

ACP By Example



# Summary

Quick introduction to Autolabel Communication Protocol including several real-world examples. The intended audience is system integrators that need to be able to control Autolabel devices.

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Comment | Author |
| 1.0.1 | 2014-02-20 | Added information about Unicode support and json layouts | Henrik Bohre |
| 1.0.0 | 2014-01-15 | First version | Henrik Bohre |

# Content

[Summary](#_75h4irya4uz2)

[Revision History](#_lsn1b7byxxha)

[Content](#_3vdmhal3uhxs)

[Before We Start](#_w1h680hlo5h)

[Examples](#_3wdw3te0ai4m)

[Example: Hello Autolabel!](#_paogdh1w0ry2)

[Example: Control online/offline state](#_hzet6t796i81)

[Example: Select a built-in layout and fill with unique data](#_i5ajhfx2fkwc)

[Using unicode characters](#_mu6wiurd09d8)

[JSON Layouts](#_2wbw2d27lbdl)

[DP Layouts](#_oixv8f8lfszt)

# Before We Start

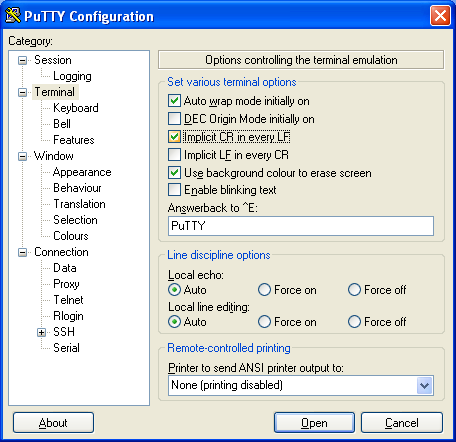
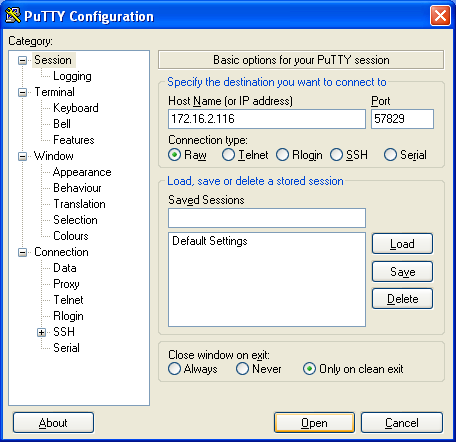
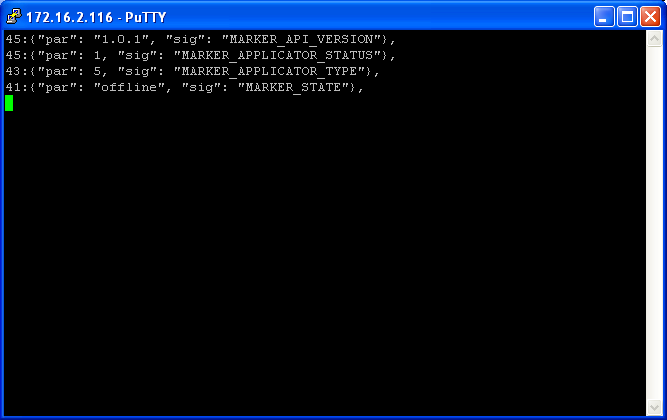
To be able to run the examples, the rest of this guide assumes that you have:

* A networked Autolabel printer or a [virtual printer](https://docs.google.com/document/d/1sPWvOVJRYnuRBgTQ1eFL64QM4BipTKQep_T-ngw8Ao4/edit?usp=sharing&authkey=CPC_pcYG). You need to know the IP address of that printer.
* A networked computer with Windows XP or later.
* [PuTTY](http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe) - a great terminal program (you don't even have to install it, you can just download it and run it).
* Complete knowledge of the [ACP reference manual](https://docs.google.com/document/d/101zEW7FMYmqB1F-VucCmW09RoQmIwyWF-jV7GoyDbCw/edit?usp=sharing&authkey=COPk-v4G). No, just kidding! You probably wouldn't read this guide if that was the case.

# Examples

## Example: Hello Autolabel!

Let's start by connecting to the printer using it's IP address.

1. First, switch to the Terminal settings page, by clicking the Terminal node in the left Category pane. Now select the "Implicit CR in every LF" check box. This is strictly not necessary but will format the information much nicer on the screen. If you save your session later this configuration will be used the next time you use the session.  
   
2. Now, switch back to the Session page.
3. Type in the printer's IP address in the Host Name field. Change the port to 57829 and for Connection type, select Raw. 
4. If you like, you can save the session by entering a name in the Saved Sessions text field and click Save. The next time you can double click the session name in the list to open the session with the saved parameters.
5. In order to connect to the printer, click Open or double click the saved session name.
6. You should now be greeted with a black terminal window

Can you see which API version number the system reports, and which state is in? In that case, congratulations! You have successfully read ACP.

Try setting the printer online, open the printhead etc, and see how state is reflected in the terminal.

## Example: Control online/offline state

Ok, so we have seen that it is possible to observe the status of the printer. What if we want to control the online/offline state?

In the ACP reference you can find the MARKER\_STATE\_SET signal. It gives us the possibility to request a state change, which the printer may or may not be able to execute. If e.g. the printer is in alarm state due to the print head being open, it is not possible to set the printer online.

The syntax for setting the printer online is:

{"par": "online", "sig": "MARKER\_STATE\_SET"}

As you may have noticed, all messages received from the printer begins with a number and a colon, and ends with a comma. This is the [netstring](http://cr.yp.to/proto/netstrings.txt) format, which ensures that complete messages are received.

So, in order to send an ACP command for setting the printer online, we need to wrap the message like this:

44:{"par": "online", "sig": "MARKER\_STATE\_SET"},

Since offline is one character longer than online, we get:

45:{"par": "offline", "sig": "MARKER\_STATE\_SET"},

If you copy one of these lines and paste them into PuTTY, you shall be able to set the printer online and offline. You will also see the state change as before.

## Example: Select a built-in layout and fill with unique data

So, let's move on to a more compelling use case. Our goal is to activate a built-in layout and fill in the variable data for that layout.

We assume that you have a connection to the printer with PuTTY and that you have understood how netstrings work.

The command below will activate layout\_id=7 and populate the variables using a dictionary.

199:{"par": {"cmd": "render", "args": {"dynamic\_data": {"w1": "42"}, "layout": {"data": 7, "format": "layout\_id"}, "printable": true, "static\_data": {}, "previewable": true}}, "sig": "MARKER\_COMMAND\_DO"},

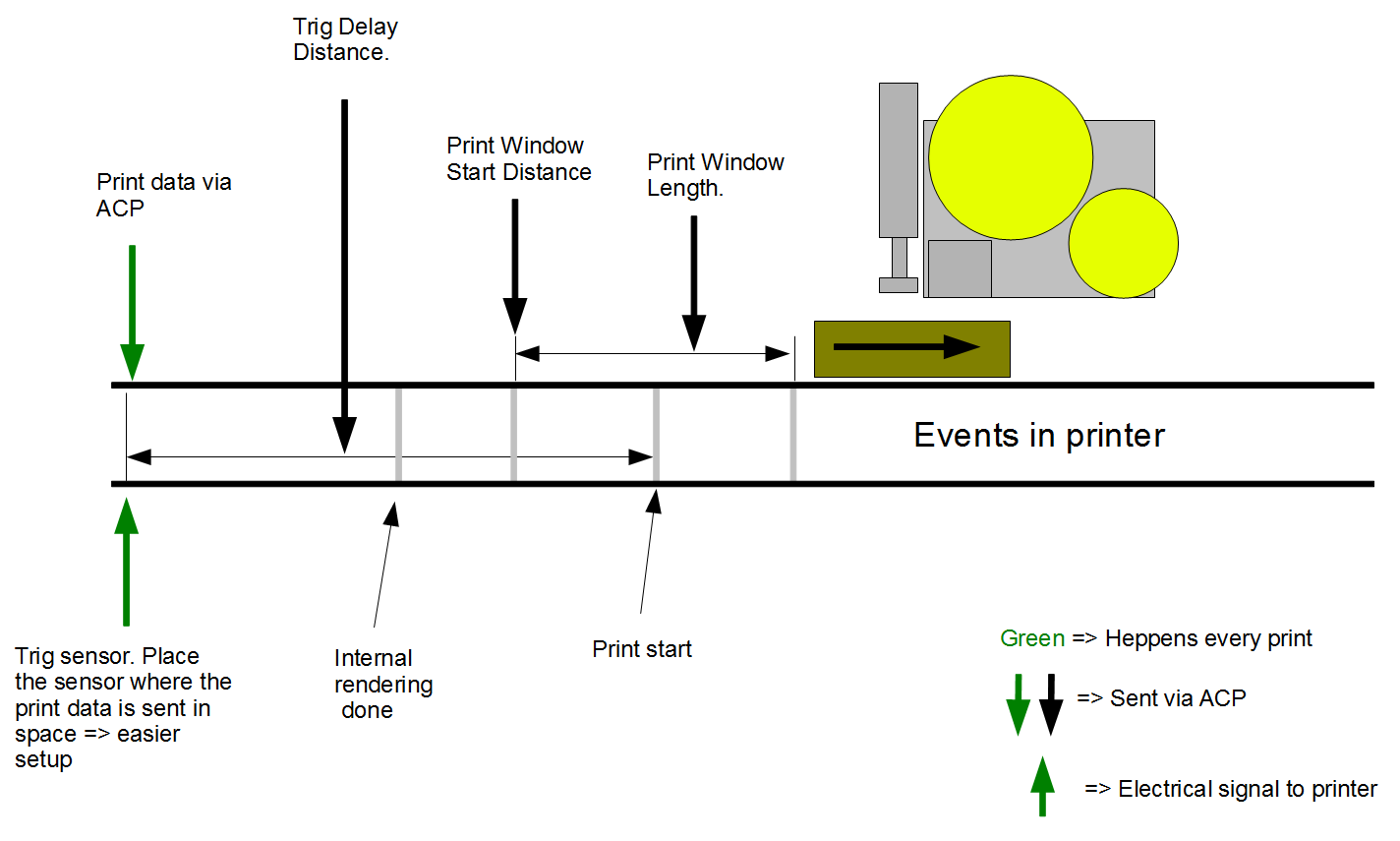
(Note: If you copy this string, ensure that you don’t get any extra line feeds).

Note that the weight is 42 in the image below.  


If you want to try out other values on the w1 variable, please ensure that you update the netstring length as well. If you do more extensive changes, you can use the online [netstring creator](https://docs.google.com/spreadsheets/d/16to6nHDIdQYvLF_EjKLMLeQ65Uq298OGRVnsyFdztNg/edit?usp=sharing). Note that no validation on the payload occurs in this tool. It only wraps your message in a netstring.

## Example: Subscribe to signals 62:{"sig": "JSON\_FORWARDED\_ADD", "par": ["MARKER\_LABEL\_PRINTED"]},

## Example: Sending unique variable data for every print

This is an example on how to set up WPL-labelling system where the variable data is sent via ACP/LAN but the the sensor is connected to the printer and the logic to handle the timing is placed in the printer. Timing windows are used in the printer, starting when data reaches the printer if print is started inside the open timing window the data is printed, if not the data is deleted. 

For an easy setup do the following:

* Use the setting “Setup->Installation->Print&Apply->Trigger->Trigger Type” = “Timing from weight and HW trigger”.
* Use the setting “Setup->Installation->Print&Apply->Trigger->Trigger Configuration” = “Print and apply on HW trig”.
* Write the correct value to setting “Conveyor Speed” via ACP or at “Setup->Installation->Pheripherals->Conveyor”
* Place the trig sensor where the print data is sent to the printer.
* Keep the setting “Setup->Installation->Print&Apply->Trigger->Trigger Delay Distance” over a distance that equals 300 ms. At 60 m/min this is 300 mm.
* A 90 mm long label printed at 200 mm/s will be applied 700 ms after start of print with a tamp applicator. This means that the time between the trig sensor and the point of label application is 300 (min) + 700 ms = 1000 ms = 500 mm at 30 m/min and 1000 mm at 60 m/min.
* Set the “Print Window Start Distance” to the same value as “Trig Delay Distance” minus a margin of around 150 mm. The setting is found at :”Setup->Installation->Print&Apply->Printer”.
* Set the “Print Window Length” to around 300 mm. The setting is found at :”Setup->Installation->Print&Apply->Printer”.

## Using unicode characters

Unicode characters are represented by \uHHHH escape sequences or UTF-8 encoded strings in json.

Example: ÅÄÖ

271:{"par": {"cmd": "render", "args": {"dynamic\_data": {"w1": "\u00c5\u00c4\u00d6"}, "layout": {"data": 7, "format": "layout\_id"}, "printable": true, "static\_data": {}, "rendering": "hide\_errors", "error\_handling": "ignore", "previewable": true}}, "sig": "MARKER\_COMMAND\_DO"},

Any JSON encoder/decoder shall be able to handle this.

C# has the built-in [json serializer](http://msdn.microsoft.com/en-us/library/bb412179.aspx) and the [Json.NET](http://james.newtonking.com/json) library.

## JSON Layouts

Layouts can be included in the render command. The example below shows how to send a json layout which contains a text field with two static lines and a third variable line.

784:{"par": {"cmd": "render", "args": {"dynamic\_data": {"weight": "19", "w1": "19", "variable here": "Three words here"}, "layout": {"data": {"fields": [{"line\_spacing": 125, "spans": [{"fontref": "1", "fontsize": 7.79527559055, "text": "One\nTwo words\n${variable here}"}], "y": 158, "x": 407, "rotation": 0, "type": "text", "alignment": 5}, {"line\_width": 1, "height": 100, "width": 300, "y": 123, "x": 262, "rotation": 0, "type": "rect", "alignment": 7}], "fontdef": {"1": "/usr/share/fonts/truetype/freefont/FreeSansBold.ttf"}, "height": 461, "width": 803, "y": 0, "x": 0, "rotation": 90, "autolabel\_layout\_version": 1}, "format": "json"}, "printable": true, "static\_data": {}, "rendering": "hide\_errors", "error\_handling": "ignore", "previewable": true}}, "sig": "MARKER\_COMMAND\_DO"},

## DP Layouts

Since DP Layouts may contain PCX images and other binary data, the actual payload is base64 encoded. This example includes UTF-8 encoded strings in the layout:

653:{"par": {"cmd": "render", "args": {"dynamic\_data": {"weight": "18", "w1": "18"}, "layout": {"encoding": "base64", "data": "U0VUVVAgIk1FRElBLE1FRElBIFNJWkUsV0lEVEgsMTI4MCIKU0VUVVAgIk1FRElBLE1FRElBIFNJ\nWkUsTEVOR1RILDYwMCIKQ0xMClBQMTAwLDUwCkFOMQpQVCAiSGVsbG8gd29ybGQuIFRoZSB3ZWln\naHQgaXMgJHt3ZWlnaHR9IgpOQVNDICJVVEYtOCIKUFAxMDAsMTAwClBUICJIZWogdsOkcmxkZW4i\nCk5BU0MgIlVURi04IgpQUDEwMCwxNTAKUFQgIkhlaiB2w6RybGRlbiDDpcOkw7YiCk5BU0MgOApQ\nUDEwMCwyMDAKUFQgIkhlaiB2w6RybGRlbiDOsc+DzrTPhiIKUEYK\n", "format": "dp"}, "printable": true, "static\_data": {}, "rendering": "hide\_errors", "error\_handling": "ignore", "previewable": true}}, "sig": "MARKER\_COMMAND\_DO"},

The source DP file looks like this:

SETUP "MEDIA,MEDIA SIZE,WIDTH,1280"

SETUP "MEDIA,MEDIA SIZE,LENGTH,600"

CLL

PP100,50

AN1

PT "Hello world. The weight is ${weight}"

NASC "UTF-8"

PP100,100

PT "Hej världen"

NASC "UTF-8"

PP100,150

PT "Hej världen åäö"

NASC 8

PP100,200

PT "Hej världen ασδφ"

PF

SETUP "MEDIA,MEDIA SIZE,WIDTH,1280"  
SETUP "MEDIA,MEDIA SIZE,LENGTH,600"  
CLL  
PP100,50  
AN1  
PT "Hello world. Weight:";/${weight}/  
NASC "UTF-8"  
PP100,100  
PT "Hello world"  
NASC "UTF-8"  
PP100,150  
PT "Hej world"  
NASC 8  
PP100,200  
PT "Hej world"  
PF