autolabel

Autolabel Communication Protocol v1.0.7

Abstract

Autolabel devices are built as a set of modules that communicate via events or messages. The Autolabel Communication Protocol (ACP) gives other applications the ability to communicate with Autolabel devices through a similar mechanism.

Version

1.0.7	2015-02-27	Added MARKER_SEQUENCE_STATUS
1.0.6	2014-12-03	Updated MARKER_SYSINFO
1.0.5	2014-10-27	Corrected rendering info.
1.0.5	2014-09-26	Added MARKER_SYSINFO_GET
1.0.4	2014-09-16	Added MARKER_SYSINFO
1.0.3	2014-08-14	Added FPGA_RD_DATA_REQ and FPGA_WR_DATA_REQ_TIME_MS signals
1.0.2	2013-10-22	Added abort command for applicators.
1.0.1	2013-09-11	Added MARKER_APPLICATOR_STATUS_GET Added MARKER_INHIBIT Added MARKER_INHIBITED
0.9.9	2013-04-09	Updated example json and netstrings to reflect updated protocol behavior.
0.9.8	2012-09-25	Added belt and blow vac applicators to MARKER_APPLICATOR_TYPE.
0.9.7	2012-09-06	Support for sending weight in ACP message
0.9.6	2012-05-22	Added MARKER_APPLICATOR_STATUS Added "applicator_stopped" and "label_error" alarms. Added pallet applicator type to MARKER_APPLICATOR_TYPE.
0.9.5	2012-03-20	Added settings for TT (Thermal Transfer)
0.9.4	2011-12-13	Added DONE message.
0.9.3	2011-10-20	Added MARKER_ALARM_RESET message.

0.9.2	2011-10-17	Added MARKER_APPLICATOR_MODE_SET message.
0.9.1	2011-10-03	Added MARKER_SETTING_GET message.
0.9c	2011-08-24	Updated MARKER_ALARM message.
	2011-08-12	MARKER_APPLICATOR_TYPE added. List of settings updated with range and default values.
	2011-05-17	Removed the “caching” parameter and separated static and dynamic data for the “render” command.
	2011-05-11	Added MARKER_LABEL_PRINTED message.
	2011-05-10	Added “layout_id” format for selecting layouts from internal SqLite database.
0.9b	2011-04-18	Added configuration use case. Added documentation on sample client.
0.9	2011-04-11	Reworked rendering signals.

Status

This document is a draft and is subject to change without prior notice.

Table of Contents

Introduction

The Autolabel Communication Protocol (ACP) is typically used for integrating Autolabel products in larger control systems. It provides mechanisms for receiving status notifications and performing different operations on the Autolabel devices.

ACP is built on a set of open technologies to simplify development, debugging and maintenance.

Protocol Definition

The protocol is based on netstring wrapped json encoded messages.

json

Events are sent as a UTF-8 encoded json string. See json.org for further information.
Example:
When the print head is opened, the MARKER_STATE event is sent with the new state as parameter. This translates to the following json string:

```
{"sig": "MARKER_STATE", "par": "alarm"}
```

netstring

Each json object is wrapped in a netstring to ensure that complete messages are received. The syntax is: `[len]":"[string],"` where len is the length of the string.
See http://cr.yp.to/proto/netstrings.txt for further information.
Example:

```
44:{"par": "online", "sig": "MARKER_STATE_SET"},
45:{"par": "offline", "sig": "MARKER_STATE_SET"},
```

TCP Server

Each Autolabel product that support ACP implements a TCP server on port 57829 which allows three simultaneous clients.

Protocol Behavior

When a client connects to the server, the server begins by transmitting a series of events which makes it possible for the client to grasp the state of the server. It is optional for the client to handle these events. The actual events that are saved between sessions and transmitted on connection may be set by the JSON_PERSISTENT_ADD signal. It is possible to add signals to listen to using the JSON_FORWARDED_ADD signal.

The protocol does not contain any explicit acknowledge mechanism to ensure that a message has been received. Instead, it is the responsibility of the client to listen for events that indirectly gives this information. E.g. if a printout has been requested by a client, it may then listen for the FPGA_RD_PRINT_TRIG_FIRED event as a notification that a print out has been started.

Use Cases

Here, some common use cases are listed. For all use cases, an active connection to the ACP server is assumed.

Display status of an Autolabel device

The user wants to display the status of an Autolabel device.

- For each received MARKER_STATE signal, display the parameter value.

Start/stop an Autolabel device

The user wants to control the state of an Autolabel device.

- Send the MARKER_STATE_SET signal with the parameter set to 'online' or 'offline'.

Configure an Autolabel device

The user wants to setup and configure an Autolabel device through ACP.

- Send the MARKER_SETTING_SET signal with the wanted parameters.

Print labels with static information

The user wants to print labels that are identical. No internal counters or dynamic date/time variables are available.

Preparation

1. Send the MARKER_COMMAND_DO signal with render command and also supply layout and optional data set for variable substitution.
2. Wait for the MARKER_RENDERER_DONE signal until a printout is allowed.

For each label

- Trig any number of labels.

Print labels with dynamic information

The user wants to print labels that have information that may change for each label.

Preparation

1. Send the MARKER_COMMAND_DO signal with render command, layout and optional data set for partial variable substitution.
2. Wait for the MARKER_RENDERER_DONE signal until the preparation phase is completed.

For each label

1. Send the MARKER_COMMAND_DO signal with render command, no layout, and with dynamic data for the final variable substitution.

Use case	layout	static	dynamic	printable	previewable

Static bitmap rendering with preview. Used for batch printing.	layout	static		true	true
Prepare for session with same layout but dynamic data. Preview should not be printable.	layout	optional	optional	false	true
Render dynamic label in prepared session.	-		dynamic	true	false

Signals received from the ACP server

This section describes the signals published in the marker. Note that all signals are not sent on the json channel by default. To subscribe to signals that are not sent by default, please use the JSON_FORWARDED_ADD signal.

MARKER_API_VERSION

par = version string, e.g. "1.0.1"

This signal is sent on connection with a new client.

MARKER_IN_POSITION*

par = is in position "0" or "1"

Signalling if the printer is in its correct printing position.

*This is only available if the marker is mounted on a stand equipped with sensors indicating home position.

MARKER_LABEL_PRINTED

par = <dictionary with variables>

This message is sent after each printed label. It contains the data set used for rendering the bitmap. This signal is published when the label is printed but before the label has been applied (see also signal DONE).

The following special fields are added to the data set. If one of these field is defined in the supplied data set, it is set to that value. This can be used e.g. to reset the label counter.

“__render_id”	Integer that is incremented for each rendered bitmap. Not updated for static labels or when printing a copy with the print button.
“__label_id”	Integer that is incremented for each printed label since start of the system.

MARKER_RENDERER_DONE

par = 0

Published when a successful rendering has been completed and the marker is ready to trig/print.

MARKER_SETTING_CHANGED

par = [setting, value]

This setting is published when a setting is changed. It may be subscribed to by different drivers to invoke a suitable response. See also MARKER_SETTING_SET for a full list of settings.

MARKER_STATE

par = state

'unknown', 'online', 'offline', 'alarm', 'starting', 'adjusting'

This signal is published when the printer's state is changed, and on connection with a new client.

MARKER_ALARM

par = {“alarms”: <list with active alarms>, “warnings”: <list with active warnings>, "alarms_wait_ack": <list with alarms that require manual reset>, "warnings_wait_ack": <list with alarms that require manual reset>}

This signal is published whenever an alarm or warning is added or removed. Below are lists of common alarms and warnings but other strings may appear in certain custom installations.

Alarm	**Meaning**
“paper_out”	Paper out detected.
“no_air_pressure”	The air pressure is below the set limit.
“printhead_open”	The print head is open.
“rewinder_full”	The rewinder for backing paper is full.
“layout_not_found”	The selected layout is not found in the internal database.
“rendering_error”	There was an error when rendering the layout.
“ribbon_out”	Out of thermal transfer ribbon
“applicator_error”	An error was detected when applying the label.
"applicator_stopped"	Pallet applicator stop button is active.
"label_error"	Label stuck/label lost detected when applying the label.

Warning	**Meaning**

“paper_low”	Paper low detected.
“low_air_pressure”	Air pressure below set limit.
“ribbon_low”	Ribbon low detected.
“rewinder_almost_full”	The rewinder for backing paper is almost full.
“trig_outside_window”	The trigger was activated outside the print window.
“trig_during_print”	The trigger was activated during print.
“trig_without_data”	The trigger was activated but no dynamic data was available.
“trig_open_head”	The trigger was activated with the print head open.
“trig_righter_not_ready”	The trigger was activated but the righter was not ready.
“trig_applicator_not_ready”	The trigger was activated but the applicator was not ready.
“missing_trigger”	Missing apply trig within the specified time after print trig.

Example:

{"sig": "MARKER_ALARM", "par": {"alarms": ["printhead_open"], "warnings": ["paper_low", "low_air_pressure"], "alarms_wait_ack": [], "warnings_wait_ack": []}

FPGA_RD_DATA_REQ

par = 1

Can be used to request print data just in time for print start. This signal is published when the trig delay has finished and after a subsequent delay of FPGA_WR_DATA_REQ_TIME_MS the printing process starts.

FPGA_RD_PRINT_TRIG_FIRED

par = status

This signal is published when the machine is triggered by a trig input. The parameter describes error conditions, where zero means a successful print start.

par	Meaning
0	Successful printing started
1	Trig outside window

2	Trig during print
3	Trig without data
4	Trig when print head was open
5	Trig when righter was not ready
6	Trig when applicator was not ready
7	Trig when custom logic was not ready
8	Trig when print limit was reached
9	Trig when inhibited (and silent inhibition was not requested)

MARKER_APPLICATOR_TYPE

par = applicator_type

This signal is published on connection and when the applicator type is changed.

par	Meaning
0	No applicator found
1	Blow applicator
3	Pluck applicator
4	Tamp or Wipe applicator
5	Pallet applicator
6	Belt applicator
7	Blow Vac applicator

DONE

par = 0

This signal is published when a print and apply cycle is finished.

MARKER_APPLICATOR_STATUS

par = applicator_status

This signal is published during a print & apply cycle.

par	Meaning
0	For applicators with home sensor (wipe large, pallet): Apply cycle finished. For applicators without home sensor: Applicator returning to home position.
1	Apply cycle active.

For applicators with a home position sensor, bit 5 (decimal value 32) is active when the applicator is not home.

MARKER_INHIBITED

par = <dictionary>

This signal is published when the inhibition status canges. The parameter contains a dictionary where the keys consist of inhibition senders.

MARKER_SYSINFO

par = {
"sw_version": <version>,
"serial_number": <serial_number>,
"fpga_version": <fpga version>,
"printhead_dots": 1280|1920,
"printhead_inches": 4|6
}

This signal is published as a response to a MARKER_SYSINFO_GET request to the ACP server.
Note 1: You need to subscribe to the MARKER_SYSINFO signal to get the response.
Note 2: It may take up to one minute after the printer has started, until you get a response from MARKER_SYSINFO.

MARKER_SEQUENCE_STATUS

par = {"state": <state>, "seq": <seq no>}

This signal is published whenever the sequence status changes.

state	Description
inactive	The system is offline or in alarm
ready	The system is online and ready to start a print and apply sequence
printing	The printer is busy printing a label
printed	The printer is ready with printing
wait_apply	Waiting for an apply trig
applying	The applicator is applying
wait_print	Waiting for the next label in the sequence to be printed (only applicable for multi-label sequences)
done	A print and apply sequence is done
finished	A complete sequence is finished. If a sequence consist of multiple labels, this only occurs when the last label is successfully printed and applied.

Signals written to the ACP server

This section lists signals listened to by the marker.

COUNTER_ATTRIBUTE_SET

par = list with definition of attributes.

The list must consist of three elements: [index, attribute_name, value].

index is the counter number 1-10.

attribute_name is one of

- start
- width
- inc
- copy
- stop
- restart

value must be a string even if it contains a number, e.g. "23"

Example:

```
{"par": [1, "start", "23"], "sig": "COUNTER_ATTRIBUTE_SET"}
```

See the DP specification for more detailed information about the attribute functions.

FPGA_WR_PC_HW_PRINT_OR_APPLY_TRIG

par = 0

FPGA_WR_PC_HW_PRINT_TRIG_WHEN_DUAL_TRIG

par = 0

FPGA_WR_DATA_REQ_TIME_MS

par = <time between DATA_REQ and print start>

Sets time between DATA_REQ signal and print start. A longer time gives the external system and the printer more time to process the rendering time before the actual print start.

JSON_FORWARDED_ADD

par = list of signals to forward on json channel.

Example:
```
{"sig": "JSON_FORWARDED_ADD", "par": ["HARDWARE_SCANNER_DATA"]}
```

JSON_PERSISTENT_ADD

par = list of signals to save and resend between sessions on json channel.

Example:
```
{"sig": "JSON_PERSISTENT_ADD", "par": ["MARKER_IN_POSITION"]}
```

MARKER_COMMAND_DO

par = {'cmd': <command>[, 'args': <argument list>]}

This signal executes a command in the printer.

Command	**Parameters**	**Description**
"abort"		Aborts an application cycle. Currently this is only supported for eTamp applictaors.
“apply”		Performs an application
“print”		Performs a printout
“print_and_apply”		Performs a printout and a subsequent application
“render”	layout, static_data, dynamic_data	Renders a layout and data set. *layout* is an optional layout object. *static_data* and *dynamic_data* are optional

		variable dictionaries.

apply

Performs an application.

print

Performs a printout of the current rendered bitmap.

print_and_apply

Performs a printout of the current rendered bitmap and a subsequent application.

render

Renders a layout with a data set using specified caching and rendering error strategy. When rendering is finished a MARKER_RENDERER_DONE signal is emitted.

Use case	layout	static	dynamic	printable	previewable
Static bitmap rendering with preview. Used for batch printing.	layout	static		true	true
Prepare for session with same layout but dynamic data. Preview should not be printable.	layout	optional	optional	false	true
Render dynamic label in prepared session.	-		dynamic	true	false

layout

This is the JSON schema for the layout object:

```
{
  "name":"Layout",
  "properties":{
    "format":{
      "type":"string",
      "description":"Layout format",
      "required":true
    },
    "data":{
      "type":"string",
      "description":"Layout data",
      "required":true
    }
    "encoding":{
      "type":"string",
      "description":"Layout data encoding",
      "required":false
    }
```

```
  }
}
```
Where format can be one of

Value	Description
"dp"	The data string contains a layout encoded in the Intermec Direct Protocol.
"json"	The data string contains a layout encoded as a json object. Implemented in 20131204-092246 and later.
“layout_id”	The data string contains a layout_id that must exist in the internal SqLite database. When using the layout_id format, the data string must contain the layout id.

and encoding can be one of

Value	Description
"base64"	Base64 encoded data. Use this if layout contains 8-bit characters which are not utf-8 encoded.
"utf8"	Default encoding.

If no layout is specified a previously cached layout will be used in the current rendering.

data

The data is separated in two parameters, “static_data” and “dynamic_data”. Each parameter contains a dictionary of the variables that should be rendered. The static data is substituted once and will become part of the cached layout. The dynamic data is only used for the current rendering.

Expression evaluation

The renderer evaluates expressions in two steps:

Example:
```
{"best_before": "datetime.now()+timedelta(days=duration).strftime('%Y-%m-%d')",
"weight": "str('${w1}')", "duration": 5}
```

The high-level renderer evaluates this to something like this:
```
{"best_before": "2012-12-24", "weight": "${w1}", "duration": 5}
```

The ${w1} expression is evaluated in the low-level renderer.

Weight

If the dynamic data set contains a variable with name "w1_g", it will be treated as if the weight came from a scale.

printable

This is a flag that controls whether to render a printable bitmap.

Value	Description

true	Renders a printable bitmap
false	Prevents bitmap from getting printed. This may be used for labels that require “live” data, and rendering is done for caching a layout.

MARKER_RENDERER_DATA_SET

par = null | dictionary of variables

Updates the current data set used for rendering the current layout.

Sending this signal with par = null, clears the rendering buffer and the preview. This can be used to ensure that no more copies of the rendering buffer can be printed when using static trig configuration.

Note that the render command is the recommended way to control rendering.

MARKER_SETTING_GET

par=<string>

Request value of a setting. A successful request triggers a MARKER_SETTING_CHANGED event.

MARKER_SETTING_SET

par = {"name": <string>, "value": <value>}

Set the value of a setting.

Name	Description	Default
applicator.apply_delay	Delay between finished printing and start of application (ms). This setting is only used for special applicators. You may want to look at the Trig Delay setting under Trigger instead. [0, 2000]	0
applicator.apply_time	Applicator activation time (ms) [0, 2000]	80
applicator.vacuum_stop_time	Time that vacuum is stopped before application (ms) [0, 2000]	0
config.speed_compensation	Speed compensation in percent. Increase gives less delay. [50, 200]	100
equipment.conveyor1.speed	Conveyor speed (m/min) [10, 60]	22

equipment.printtrigpolarity	Print start trigger polarity when a two stage trigger configuration is used (positive is usually the start of the product) [0 (Positive flank), 1 (Negative flank)]	0
equipment.trigconfig	Select which conditions should print and apply the label [('Print and apply on HW trig', 0), ('Print on righter or SW, apply on HW trig', 1), ('Print and apply on custom CAN', 4), ('Print on custom CAN, apply on HW trig', 5)]	0
equipment.trigpolarity	Print/apply start trigger polarity (positive is usually the start of the product) [('Positive Flank', 0), ('Negative Flank', 1)]	0
equipment.trigtype	Selected trigger type [('Timing from weight', 0), ('Timing from weight and HW trigger', 1), ('Timing from HW trigger (static rendering)', 2), ('Timing from HW trigger (dynamic rendering)', 3)]	3
plc.timer1.active_ms	Active duration (ms) [0,8191]	0
plc.timer1.block_ms	Block duration (ms). During this time all activations are blocked. [0,8191]	0
plc.timer1.delay_ms	Delay time from valid trig to active output (ms) [0,8191]	0
plc.timer1.enable	Enable or disable the timer block [0, 1]	1
plc.timer1.trig_filter_ms	Filter time for trig input (ms) [0,8191]	5
plc.timer1.trig_polarity	Trig input active edge [('Positive Flank', 0), ('Negative Flank', 1)]	0
plc.timer2.active_ms	Active duration (ms) [0,8191]	0
plc.timer2.block_ms	Block duration (ms). During this time all activations are blocked. [0,8191]	0
plc.timer2.delay_ms	Delay time from valid trig to active output (ms) [0,8191]	0
plc.timer2.enable	Enable or disable the timer block [0,1]	1
plc.timer2.trig_filter_ms	Filter time for trig input (ms)	5
plc.timer2.trig_polarity	Trig input active edge [('Positive Flank', 0), ('Negative Flank', 1)]	0
print.winlen_mm	Valid length where printing must start (mm)	100

	[0, 1000]	
print.winstart_mm	Required distance from trigger and print start (mm) [0, 2000]	100
printer.airpressure_alarm_level	Lowest allowed air pressure before alarm [0, 4000]	3500
printer.blowfanspeed	Blow applicator fan speed [0, 99]	50
printer.contrast	Determines the darkness of the printout [100, 600]	350
printer.labellength	Controls how long to search for a label gap [30, 240]	140
printer.labelwidth	Controls the torque for the rewinder. [1, 180]	40
printer.media_pwm_offset	Media sensor PWM offset value [-1000,1000]	0
printer.media_sensor_mode	Use dark for black mark and light for normal labels [('Light', 0), ('Dark', 1)]	0
printer.orientation	Selected label rotation [0, 90, 180, 270]	0
printer.printspeed	Printing speed in mm/s [20, 500]	160
printer.reverse_delay	Delay between finished printing and start of reversing (ms) [0, 2000]	120
printer.rewinder_enable	Turn off when printing without rewinder [0, 1]	1
timing.blowdelay	Time between start of printing to blow apply start (ms) [0, 5000]	0
timing.print_or_apply_trig_filter_time	Filter time for printer hardware input (ms) [0, 8191]	5
timing.trigblocktime	Time window in which new HW triggers are blocked (ms) [0, 5000]	0
trig.delay_base_mm	Delay distance between trigger and print start (mm) [0, 1000]	100
printer.tt_enable	Enable thermal transfer motor [0, 1]	0
printer.tt_forward_torque	Forward torque of thermal transfer motor [0, 100]	50
printer.tt_backward_torque	Backward torque of thermal transfer motor [0, 100]	10

printer.tt_stretch_duration	Duration that ribbon should be stretched after reversing (ms) [0, 1000]	500
printer.tt_ribbon_out_steps	Number of steps without movement before ribbon out is detected [600, 4800]	1000
printer.print_limit_enable	Limit number of printouts [0, 1]	0
printer.print_limit_count	Set the number of allowed printouts when print limit is enabled [1, 1000]	1
printer.dispensing_feed	Set the how much paper that is fed out after a label is printed for proper dispensing (steps) [0, 240]	198
printer.reverse_feed	Set the amount of reverse feed before printing the next label to position the start of the label at the printhead (steps) [0, 240]	210
printer.tracking	Set the position of the inner media guide (mm) or 0 for centered media. [0, 180]	0

MARKER_STATE_SET

par = new_state

'online', 'offline'

This signal sets the printer's state if allowed. If successful, a MARKER_STATE signal is emitted from the printer.

MARKER_APPLICATOR_MODE_SET

par = new_mode
This signal sets the mode of the applicator. It is currently only used for the pallet applicator.

Value	Description
1	Pallet: 0 degrees (linear) application
2	Pallet: 90 degrees (left) application

MARKER_APPLICATOR_STATUS_GET

par = 0 (not used)

This signal forces the last value of applicator status to be sent.

MARKER_ALARM_RESET

par = 0 (not used)

This signal resets all active alarms in the printer. Note that alarms that are reset auomatically by user actions are not affected by this command i.e. paper out and head lifted.

RT_RENDER

par = 0

Force rerendering of current layout and data set.

MARKER_INHIBIT

par = {"sender": <string>, "inhibit": true|false[, "silent": true|false]}

The sender can be any string. It is used to identify the source of an inhibition. The same sender is used to clear the inhibition (inhibit=false).

The silent parameter is optional and default false. A silent inhibition does not cause any warnings when a trigger occur during an active inhibition.

MARKER_SYSINFO_GET

par = null

Request a MARKER_SYSINFO notification.

Barcodes

List of supported barcodes

EAN13
EAN8
CODE128
GS1-128
PDF417
QR-CODE
DATAMATRIX
GS1-DATABAR
GS1-DATABAR-EXP

GS1-DATABAR-LIMIT
GS1-DATABAR-STACK
GS1-DATABAR-STACK-OMNI
GS1-DATABAR-EXP-STACK
INT2OF5
UPC-A

EAN13

EAN13 barcodes can contain static or variable data. Different regions may use different schemes for encoding variable data. This list describes the built-in support for some regions. Setting the barcode message string to one of the built in function strings will give a dynamic output when connected to a scale.

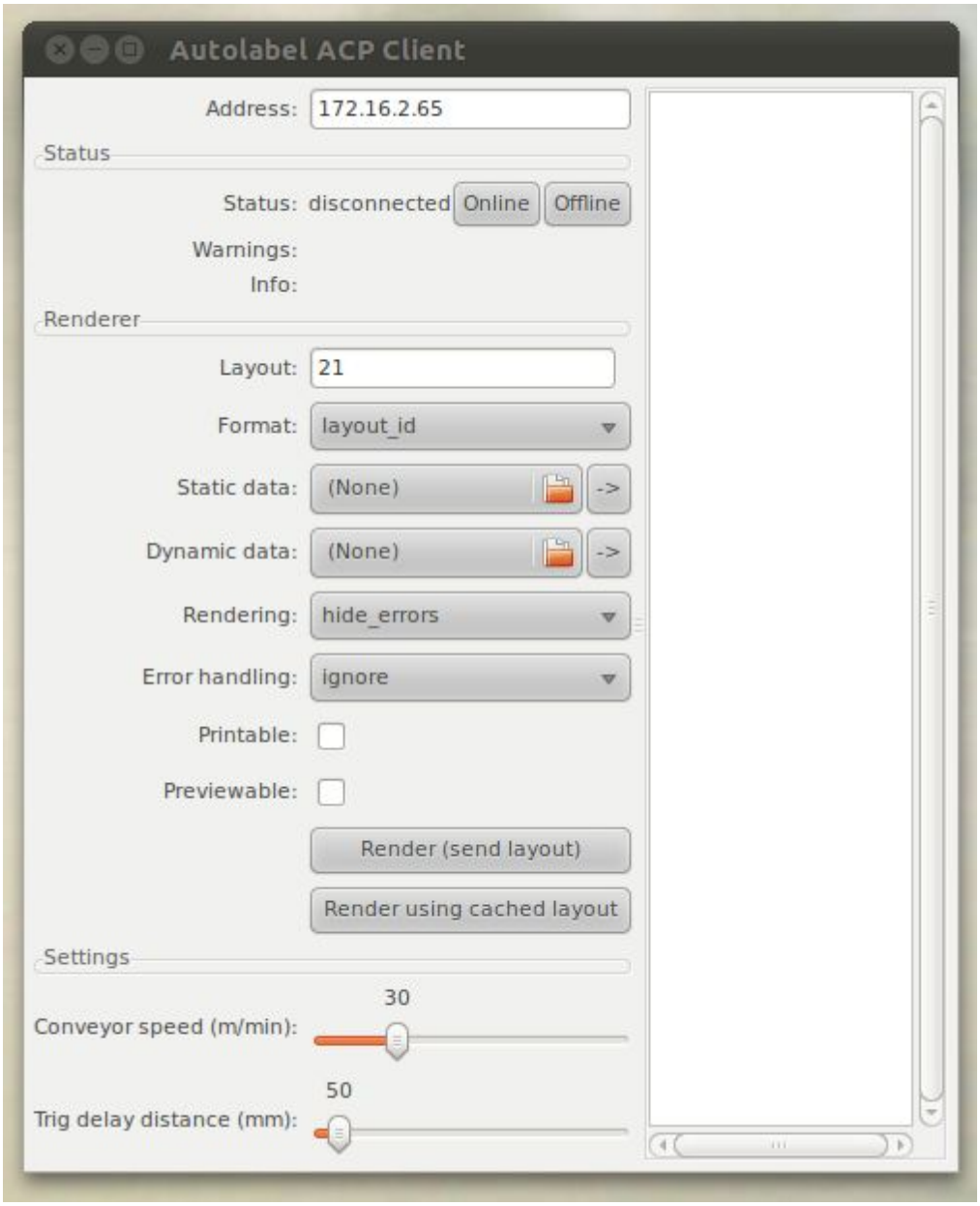
Country	Type	Function	Output
SE (Sweden)	Price	${ean20vpw1,ARTNR,KGPRICE} where KGPRICE is expressed as the price in decimal SEK. Example: ${ean20vpw1,123456,18.39}	2[0-2]AAAAAAPPPP. PPPP is the price expressed with four digits. The second digit may be 0, 1 or 2 and indicates: 0: price is in SEK with 2 decimals 1: price is in SEK with 1 decimal 2: price is in SEK without decimals
SE (Sweden)	Weight	SE23AAAAAA	2[3-5]AAAAAAWWWW. WWWW is the weight expressed with four digits. The second digit may be 3, 4 or 5 and indicates: 3: weight is in kg with 3 decimals 4: weight is in kg with 2 decimals 5: weight is in kg with 1 decimal
DK (Denmark)	Weight	DK26AAAA	26AAAAWWWWWW. WWWWWW is the weight in grams expressed with six digits.
DK (Denmark)	Price	${ean27vpw2,ARNT,KGPRICE} where KGPRICE is expressed as the price in decimal DKK.	27AAAAPPPPPP. PPPPPP is the price in DKK with 2 decimals expressed with six digits.
PL (Poland)	Weight	PL28AAAA	28AAAAWWWWWW. WWWWWW is the weight in grams expressed with six digits.
RU (Russia)	Weight	RU27AAAA	27AAAAVWWWWW. WWWWW is the weight in grams expressed with five digits. V is a checksum.

GR (Greece)	Weight	GR29AAAAA	29AAAAAWWWWW. WWWWW is the weight in grams expressed with five digits.

API Sample Client

The API Sample Client is a Python application that describes techniques for controlling an Autolabel marker. It implements some of the most common use cases:

- Activating a layout from file.
- Rendering the layout with a dataset.
- Displaying status from the marker.
- Change settings



GUI

Address

The first section includes an address field. Type the IP address of the Autolabel device here, and press Enter.

Status

The status indicator displays the device status or disconnected if no successful connection has been established. The application automatically tries to reconnect if the connection is aborted.

The Online/Offline buttons sends commands to set the device online or offline if possible.

The Warnings and Info fields displays active warnings and information messages. *Not yet implemented.*

Rendering

The Layout text box points out a layout file or a layout_id from the built-in database.

The Format drop-down list selects the function of the above Layout control

The Static data selector points out a json file with data which will be statically substituted.

The Dynamic data selector points out a jason file with data which will be dynamically substituted.

The Rendering setting determines how errors should affect rendering.

The Error setting determines how errors should affect status messaging.

Printable and Previewable controls whether it should be possible to print the rendered layout and if a preview should be generated.

There are two versions of the Render button. “Render (send layout)” sends the current layout configuration and data set. “Render using cached layout” omits the layout data and only sends the data set.

Settings

The setting section displays a number of settings that can be transmitted to the device. Note that changing a setting on the device also updates the ACP GUI.