys: shape: text, "A" for square and "B" for rectangle. height: integer specifying datamatrix height in mm. body: text to be encoded.
throws	"invalid datamatrix command argument": arg in command json is null. "invalid datamatrix command argument, shape must be defined": shape is null. "invalid datamatrix command argument, shape must be A (square) or B(rectangle)": shape shoule be "A" for square or "B" for rectangle. "invalid datamatrix command argument, height must be defined": height is null.

"invalid datamatrix command argument, height must be > 0": height must be > 0.
"invalid datamatrix command argument, cell size is out of range": cell size (calculated based on height in mm and printer dpi) is outside valid range 1 to 99, inclusive.
"invalid datamatrix command argument, body is not found": body is null.
"invalid datamatrix command argument, body is empty": body is empty string.
"invalid datamatrix command argument, body length (N) exceeds maximum of (N)": body string length exceeds maximum size, 64 character for square shape and 22 for rectangle shape.

example

```
{ "cmd":"dm",
"arg":{"shape":"A",
        "height":15,
        "body":"1234ABCD"
        }
}
```

bmp

Print bitmap image.

cmd: bmp

arg: text specifying hexadecimal-encoded image. Maximum bytearray length is 99999 (or 199998 hexadecimal-encoded text length). Please note that only monochrome with 1 bit per pixel bitmap is correctly printed.

throws: "invalid bmp command argument, bitmap size it out of range": bitmap image size is outside valid range.

example

```
{ "cmd":"bmp",
"arg":"424db6000000000000003e000000280000001e000..."
}
```

An easy way to convert bitmap image represented in bytearray in PostgreSQL is using built-in function:
SELECT encode(<bitmap-bytearray>, 'hex');
If bitmap is stored in file then read it using pgreadfile[11] (please ensure that current user running PostgreSQL service has read persmission to the file):
SELECT encode(pgreadfle(<text specifying filepath>) 'hex');

qty

Set printing quantity.

cmd: qty

arg: integer specifying number of labels to print. Maximum value is 999999.

throws: "qty command argument is out of range": quantity is outside valid range.

example: {"cmd":"q", "arg":1}

feed

11 See pgreadfile on page 2.

Forward-feed printing media.

cmd	feed
arg	json with keys: height: decimal specifying label height in mm. qty: integer specifying number of label feed. Multiplication of height in dots and qty should be within minfeed and maxfeed[12]. With dpi 203, minfeed 48 dots, maxfeed 1600 dots and qty 1, minimum and maximum height are 6.00 mm and 200.19 mm, respectively.
throws	"invalid feed command argument": arg in command json is null. "invalid feed command argument, height is not found": height is null. "invalid feed command argument, qty is not found": qty is null. "invalid feed command argument, feed length is out of range": multiplication of height in dots and quantity is outside valid range.
example	{ "cmd":"feed" ,"arg":{"height":42, "qty":1 } }

backfeed

Backward-feed printing media.

cmd	backfeed
arg	json with keys: height: decimal specifying label height in mm. qty: integer specifying number of label backfeed. Multiplication of height in dots and qty should be within minbackfeed and maxbackfeed[13]. With dpi 203, minbackfeed 48 dots, maxbackfeed 480 dots and qty 1, minimum and maximum height are 6.00 mm and 60.05 mm, respectively.
throws	"invalid backfeed command argument": arg in command json is null. "invalid backfeed command argument, height is not found": height is null. "invalid backfeed command argument, qty is not found": qty is null. "invalid backfeed command argument, feed length is out of range": multiplication of height in dots and quantity is outside valid range.
example	{ "cmd":"backfeed", "arg":{"height":42, "qty":1 } }

uhfwrite

12 See Configuration Table on page 4.

13 See Configuration Table on page 4.

Record EPC data to UHF-type RFID tag.

cmd	uhfwrite
arg	json with keys: data: text specifying EPC data to be recorded. The text will be automatically left-padded with zeroes so that its length is multiplies of 8. validate: boolean specifying if validation is required. If this flag is set to true then sato.print function will set output arguments indicating RFID recording status.
throws	"RFID UHF data must be >=8 and <=124": The text length after automatic left-padded with zeroes is outside valid range.
example	{ "cmd":"uhfwrite", "arg":{"data":"1234ABCD" ,"validate":true } }

Chapter 4. Example

Print Text, Bitmap, Datamatrix and Record RFID

```
1    select * from sato.print(
2    'sato1'
3    ,'{ "cmd":"mediasize","arg":{"ht":42, "wd":64}}'
4    ,'{ "cmd":"qty","arg":1}'
5    ,'{ "cmd":"origin","arg":{"y":1.0, "x":16.0}}'
6    ,'{ "cmd":"dm","arg":{"shape":"A","height":15,"body":"1234ABCD"}}'
7    ,'{ "cmd":"origin","arg":{"y":1.0, "x":38.0}}'
8    ,'{ "cmd":"text","arg":{"body":"1234ABCD", "font":
     {"pitch":0.1,"file":"SATO0.ttf","style":1,"wd":2.0,"ht":3.0}}}'
9    ,'{ "cmd":"origin","arg":{"y":4.0, "x":40.0}}'
10   ,'{ "cmd":"text","arg":{"body":"ABCD-1234", "font":
     {"pitch":0.1,"file":"SATO0.ttf","style":1,"wd":2.0,"ht":3.0}}}'
11   ,'{ "cmd":"origin","arg":{"y":9.0, "x":33.0}}'
12   ,('{ "cmd":"bmp","arg":"' ||
     encode(pgreadfile('/home/postgres/logo.bmp'), 'hex'::text) || '"}')::json

13   ,'{ "cmd":"uhfwrite","arg":{"data":"1234ABCD","validate":true}}'
14   );
```

Line number 2: set 'sato1' as printer unique id.

Line number 3: set label size to 42 mm width and 64 mm height.

Line number 4: set print quantity to 1.

Line number 5 and 6: set origin at 1 mm vertical and 16 mm horizontal then print 1234ABCD as datamatrix.

Line number 7 and 8: set origin at 1 mm vertical and 38 mm horizontal then print 1234ABCD as text.

Line number 9 and 10: set origin at 4 mm vertical and 40 mm horizontal then print ABCD-1234 as text.

Line number 11 and 12: set origin at 9 mm vertical and 33 mm horizontal then print monochrome bitmap from /home/postgres/logo.bmp filepath.

Line number 13: write 1234ABCD as EPC code to UHF-type RFID tag with validation turned on.

Alphabetical Index