

# *Well-formed printer data streams*

Version 1.2 (20.04.2005)

Dipl. - Ing. Joachim Deußen

- 
- ① Overview
  - ② UEL – Universal Exit Language
  - ③ PJI – Printer Job Language
  - ④ PCL – Printer Command Language
  - ⑤ Postscript
  - ⑥ Well formed print jobs
- 

Copyright © 2005 by Dipl.-Ing. Joachim E. Deußen. All rights reserved.  
All trademarks belong to their respective owners.

## 1 Overview

Well formed? No, this is not something about beautiful bodies in a high-gloss magazine. This is all about data structures, protocols and data streams. Let me present three examples to you and then decide for yourself:

### HTML:

```
<html ><body><p><font face="Century Gothic">Di pl . -l ng. <br>Joachi m E. DeuBen</font></p><p><font face="Century Gothic">Konrad-Adenauer-Ri ng 41</font></p><p><font face="Century Gothic">41747 Vi ersen</font></p><p><font face="Century Gothic">+49-172-21 29 285</font></p><p><font face="Century Gothic"><a href="http://www.joachimdeussen.de">www.joachimdeussen.de</a><br><a href="http://www.joachim-deussen.de">www.joachim-deussen.de</a><br><a href="http://www.cyrion-technologies.de">www.cyrion-technologies.de</a><br><a href="http://www.cyrtech.de">www.cyrtech.de</a></font></p></body></html >
```

```
<html >
<body>

<p><font face="Century Gothic">Di pl . -l ng. <br>Joachi m E. DeuBen</font></p>
<p><font face="Century Gothic">Konrad-Adenauer-Ri ng 41</font></p>
<p><font face="Century Gothic">41747 Vi ersen</font></p>
<p><font face="Century Gothic">+49-172-21 29 285</font></p>
<p><font face="Century Gothic">
<a href="http://www.joachimdeussen.de">www.joachimdeussen.de</a><br>
<a href="http://www.joachim-deussen.de">www.joachim-deussen.de</a><br>
<a href="http://www.cyrion-technologies.de">www.cyrion-technologies.de</a><br>
<a href="http://www.cyrtech.de">www.cyrtech.de</a></font>
</p>

</body>
</html >
```

### HPGL/2:

```
I NWUPWO, OPWO, 1LTO, 10, 1LTSP1TD1L021CFPP1
```

```
I N
WU
PWO, 0
PWO, 1
LTO, 10, 1
LT
SP1
TD1
L021
CF
PP1
```

### (Visual) BASIC:

```
Public Function FileExists(ByVal FileName As String, Optional mode As VbFileAttribute = vbNormal) As Boolean
Dim tempstr As String
FileExists = False
If FileName <> "" Then
If Right(FileName, 1) = "\" Then FileName = Left$(FileName, Len(FileName) - 1)
tempstr = Dir(FileName, mode)
If tempstr <> "" Then If InStr(UCase(FileName), UCase(tempstr)) > 0 Then FileExists = True
End If
End Function
```

```
Public Function FileExists(ByVal FileName As String, _
Optional mode As VbFileAttribute = vbNormal) As Boolean
    Dim tempstr As String
    FileExists = False
    If FileName <> "" Then
        If Right(FileName, 1) = "\" Then FileName = Left$(FileName, Len(FileName) - 1)
        tempstr = Dir(FileName, mode)
        If tempstr <> "" Then
            If InStr(UCase(FileName), UCase(tempstr)) > 0 Then FileExists = True
        End If
    End If
End Function
```

Now, what do you think? The second form is much more readable, right? But nevertheless both forms represent in all languages a perfectly executable and valid data stream.

How come? In the ancient days of computing CPU cycles, memory and bandwidth were a precious good and so they were very expensive. So the designers of a data stream, protocol or program defined their data to either be very compressed or to be very readable. They used the readable form, when they wanted to test their software (for instance a printer driver with PCL/HPGL output) but in the final product, they wanted to use as few memory and bandwidth as possible.

So the most data streams are not very human readable. But by the time we got to learn, that readability was not something valued only in design but also in troubleshooting a system. So when memory and bandwidth and CPU cycles become cheaper and cheaper it came into focus to have the data streams not compressed beyond readability but leave them readable for the sake of troubleshooting.

Adobe was the first company to discover this fact. With all the programmers and postscript driver writers coming to them and asking for help in their postscript output, they decided that some convenient structure in all those postscript data would be much convenient for them. So they invented the DSC – Document Structuring convention. About that a little bit later.

The situation got more complicated when printer vendors started to integrate more than one printer language into their printers. Most of the time this was a second personality like Postscript that extended the build-in PCL, ESC/P or Prescribe.

A common problem with more than one personality is automatic language switching: How should the printer know how to interpret the next command? To what language/personality does it belong?



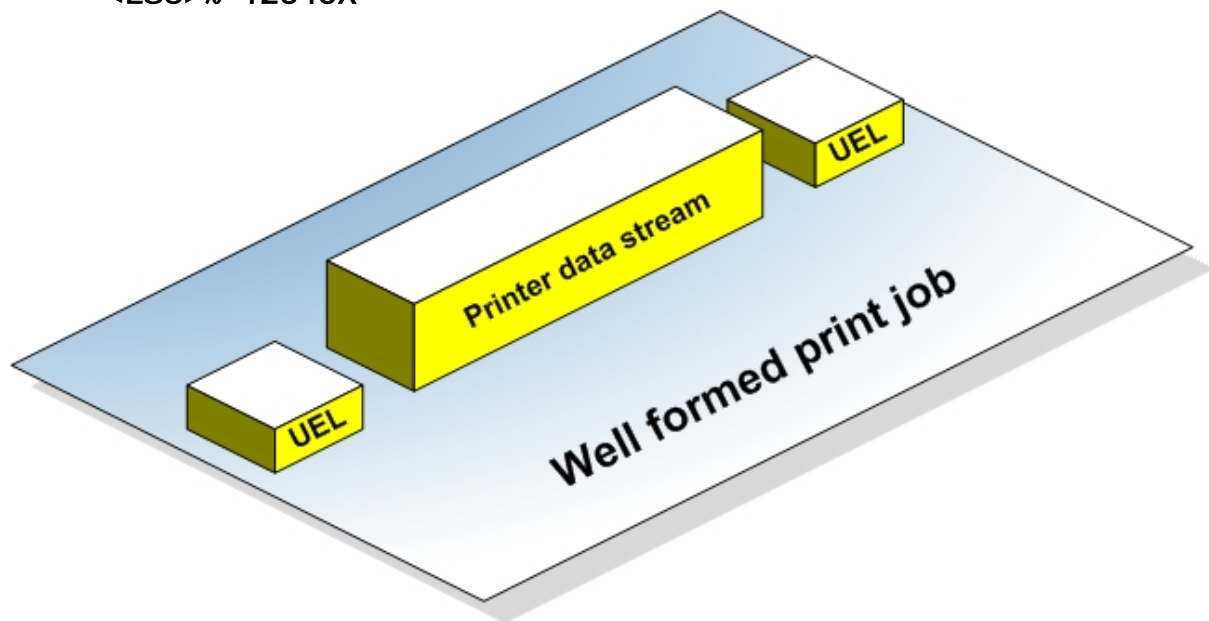
A well formed print job helps the printer's language interpreter a lot in determine what to do next.

## ② UEL – Universal Exit Language

Hewlett-Packard started first a more structured approach to this topic, by defining a workflow (for your programmers out there: a state machine) for PCL data streams.

The first topic was the **UEL - Universal Exit Language**. This “new” printer language has exactly one command: The Reset command.

`<Esc>%-12345X`



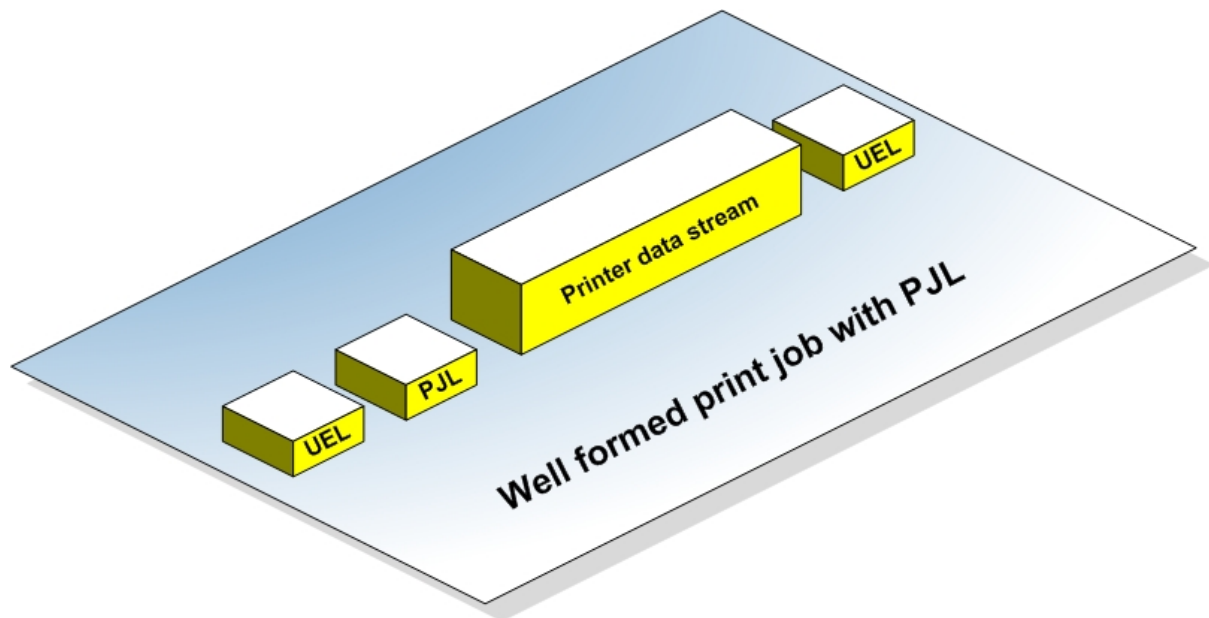
This command often is also referred to as UEL. Its sole purpose is to reset the printer in it's entirely. The printer should finish all processing in any language and return to an idle state where it ways for new data on one of its interfaces.

- ➔ A print job should start and end with this UEL command to reset the printer in a defined state.

Today almost all modern printers implement this language. From this point on i.e. after the UEL command, the printer can enter each personality.

### 3 PJP – Printer Job Language

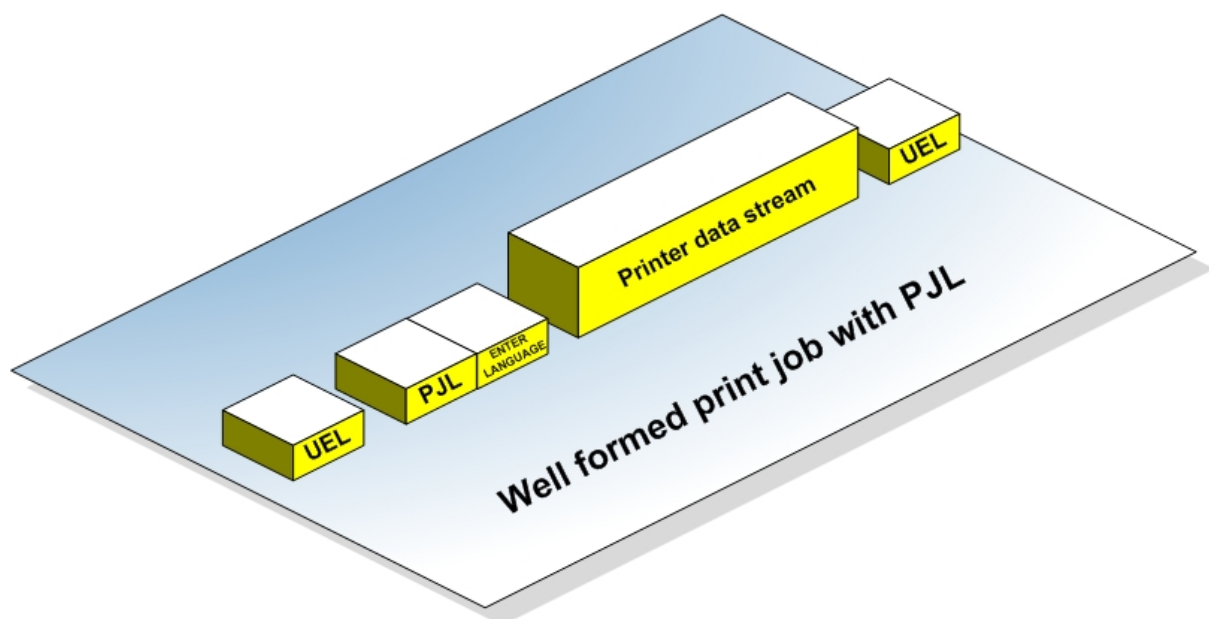
Hewlett-Packard defined a second additional language for PCL, the so called printer job language. This command language is used to set all job related parameters rather than page related parameters as a language like PCL or ESC/P do.



PJP can only be entered right after a UEL command, so sometimes you find the UEL command language described in the frame of the PJP command sequences.



So a "well-formed" printer data stream thus should start and end with an UEL command where the first UEL command should be followed by PJP commands if they are needed!



As stated above automatic language switching is a problem for the printer. From the UEL command on every personality could be entered. Also every personality (except PjL itself) could be entered from PjL on.

To guide the interpreter a little bit the PjL language defines a command that must be used as the last valid PjL command:

**@PjL ENTER LANGUAGE = <personal i ty>**

The <personality> is defined by each printer, but common values are "PCL" or "POSTSCRIPT". If the PjL interpreter sees this command, it knows, that from now on PCL or Postscript will follow and it instruct the automatic language switcher to change to that personality immediately.

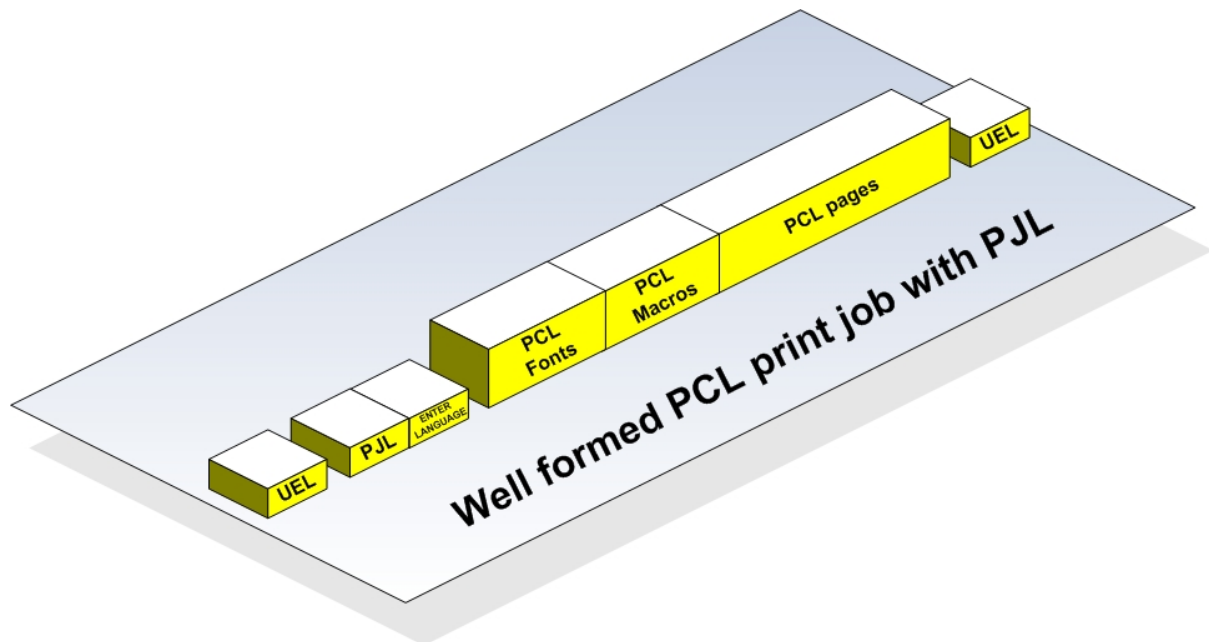


A well formed print job should have at least one PjL command: **PjL ENTER LANGUAGE** to help the printer in finding the correct personality.

## 4 PCL – Printer Command Language

PCL code is normally a mixture of PCL and HPGL/2 code. But nevertheless you should stick to some easy rules, when creating a PCL data stream.

If you need to use any **recurring elements**, like fonts, patterns or macros, you should define them prior to drawing on the first page. The language itself does not imply any rules, when a font or macro is created and used, but it is certainly good programming style to define everything first and then use it, rather than define it just the moment it is needed.



When a page is finished you normally eject it by using the FORMFEED character 12hex. But, if you issue some other commands also a page ejection occurs. These other commands include the PRINTER RESET command, the page size or page orientation change for instance.

If you use one of these commands to do two things at the same time, for instance changing from A4 to A5 and ejecting the current page, this is a very **bad programming style**, because it relies on a side effect of the original command.

### Example:

...	PCL code
<FORMFEED>	→ Eject Page 1
<PAGE SIZE=A4>	<i>As long as nothing is on the page it is not ejected</i>
...	PCL code
<PAGE SIZE=A5>	→ Eject Page 2
...	PCL code
<FORMFEED>	→ Eject Page 3
...	PCL code

Imagine what will happen, if the designers of the PCL interpreter change they behaviour? The command will change the next page size after a FORMFEED now:

...	PCL code
<FORMFEED>	→ Eject Page 1
<PAGE SIZE=A4>	<i>As long as nothing is on the page it is not ejected</i>
...	PCL code
<PAGE SIZE=A5>	<b>Change Size from next page on!</b>
...	PCL code
<FORMFEED>	→ Eject Page 2
...	PCL code

As you can see you now get only two pages instead of one. This is because the designer relied on the page ejection of the PAGESIZE command, which is gone now. So more correctly this code would look like this:

...	PCL code
<FORMFEED>	→ Eject Page 1
<PAGE SIZE=A4>	<i>As long as nothing is on the page it is not ejected</i>
...	PCL code
<FORMFEED>	→ Eject Page 2
<PAGE SIZE=A5>	<i>As long as nothing is on the page it is not ejected</i>
...	PCL code
<FORMFEED>	→ Eject Page 3
...	PCL code

Now it is not important anymore if the PAGESIZE command ejects a page or not, because it is already ejected by the FORMFEED and regardless what happens, the PAGESIZE command does not eject a page, when nothing is on the page.



Do not rely on side effects of the used PCL interpreter! If you use the same code on a different printer you may get different results.

## 5 Postscript

Postscript by concept is a true programming language and not a descendent of pure text printers. So by design there is already some structure in this printer data stream.

But as you can see above in the Visual Basic example the same program, code can look totally different. So for trouble shooting reasons Adobe created a definition how to structure postscript code. This definition is called Document Structuring Convention.

```
%-12345X@PJL JOB
@PJL ENTER LANGUAGE = POSTSCRIPT
%!PS-Adobe-3.0
%%Title: Testseite
%%Creator: PScript5.dll Version 5.2
%%CreationDate: 7/23/2004 2:12:10
%%For: klaatu
%%BoundingBox: (atend)
%%DocumentNeededResources: (atend)
%%DocumentSuppliedResources: (atend)
%%DocumentData: Clean7Bit
%%TargetDevice: (LANIER LD238c PS3) (3011.103) 2
%%LanguageLevel: 3
%%EndComments
...
%%EOF
%-12345X@PJL EOJ
```

In the above example you can see the start and the end of a well formed Postscript print job:

1. UEL command
2. @PJL JOB and @PJL ENTER LANGUAGE commands
3. Postscript according to DSC 3.0
4. UEL command
5. @PJL EOJ command

A Postscript data stream adhering to DSC 3.0 ever always starts with the phrase "%!PS-Adobe-3.0" (A line starting in postscript with the percent sign designates a comment. So this line is actually a comment for the postscript interpreter).

Defining and explaining DSC is beyond the scope of this document and you can find many hints and tips for postscript either by Adobe (<http://www.adobe.com/>) the inventor of the Postscript language itself or at Acumen Training (<http://www.acumentraining.com/>), a site by one of the gifted trainers I have ever meet.

The DSC documentation itself can be downloaded from Adobes page as [http://partners.adobe.com/asn/developer/pdfs/tn/5001.DSC\\_Spec.pdf](http://partners.adobe.com/asn/developer/pdfs/tn/5001.DSC_Spec.pdf)

## 6 Well formed print jobs

Now we have everything together for well formed print jobs:

1. UEL to embrace the whole printer data stream
2. PJI as the first language for job-related parameters
3. The last PJI command should be @PJI ENTER LANGUAGE=<...>
4. The actual print job language

### PCL5:

- If you have a PCL print job this job should first start with all generic definitions like macros patterns and fonts.
- Each page should be ended with a form feed
- Side effects like PRINTER RESET <Esc>E for ejecting a page should be avoided at any time.
- Check boundaries (non-printable area) to avoid bad effects on other printers
- Check the font looks and behaviour on forms and documents, to avoid using substitutes

### Postscript:

- Postscript job should stick to the Adobe Document Structuring convention DSC 3.0. This make live easier when troubleshooting Postscript.