

Graduation Report

V4 Printer Drivers & Cloud Integration

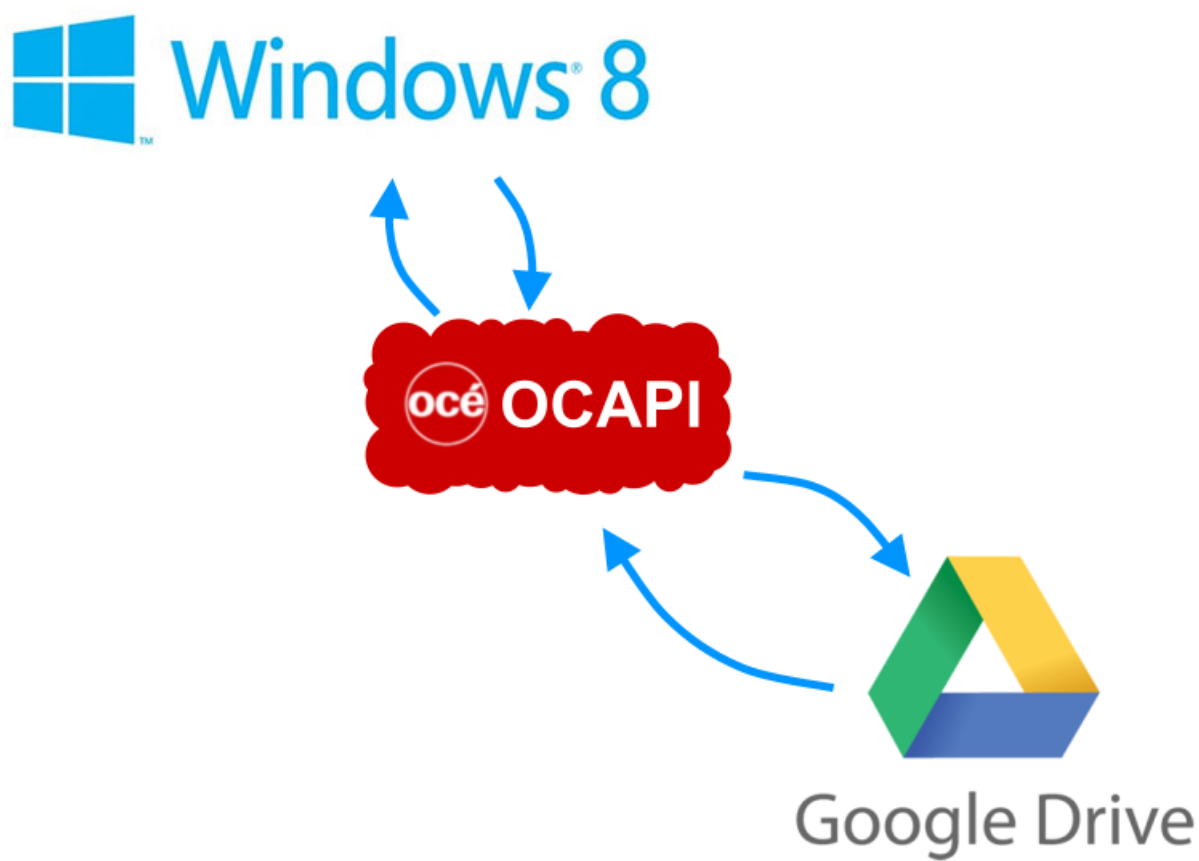


Table of content

Foreword	4
Summary	5
Glossary	6
1 Introduction	8
2 Creating a V4 Printer Driver	9
2.1 Introduction	9
2.2 Architecture	9
2.3 Test case requirements and pipeline configuration design	10
2.4 Implementation of the filters	11
2.4.1 Print Ticket Parser Filter	11
2.4.2 JDF Insert Filter	11
2.4.3 Configuration files	11
2.4.4 Implementation challenges	11
2.5 Testing the driver	12
2.6 Conclusions and recommendations	12
3 Enhancing with PrinterExtensions	14
3.1 Testing with the Microsoft samples	14
3.2 Background information	15
3.3 Modifying the Sample	15
3.4 Conclusions	16
4 Metro style Apps	17
4.1 Coupling the sample	17
4.2 Background information	18
4.3 Modifying the code	18
4.3.1 Getting SNMP Data	18
4.3.2 Devices & Print Slide-Out	19
4.3.3 Support from Microsoft	19
4.3.4 Progress halted	20
4.4 Changes in Metro & Windows 8	20
4.5 Conclusions	20
5 Researching the Cloud	22
5.1 Collecting in-house sources	22
5.2 Research delineation	23



5.3	The real research	23
5.3.1	What research had been done?	23
5.3.2	What technologies can be used?	23
5.3.3	What existing cloud services can be used?	24
5.3.4	What interaction should there be with the cloud service?	24
5.3.5	Where do we integrate the cloud?	24
5.3.6	What do competitors offer?	25
5.3.7	Can creating our own cloud service help in integrating more services?	25
5.3.8	How can we build our own cloud service?	25
5.3.9	What infrastructure should be used?	25
5.4	Findings and conclusions	26
6	Designing a Proof of Concept Cloud Solution	27
6.1	The Concept	27
6.2	Direct or indirect cloud access	27
6.3	Choosing the service	27
6.4	OCAPI	28
6.5	The Design	28
6.6	The Implementation	30
6.6.1	OCAPI Service	31
6.6.2	OCAPI Client	31
6.6.3	The combined result	32
6.7	Conclusions	32
7	Integrating the Cloud	33
7.1	Designing the integration	33
7.2	Triggering upload	34
7.3	Integrating OCAPI Client into the PrinterExtension	35
7.4	Problems Arise	35
7.5	Further development steps	36
7.6	Conclusions	36
Sources & Installable Binaries		37
First V4 Driver (Consumer Preview Version)		37
Desktop PrinterExtension & Metro Device App		37
Cloud Concept		38
Cloud Integration		38
Evaluation		39
Sources		40
Appendices		41



Canon
CANON GROUP

Foreword

This document is the graduation report of Maurice Wingbermhle, student ICT & Technology at Fontys University of Applied Sciences. During a period of 20 weeks, I completed my graduation on V4 printer drivers and cloud integration at Océ Technologies B.V. located in Venlo, The Netherlands.

The main reasons that I chose for this assignment are the flexibility of the assignment and the focus of the assignment on my graduation. Océ does not pressure me to develop or research anything; they just want full access to the results of my research and products that I might develop. Océ Technologies B.V. is a company with a high technological profile and together with Canon, they are a leading company in the printing industry.

The subject of this report is about the new V4 Printer Driver model, introduced in Windows 8 and Windows Server 2012, and integrating cloud services into a new printer driver. Because Fontys required me to do a decent research, the cloud integration part of the assignment would be ideal as a research subject. Especially because of the environment of the integration, the printer driver, makes this research interesting. For as far as we know it is a world first attempt of combining the new V4 printer driver technology with cloud technology.

I have learned a lot from my period at Océ and I have enjoyed working on the project. I would like to thank all the people that were involved in any way, and helped me to accomplish my goals. It has been a great experience and a privilege working with them. Special thanks go out to my mentor Johan Hoogendoorn, Gé Kessels, Christian Luijten, Matthieu Helder and Rob Kersemakers for providing me with very helpful clues, support and information for my cloud investigation. I would not have been able to come this far without their help.



Summary

Océ Technologies B.V. is the research & development department of the printer producer Océ N.V., which is a part of Canon Group. Because Océ is part of Canon, Océ gets to focus on wide format and office printers. Before my graduation project started, there already was some knowledge of the V4 printer driver model and the surrounding technologies, some research had been done on cloud integration, but there was no working cloud driver for Metro yet. The research that had been done by Océ employees was accessible for me. There is a need of information about the new V4 architecture and technology, so Océ would be able to assess the impact on their current code legacy. My primary goal was to build a proof of concept to combine all these technologies, and show that it can be done, and how it can be done. I started with creating a basic V4 printer driver, which later on will act as a base to integrate the cloud concept. The printer driver has to comply with a small number of requirements, to be able to demonstrate the driver on Océ printers. After building the printer driver, I started building the UI parts that can enhance the printer driver. I used the samples available on the Microsoft website, modified them and coupled them to the printer driver. The coupling was the hardest step of all because the documentation was not yet complete at the time I made the attempts to couple UI with printer driver. I encountered a bug in a sample, which I reported to Microsoft. Sadly enough the bug vanished while Microsoft was working on the case, so the problem disappeared. That does not mean that it is solved. Next I started my research for cloud integration. I investigated a lot of services, and a lot of aspects of integrating the cloud into a printer driver. I had special focus on networking problems and how to handle / bypass them such as a big corporate network like at Océ. I investigated the possibility of building our own cloud service and what advantages that would have. In the research document I eventually came up with a number of conclusions and advices to help create a concept for cloud integration into a printer driver. With the research complete and a lot of additional knowledge about the cloud, I started designing and building a concept for cloud access, that later will be integrated into the existing printer driver from phase one and two. In the design I combined as much of the advantages that came out of our research as possible. I describe the steps taken in setting up the design, and implementing the client and server side. The implementation of the cloud concept is capable of authenticating and with Google and retrieve a list of files from a personal Google Drive, and with that proves that the concept is working. The next step was to integrate the cloud concept into the printer driver. The main target of the integration is the Metro style device app, and the integration should have been no problem, but because of unexpected and seemingly undetectable errors, I could not continue as I would have wanted. I had to stop investigating the issue because I would have run out of time to finish my report and other documentation, but after a reboot of the Windows 8 environment, the issue was gone and the app was working fine. At the end of the project I can conclude that my cloud concept is the most powerful result of my graduation project. The reason for the success of OCAPI was combining a number of stable, widely used standards to deliver a cloud solution to a cutting edge and sometimes even bleeding edge technology environment. With the project I have successfully displayed the opportunities of the V4 Printer Driver model in combination with the latest cloud technology, and proved that low level cloud integration in the printer driver is not impossible, and it is worth to invest more time into further investigation.

Glossary

Defenition	Meaning / Explanation
.NET	Pronounced "Dot Net". Software framework, developed by Microsoft. Extension to the Windows API.
API	Application Programming Interface. A collection of definitions on base of which applications can communicate with other applications or application parts.
App	Short for application. Popular word and mostly used to refer to an application for a mobile operating system.
ASP.NET	Active Server Pages .NET. Microsoft developed server side application architecture. Running on the .NET framework
C#	Pronounced "C Sharp", Object Oriented programming language originally developed by Microsoft within the .NET initiative. One of the programming languages for the Common Language Infrastructure. Also needs a virtual machine environment to run in.
C++	Pronounced: "C Plus Plus". One of the most versatile and most used programming languages.
DLL	Dynamic Link Library. Software library that can be distributed seperately, or combined with other software. Applications can use these libraries, and might even require them for proper functioning.
DNS	Domain Name System. Hierarchical distributed naming system for computers, services, or any resource connected to the internet or a private network.
exe	Executable, filename extension that indicates that the file is a executable application for Windows.
firewall	Software for protecting a computer from unwanted intruders like computer virusses, hackers, or just unwanted content or behaviour.
HTTP	HyperText Transfer Protocol: Protocol for transmitting text and other internet content from a server to a client.
HTTPS	Secure version of the HTTP protocol (hence the extra S). The HTTP data stream is protected by a SSL encryption.
IaaS	Infrastructure as a Service. Form / type of cloud service that provides infrastructure in the cloud, like virtual machines and virtual networking.
ID	Identifier. ID can also be a shorthand for an identification number or other means of identification.
INF file	Information, or Setup Information file. Plain text file, containing all information needed to install specific software.
InterFilter Communicator (IFC)	Part of the XPSDrv pipeline, that converts and buffers data for pipeline filters.
IP address	Internet Protocol address. Identifier for a host in "the internet".
JDF	Job Defenition Format. Data format for PrintTicket, based on XML. Developed by Adobe.
Metro	Design language, created by Microsoft. First real focus in Windows 8 UI.
OS	Operating System – In this context: Windows 8
PaaS	Platform as a Service. Form / type of cloud service, that provides a development platform in the cloud.
PCL6	Printer Command Language version 6. A PDL developed by Hewlett Pakard (HP)
PDF	Portable Document Format. Developed by Adobe. Standard format for exchange of electronic documents.
PDL	Printer Description Language. Data format for document description, used to communicate to printers.
PostScript (PS)	PostScript. A PDL developed by Adobe.

PrinterExtension	UI extension for a V4 printer driver. Replaces the default Windows Advanced Print Settings dialog.
PrintTicket	Set of (User configured) print settings, specific for one print job.
PropertyBag	Part of the XPSDrv pipeline, capable of storing data for as long as the pipeline is alive.
REST	REpresentative State Transfer. Standardized software architecture. Used for implementations of web and cloud services.
sandbox	A security mechanism to separate running programs. Often used to protect systems from untrusted code.
SDK	Software Development Kit. Package of software tools and libraries to enable software development for a certain platform, API or framework.
SNMP	Simple Network Management Protocol. A protocol for retrieving and setting properties to network devices.
SSL	Secure Socket Layer. Cryptographic protocols for communication security over the internet.
UI	User Interface. A way of letting the user interact with a electronic system.
V4 Printer Driver	A printer driver built according to the specification of the new Microsoft V4 (Version 4) Printer Driver Model.
Visual Studio	Development environment for Windows environments, developed by Microsoft.
WDK	Windows Driver Kit - Software development package for developing Windows Drivers.
Win32	Windows 32bit API. The Windows API to access system resources, built for 32bit operating systems.
Windows	Operating System, developed by Microsoft. Numerous versions have been released, but in this document we mostly talk of the latest version: 8.
Windows Event Viewer	Tool that displays messages, warnings and errors that happen in the Windows system software.
WinRT	Windows Runtime. Actually the same as Windows API, but suitable for ARM and Intel core processors. Windows 8 uses this runtime for all Metro related software.
XAML	eXtensible Application Markup Language. A XML based definition language to describe a UI. Can be programmed by a software developer or generated from a designer tool.
XML	eXtensible Markup Language. Human readable data format.
XMPP	eXtensible Messaging and Presence Protocol. Has its origin as Instant Messaging protocol, but now being adapted for a wider application base. Popular because of its speed. Uses HTTP or HTTPS as data transport protocol.
XPS	XML Paper Specification. A PDL developed by Microsoft.
XPSDrv pipeline	Rendering component of a XPS based printer driver. Software uses the Pipeline Design Pattern, and the filters in the pipeline can be built by printer driver developers. The pipeline itself is built and managed by Microsoft.
zip	File type. Compressed archive which can contain multiple folders and files.



1 Introduction

Windows 8 Consumer Preview and Release Preview, Visual Studio 11 Beta and 2012 Release Candidate and Windows Driver Kit Beta; can it get even more bleeding edge? Sure! Throw in some cloud services! That is where this graduation is about. Using the very latest of technologies to create a proof of concept and discover new possibilities that never have been done before. Learn about every technology, how it works, how to use it, and what happens if we were to use in combination with other new technologies, or even old legacy code.

Océ Technologies B.V. is constantly searching for new technologies and possibilities to enhance their products. The announcement of the coming of Windows 8 was a trigger to investigate the impact that the new Printer Driver Model (version 4) will have on their systems. Combined with the personal interest of Johan Hoogendoorn and rumours of competitors' activities in the cloud, it resulted in a very interesting graduation assignment.

The assignment is to build a V4 printer driver, enhance the driver with PrinterExtensions. After that do a research on cloud services and develop a concept to integrate cloud into the printer driver the best way possible. In order to accomplish all this, I need a great deal of information and knowledge about a variety of subjects, and I will document as much of that information as I can. For more information about the assignment, please see Appendix A: Project Initiation Document.

To give the entire project structure, I made phased planning. This will help to track the process of the project and all the separate deliverables, and gives me and my mentor the means to act fast when a phase is running late. The planning and the complete phase description can also be found in appendix A.

The structure of this document follows my process in fulfilling my assignment. Chapter 2 describes the steps that I have taken to create a basic V4 printer driver, together with bit of background information on the V4 Printer Driver Model. The third chapter is about the first UI expansion that I have built for the printer driver: the PrinterExtension for desktop environment. The fourth chapter is still about the same phase as chapter three, but this time about the Metro style PrinterExtension. I have had a bit more issues with that part, and the chapter contains the complete story on how I tried to resolve the issue with the help of Microsoft. The fifth chapter is about the research that I have done on the cloud. I will just discuss the process of creating the document, and a small summary of the document, and if you want to know more, you will have to read the research document itself. The sixth chapter is about designing and developing a concept for cloud access and intermediate cloud solution, as a preparation on chapter 7, which will describe the steps taken to integrate the concept into the printer driver, and the challenges that I encountered during that phase.

2 Creating a V4 Printer Driver

2.1 Introduction

This chapter belongs to phase two of my planning, where I create my first own printer driver. We need a printer driver to integrate the cloud into, and by starting with the printer driver itself, we learn a lot about the environment of the printer driver and the capabilities that it has.

As a test case vehicle, we need a basic V4 Printer driver, to experiment with. We needed a driver that should actually output to a Prisma-Sync printer from Océ, with a limited subset of functionality and configurable options. This involved reading a lot about the driver architecture, the communication standards in printers and how to build the actual driver. The documents that I have used for this phase can be found in the Sources, marked with reference **MS01** and **MS02**.

To get started with the V4 Printer Drivers, I started with the templates that Microsoft provides for Printer Drivers in the WDK. I've described how to start with the templates in another document (see Appendix B). The reason to do this is that I would always have a safe fallback solution if the implementation would not succeed. This way I always succeed, and the number of features depends on the speed of implementing.

2.2 Architecture

The new V4 architecture uses the XPSDrv pipeline, and Microsoft makes sure that the input for the driver is XPS. Microsoft also provides two converter filters for the pipeline to convert XPS to PostScript or PCL6. These filters are developed by Monotype Imaging, and they licensed the use of the filters to Microsoft and their users. To use those filters, you will have to configure them in a pipeline configuration file, as part of the driver configuration.

In the image below you can see the V4 rendering choices at high level. This is how the framework provides the XPS input for the filter pipeline, and how to get XPS or PCL6 / PS output by using different pipeline configurations. The gray parts are system components that are immutable by us, the printer driver developers. The red parts are only configurable by us, and the blue parts are the filters and configuration files that we can write completely. It is not shown what parts are mandatory and what parts are optional, but we will get to that.

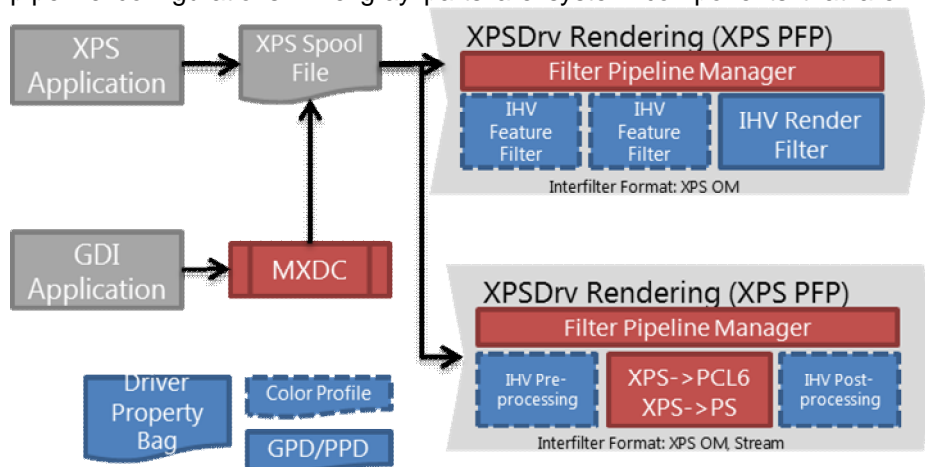


Figure 1 - High level architecture rendering choices (See document MS01, Section 4.1.1)

The XPSdrv pipeline backend is provided by Microsoft, and therefore controlled by the Windows operating system. The lifetimes and starting times of the pipeline and the filters are not within the control of the developer. You need to use the XPSDrv pipeline when you want different PDL or functionality as output for your printer driver. For custom functionality, you have to write a filter, and provide a bit of configuration XML in the pipeline configuration file, to tell the pipeline in what order to execute your filters and what type of in / output your filters expect.

2.3 Test case requirements and pipeline configuration design

So for our basic driver, we need PostScript output prefixed with a JDF ticket to have the Océ printers understand our print job. To fill the JDF ticket with actual data, we have to retrieve some data from the XPS print tickets. The JDF ticket has to be added after the conversion from XPS to PostScript has taken place. Taking this all in account, the pipeline configuration design looks like this:

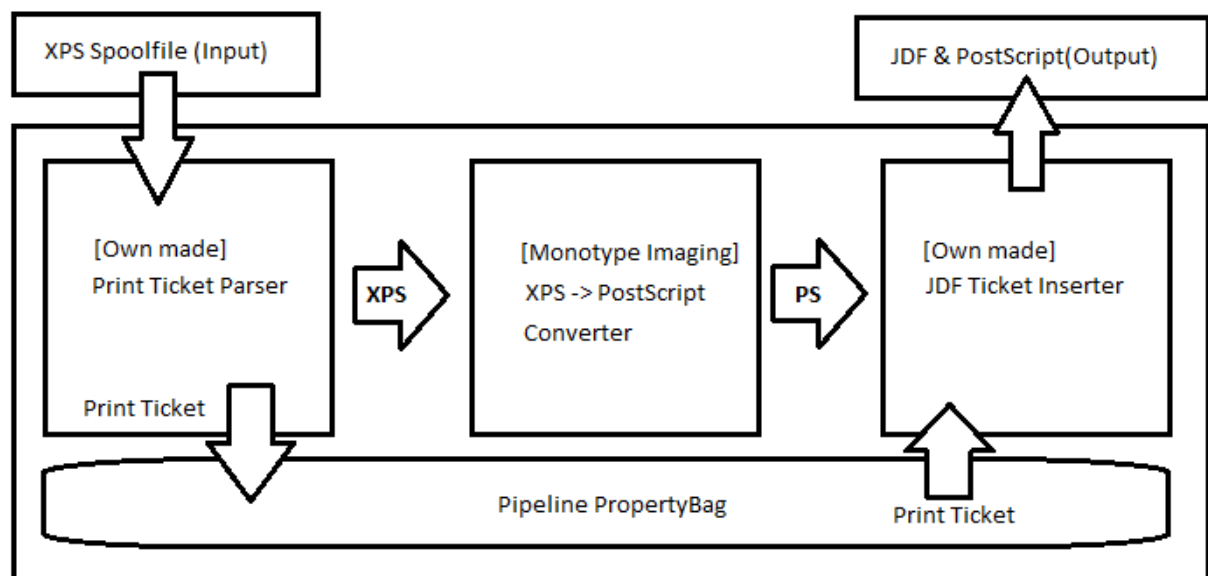


Figure 2 - My pipeline configuration design

The design above is the solution to a problem that is introduced by using the XPS to PostScript conversion filter, which converts the XPS in the pipeline into a other PDL stream. The problem is that after the conversion, you cannot access the print tickets that are in the original XPS stream. A simple solution is to extract the print tickets from the XPS stream before converting it to PostScript, store them in the PropertyBag, and afterwards generate the JDF ticket, and read the needed values from the print tickets. I could also have generated the JDF ticket in the first filter and stored that, instead of the raw print ticket, but this is a more equally distributed approach, and allows us to reuse the existing samples from Microsoft. The first filter extracts and, if needed, merges the print tickets and stores the merged ticket to the pipeline PropertyBag. The last filter reads the print ticket and extracts only specific values, and injects them into the JDF ticket.

To realize this design, we have to create 2 filters of our own, and configure the Monotype XPS PostScript converter filter. Luckily the Microsoft templates for printer drives do a lot of work for us, so the only thing we need to do is modify the template code and add pieces of our own, and configure the setup properly.

As stated earlier, Microsoft makes sure that the input is XPS, so you don't have to worry about various PDL's and applications that you have to support. Since there is nothing to configure for the start of the pipeline until our first filter is called, we are already done with that.

2.4 Implementation of the filters

This is only part that involves programming in the making of the entire driver. The technical details of implementing the pipeline filters and setting up the configuration files are described in Appendix B. In this section I will only mention the process of creating the filters.

2.4.1 Print Ticket Parser Filter

The first thing to implement according to the dataflow in the design is the print ticket parser filter. In this filter we extract the raw print ticket from the XPS input stream. I used notepad to generate my print job, which will result in creating 1 print ticket in the XPS file. It is possible that there are multiple print tickets in a XPS file and you should merge those print tickets with the main ticket if they exist. I used the pipeline PropertyBag to store the print ticket so I can retrieve it in the JDF filter.

2.4.2 JDF Insert Filter

At this point the print ticket is safely backed up in the pipeline PropertyBag, and the rest of the XPS content of the file is converted to PostScript and that stream is ready for us to read. As an intermediate solution, I have taken a JDF ticket from an Océ printer driver and used that as static string. In that string I'm injecting some values that I parse from the print ticket from the pipeline PropertyBag. When finished inserting all values into my ticket, I write it to the output stream, followed with the PostScript content from the input stream. That concludes the job for the filter and with that, the job for the pipeline.

2.4.3 Configuration files

To let the pipeline know that we want to use filters, we need to supply the pipeline configuration file. This is mandatory file, and without it, the printer driver will not even install.

Because I used the project that I created from the template, as will be described in the next chapter of this document, this configuration file should already contain the Monotype XPS – PS converter filter. Now it's our job to add our own filters at the right position and with the right input and output parameters.

Also be sure to update the INF file to comply with your pipeline configuration. Visual Studio will check these files and if there are inconsistencies, it will give build errors.

2.4.4 Implementation challenges

In the previous sections I described the pipeline and we presumed the pipeline filters are executed sequentially. However, the XPSDrv pipeline is different from other pipelines. It is the pseudo multi-threaded framework that enables single threaded filters to be executed in parallel with other filters. That

can pose a problem, because we expect the data to be ready (processed and set by the other pipeline filters), but on many occasions, that is not the case. The filters are instantiated and started in parallel, and that would mean that if we, for example, would try to access the print ticket in the PropertyBag, it would not be there yet because the first filter just started executing. The only remedy against this, is trying to read from the input stream as early as possible. The input stream read statement is blocking, and will halt further processing to the filter until the input stream is ready and has all the data available. This is managed by the pseudo multithread framework.

It took me quite a while to figure this out, because it is not mentioned in any of the documentation about the XPSDrv pipeline. Debugging and tracing my code gave strange results, and eventually Gé helped me out and said I had to place the read statement at the top of my filter. Apparently he also experienced the issue when experimenting with XPSDrv and V4.

I also had an issue with the implementation of the JDF Insert filter. I wanted to parse the XML from the PrintTicket with a decent XML parser, but did not succeed in getting a decent XML parser into C++. In a printer driver filter, you do not get the full .NET environment as you get in a normal application project, and it would also be a bit overkill to link the entire BOOST C++ library in the printer driver just for this. I noticed that I was wasting a lot of time with the issue and decided to set a timebox for fixing a decent solution and otherwise I would take the less decent route and just use text search and text selection functions to select certain values from the PrintTicket. Sadly I had to take the latter route, but it did result in a working solution within the time limits of the phase, and timeboxing helped me to maintain my project planning.

2.5 Testing the driver

To conclude the driver development of this phase, we tested on a test machine from Océ. At first there was strange output from my driver, but with a little help from TestGroup (the test team that is working on these machines) we quickly identified the problem. After a quick fix (setting a different mimetype in the JDF ticket), I was able to print to the Océ printer with a number of different printer settings, and with that I had satisfied almost every requirement for the driver.

The result of this phase still holds an issue though. When printing multiple copies, say 5 times, the MSxpsPS converter prints the document 5 times in PostScript. The printer however is also instructed to print the document 5 times. This means that I actually get the document 25 times. When I order the printer to staple every document, I get a staple every 5 documents. To fix this, we need to prevent the MSxpsPS to print multiple copies. This can be done by modifying the XPS Print Ticket before passing it to next filter. This way we backup the real number of copies in the PropertyBag, but fool the MSxpsPS filter to print only once.

2.6 Conclusions and recommendations

The task of building a v4 printer driver requires a lot of knowledge. There are so many mechanisms and subsystems that you need to take into account. The smallest error in a configuration file can cause the driver to not install or work properly. We also learnt that the new V4 printer driver model is quite a bit different from what it was before, and restricts the developer a lot more. On the other hand, it has become



a lot easier to develop a printer driver. An experienced developer should be able to do so in an hour or 2, according to Microsoft.

For me however, that was obviously not the case. The documentation, especially when I started with the project, was not complete and sometimes not even correct. This made development and understanding the mechanisms and architecture a lot harder. In the end it seems that the implementation of the filters is not that challenging, but the configuration of the driver is harder than it looks. There are a lot of values that are linked together by the system, and Visual Studio does check a lot of them, but is not able to validate them all.

The printer driver that I created does not quite satisfy all requirements that were posed. There are a couple of reasons for that. The biggest factor was time. I did not have enough time in this phase to complete all the features and fill in all requirements as I would have wanted. The printer driver as it is now just is capable of having basic printer output with a very limited subset of configurable options. This makes the printer driver compatible with virtually every Océ printer, but I have not even scratched the surface of the capabilities of the bigger Océ machines. The features that I did not implement are built-in constraint handling and Bidi communication. These features do not contribute to having basic output from the printer driver and therefore had a lower priority to being implemented into the driver. With having time issues, it is logical that these features would disappear first from the list things to implement. Maybe if I have time left at the end of the project, I will see if I can take a look at these features.

3 Enhancing with PrinterExtensions

For the next phase, the goal was to create the UI part of the printer driver. When I started this phase I did not realize that there were actually two environments to do this for. Halfway through the phase I realized that the UI part for the Metro environment is something entirely different. To split the 2 development tracks, I have divided the phase over two chapters. See Phase 3 in appendix A.

The goal for the next two chapters is to create a UI for a specific environment. This chapter describes the process of the UI development for the PrinterExtension in the desktop environment of Windows 8, and the next chapter does the same for the Metro environment. The UI is meant to be as complete as possible, and it needs to be able to set all available options in the printer driver.

3.1 Testing with the Microsoft samples

Before I start working on my own App, I wanted to test the sample from the Microsoft Developer pages, in particular the PrinterExtension sample from the website. I started to try to couple that to my printer driver. This seemed pretty easy, but it took me quite a while to figure this out due to missing, incomplete or wrong documentation. This is exactly what I was afraid of when I started the phase, and that is why I deliberately chose to start with the Microsoft samples first. If something already went wrong with those, I would have spent hours modifying code that in the end would not work anyway.

According to the documentation, the steps to couple the PrinterExtension to the printer driver would be the following:

1. Install the printer driver, if you have not done so already.
2. Get the printer driver ID, like described in Appendix C, Chapter 1
3. Insert the printer driver ID into the PrinterExtension application code (needed for enabling events)
4. Build (and optionally install) the PrinterExtension
5. Create and execute registry keys as described in Appendix C, Chapter 2 and 3
6. Run the PrinterExtension executable (either from Visual Studio or manually from explorer)
7. Open a desktop application that can print, like notepad and execute a print job to your installed printer job, click "Preferences" in the print dialog and if you did everything right, your printer extension should pop up.

If after completing the steps above, the PrinterExtension does not come up, there are a couple of things that could be wrong: Either the entered ID or location is wrong. The result is that the PrinterExtension crashes, or the framework crashes. In all these cases, the default Microsoft Preferences dialog will come up. To rule out the last option, you have to try again to start the Preferences. If after 2 or 3 times, I'd start looking in the Event log, to see what went wrong. Because we are working in a complete beta environment, the supporting frameworks are not completely stable yet, and it might not be our problem. You will still have to find out what is happening and maybe check if the problem is known with Microsoft.



I have encountered all scenarios that I described above, and at this moment the framework is the only unstable part. In the Consumer Preview, you are actually quite lucky if you can get the PrinterExtension to pop up twice in a row without showing the default Microsoft UI. In the Release Preview, this is solved.

3.2 Background information

The PrinterExtension is a substitute UI component for the advanced print dialog. When installed right, this UI comes up instead of the default Microsoft advanced printer preferences dialog. That means that this UI should have all functionality of the default dialog. The advantage of this UI is that it can be styled by the developer according to, a company policy, and add extra features to the printer driver.

When compiled, the PrinterExtension is an .exe, like a normal application on pretty much all current and previous versions of Windows. It can be accompanied by DLLs, and access system resources like normal applications do. They run on the Win32 and .NET 4 framework.

3.3 Modifying the Sample

Now that the coupling between the Microsoft PrinterExtension sample and the V4 printer driver is a success, I started modifying the code in the sample, so that it at least looks like I did a lot. I added some Océ logos that I took from the website, and styled the UI that it resembled the colours from the website. I also wanted to show more and different options than the sample did, so I also edited that.

After completing the modifications, I was surprised to see that it already was working, because I did not touch the control code of the print settings. I saw that the settings in the JDF ticket that came out of my printer driver resembled the values I chose in the PrinterExtension I just modified. After some more digging in the code I found out that all settings for the printer driver were available and loaded into the PrinterExtension, but not all were shown. I deliberately chose to only show the option that I support in the JDF ticket, because the other options would only affect the PostScript converter filter, and maybe have unexpected results in combination with other settings.

For showing printer feedback I functionality, I started implementing toner status information. In order to do that, I had to know how to access that data on an Océ printer. By simply asking my mentor, I found out that Océ uses the SMNP protocol to do this, and with the SNMP specification for the printer drivers they are working on, I could easily continue. I downloaded a tool to explore SNMP data from printers and other devices. To enable my driver to do the same, I downloaded a SNMP C# library which would handle the protocol for me. This saves me a lot of work, and if I had to implement it myself, I probably would have spent a lot of time testing and debugging on the connectivity part instead of the real feature.

The most difficult part on implementing the SNMP toner status was how to act with the different printers. Some of the Océ printers just have black toner, and some also have colour toner. I had to implement something that could switch between a text based UI for showing errors and availability of the information, and graphical bars with actual toner status.

A long time after modifying this sample, I discovered that the V4 model also has a built-in implementation of a SNMP client. The reason that I did not use it earlier is simply that I did not know of its existence.



Comparing the implementations now, I see that both implementations have their own advantages and disadvantages. Some functionality could have been implemented easier with the built-in client, but with the external client, I can retrieve information any time, no matter what entry point.

After some testing and debugging the PrinterExtension was ready. I checked if my PrinterExtension would also show in Metro apps. As I expected there was nothing to see in the Metro applications. Soon I found out that for Metro, I had to build an entirely different kind of PrinterExtension. The samples are both called PrinterExtension, but in the documentation, Microsoft describes the PrinterExtension to be desktop only, and for Metro they define the term Device Companion Apps.

3.4 Conclusions

The PrinterExtension itself is a very simple piece of software, which is easy to produce and to adapt. To communicate with the printer it uses just one API, and in the sample from Microsoft they have demonstrated clearly how to use it. The result is just a normal .exe that runs on Win32 with either the .NET 4.0 or 4.5 framework. The difference with a normal application is that a normal application is meant to be started by the user, and this PrinterExtension is launched by the Windows system, on behalf of the user, to enable him to configure advanced printing options. This is the only use of the application though, and it should not be used for another purpose.

Compared to the V3 printer driver model, this is a really big change. The V4 printer driver model replaces all printer connectivity possibilities with one API. The other difficulty is that the UI is not able to communicate with the printer driver at all, other than sending the configuration data through the API. The UI is shown up front, before even the spool file is generated, so during the setup of the print job, the UI does not know anything about what is going to be printed. The Windows Print Dialog does have a preview, but that information is not available through the API. Converting existing V3 UI's to the V4 PrinterExtension will be a challenge, and the PrinterExtension is the easy part. The Metro app will be even more challenging.

The SNMP client that I have used in the sample modification is actually a form of Bidi communication. This is not the real way to implement the communication in the printer driver, because the V4 printer driver model has standard proficiencies to make this implementation easier.

The requirements of the PrinterExtension have been met, and I have been able to implement even more than that. The SNMP client is an extra feature that I did not plan to implement in the beginning of the phase. Also the decorating of the UI was not part of the plan, but a little extra that I have been able to do because I saved a lot of time with using the sample as a base for my application.

Note: For detailed information see the document marked by **MS01** in the Sources section.

4 Metro style Apps

Desktop PrinterExtension done, it is time to get started with Metro. It did not take me long to find another sample project that was suitable for Metro. I followed the same approach as with the first PrinterExtension: First try to couple it to the printer driver, and then modify the code to adapt an Océ look and get the app up to the same functionality level as the desktop version.

4.1 Coupling the sample

The new sample project I found was available in 3 different programming languages: C++, C# (pure) and C# combined with JavaScript (JS). Since I had to make up for the time I had lost with the development of my V4 printer driver, I chose to use the C# sample, because I am most comfortable with that language.

Coupling the Metro App with the printer driver is the hardest part of the entire Metro PrinterExtension development. The documentation is not clear and if not read very carefully, it can be misinterpreted very easily. This was my downfall for the 2 times that I had to configure my test set up. Getting the Device metadata right is very tricky, and installing it holds some hidden surprises, like the location where to place them does not match with the documentation and some of the GUIDs that you have to specify, have to be made up by you. Since the documentation that was released with the Release Preview, the process of creating metadata has become a lot clearer. For the document, see source **MS03**. Be very careful when reading the documentation. The App ID has to come from the package manifest of the Metro App. The Package Family name is never needed in the Device metadata, though the document might suggest so.

Where the desktop PrinterExtension uses registry keys to couple to the driver, the Metro Device Apps need to be coupled by using Device metadata. These are special signed zip archives that contain a couple of XML files that describe a device. In one of these XML files, directives can be added to couple a Metro Device App to the device and its driver. This makes it easier to automate the installation of this kind of apps on a system, because Device metadata files are system independent. They just have to be placed into the correct folder. The registry keys for the desktop PrinterExtension contain the location of the .exe, and that can differ for every system, so that makes it a bit more difficult to deploy.

After finishing the device metadata and copying it to the local device metadata store on your machine, go to “Devices and Printers” in the Windows Control Panel and hit F5. This will trigger the Windows systems and service to process the Device metadata.

The fastest way to verify the coupling is pretty simple. First open the sample project in Visual Studio (which you probably have done already), and find the PrinterDriverID1 in the solution. Enter the printer driver ID from the printer you want to install (see Appendix C, Chapter 1 on how to get the printer driver ID). With that in place, build the solution and when that succeeds, deploy the solution. This will create an app tile in your Metro start menu. Open that app and click the “Get Associated Printers” button at the top of the screen. If your printer shows up in the dropdown box below, the coupling is a success. If not, you have done something wrong. I have encountered this situation a lot, and unfortunately I have not yet discovered a reason that it could be the system that does something wrong.

4.2 Background information

For a better understanding about the environment of the Metro Device apps, here is some global information. Detailed information can be found in the documents marked by **MS01** and **MS03** in the Sources section.

Metro apps run in the new WinRT (Windows Runtime) environment, which acts like a kind of virtual machine / sandbox. Metro apps have to provide capabilities in the appxmanifest file in order to get access to certain parts of the system. If for example an app requires internet access, the developer has to check the “Internet (client)” capability in the manifest. This capability will be shown to the user on the first run of the app or already when installing the app from the app store. This system is very similar to what e.g. Apple and Google have in their mobile operating systems iOS and Android.

Furthermore, the Metro apps run on the .NET 4.5 framework. When developing Metro apps, you do not have to pay attention to references, because by default everything is referenced already for you. Because the entire WinRT and .NET 4.5 environment is referenced, if you get errors on system parts that cannot be found, it can only mean that you cannot access it because of the sandbox.

4.3 Modifying the code

With the coupling succeeded, we can now start modifying the code to style it a bit to our wishes. Same as with the desktop PrinterExtension, I added some logos and I modified the splash screen to start with. I had to create a number of images to replace all the Microsoft logos that are defined in the appxmanifest file. I also added the control I built earlier for the desktop PrinterExtension to display the toner status. That is where I discovered a big difference between the 2 environments.

4.3.1 Getting SNMP Data

In order to get SNMP data, I need a printer port name, which serves as a DNS address for the printer. In the desktop environment I could simply access the registry and with a few lines of code I retrieved the printer port that was coupled to my printer driver. Because of the Metro sandbox environment, it is impossible to access the registry directly, so I started looking for a different approach to access the printer port in Metro. I found a solution that uses the same technique that retrieves the associated printers in the sample. I successfully retrieved the printer port name, and I could output it to the screen and see it while debugging, but I could not make a connection to the printer.

That is when I discovered that a complete namespace that I was using in the desktop PrinterExtension SNMP code was missing. That meant I had to find another way to get the IP address of my DNS name. A solution was found quickly, but it still did not work. After more searching I discovered that I had to modify the capabilities to get the Metro environment to grant my app internet and local network access.

If I had used the built-in SNMP functionality, I would have had no SNMP data for one, maybe 2 of the three entry points in the app. This is not the most elegant solution, but it is the solution that works in all entry points. It is worth considering both options when developing a commercial printer driver.



4.3.2 Devices & Print Slide-Out

Metro apps have 3 different entry points, and up until now I just tested with the main entry point. The main entry point is fired when the app is started from the Metro start menu, the second entry point that I can handle is the More Settings button on the Print Slide-Out. The third entry point is triggered by launching from notification, but I am not going to use that because I don't have notifications that I receive from my printer. Each entry point launches the app with different arguments, and that makes it possible for the developer to interact differently. Microsoft recommends designing the Print Slide-Out to use a different resolution than the full screen resolution you get from the main entry point.

However when I tested the sample app through this new entry point, it did not work. I first thought that I had made an error while coupling the app, but looking at the Windows Event viewer showed that my app was crashing, among other parts of the underlying system. Debugging the problem showed that the problem was more severe than initially suspected.

When trying to access a specific property from the entry point arguments, an exception is thrown by the framework, and when I tried to debug those errors in the Consumer Preview of Windows 8, Visual Studio and the complete Metro framework crashed. This was the confirmation for me that this error was beyond my reach and after a short conversation with my mentor we decided to contact Microsoft over this issue.

4.3.3 Support from Microsoft

The Support service engineer called me in the morning with bad news. Microsoft was not going to support the SDK (from which I took the sample) in the Consumer Preview of Windows 8. I would have to wait on the release of the Release Preview, test it again with that, and if the issue still is there, contact them with the same case number. Then they would reopen the case and see what they could do about it.

I downloaded the Release Preview as soon as it was released. Within an hour and a half I had my new Windows 8 installed but it took me at least 4 more hours to set up the rest of my development environment (especially Visual Studio took very long to install). It took me a day to install the printer driver and couple the printer extension and at first glance it looked promising, but when I started debugging the code with the correct entry point, I was able to reproduce the issue. That meant 2 things: It was not my Metro code that caused the crash, and the issue was not resolved in the Release Preview, so I could contact Nicolas again. I mailed the support engineer the bad news, with the differences between the exceptions that I was getting. He mailed me back that he was going to investigate the issue, since this was Release Preview, and Microsoft was now supporting the SDK samples as well.

The support engineer reported 2 days later that he could reproduce the issue and that he was going to track the error. Another 3 days later, he called me, reporting that he could not find any errors in the code, and that he needed a driver to investigate further. He asked if he could get the driver from me, but I knew that Océ would not approve that so easily. After discussing with Johan and Gé, as the Microsoft contact on Océ, we decided that I would send a stripped driver to Microsoft.

I started with stripping the filters from the driver, so it would be a set of configuration files only. When I started testing the driver, I also tested if the Metro PrinterExtension would still crash. Shockingly I

discovered that it would not crash any more. I made a backup of my current code, and started adding the filters back in, first with only one filter, and then with all three filters. I could not break the sample any more, and that meant that I had encountered a dreaded “ghost”-bug which was now vanished.

I reported back to the support engineer that while working on the driver, I could not reproduce the error any more, and I asked if he still wanted to pursue the issue and receive the driver, or that he would stop the chase and archive the case again. He wanted to stop the chase, because he already spent a considerable amount of time on it without any results. He archived the case again so if I found it again, I would just have to send him an e-mail and provide additional information.

The problem is solved, or gone at least. This is a rather unsatisfying solution, because we do not know what the problem really was. If it just was my own discovery without intervention from Microsoft, I would blame it on a faulty build. Because the support engineer confirmed that he could reproduce the issue, but the sample code was all right, means that it is related to the system, and the fact that it just disappeared makes me worry that it might come back.

4.3.4 Progress halted

Because the property we want to access in the print slide-out is vital for setting printer properties, I was blocked by the Microsoft issue. Together with my mentor we decided to continue with the next phase, the cloud research. When Microsoft would come with a solution or ask something about the, I would handle that first, before continuing on that phase. Since both PrinterExtensions communicate through the same API with the system, it should not be hard to implement real functionality in the Metro app, because I have a working sample with the desktop PrinterExtension. If the issue would still be there now, I would have to design around the issue in the phase where I create the cloud concept, but since the problem is solved, I will not take extra precautions.

4.4 Changes in Metro & Windows 8

If you sum it all up, it is quite shocking how many differences I have found with a simple sample. Even for companies with legacy code in XAML and C#, the restrictions in Metro and Windows RT could become quite serious issues. According to the Microsoft documentation, the V3 XPSDrv pipeline filters should be able to be reused without heavy modifications. They did somehow manage to enable the use of DLLs in Windows 8 that are built under previous version of Windows and .NET. In my experience Visual Studio just warns you about using non existing namespaces, but lets you continue to use it and the build does not fail on it.

4.5 Conclusions

The Metro environment needs some time to get accustomed to, but once you are in the flow, you quickly pick it up and adapt the changes Microsoft has made between the .NET 4 and .NET 4.5 frameworks. The Metro environment is highly limited by system access restrictions. For instance, as a developer I cannot access the Windows Registry, for internet or any other network access, I need to enable specific checkboxes in the Application Manifest file. These are no difficult steps, and they might sound familiar to Android or iOS developers, but for Windows developers this is new, and can cause confusion in the beginning. Once I had the correct rights for my app, I did not encounter any further errors and developing



for Metro becomes just as easy as for any other Windows platform. The new .NET 4.5 framework holds some new and very powerful new components and features and is certainly worth the effort.

In this sample I have discovered that my implementation of the SNMP client that serves as alternative to the printer driver implementation of Bidi communication holds some unexpected advantages. When I would have implemented Bidi communication in the driver, I would not have been able to get the Bidi information from the main entry point in the Metro app. When the Metro app is started from the Metro start menu, the app does not receive any information about the associated printer driver. Therefore it cannot access live printer data from the printer driver. This is a restriction in the V4 printer driver model; it is not possible to access Bidi information when the Windows print system is not active (that is when the user is actually printing something) to save power. Because my implementation uses a completely separated SNMP client, I am able to retrieve SNMP data when I want. The only challenge that I had with this implementation, is that I needed to retrieve the name of the printer port differently from the desktop PrinterExtension, because the WinRT environment will not let me access the Windows registry.

Also in this chapter, the requirements of the Metro app have been met, and the level of functionality is very much alike the PrinterExtension from the desktop environment. The SNMP client and the decorating of the app are also implemented here, but took a little bit more time to complete than with the desktop version. It took more time because of the challenges that came up with the WinRT environment.

5 Researching the Cloud

A major part of my graduation is research. I am going to investigate the possibilities of cloud integration into the printer driver and extension that we have already built. The research is documented separately, and I have added that document as Appendix D.

The goal of this phase is to expand the view that we have about the cloud, see what solutions are already there, see what we could use and what we would need to fully harvest the power of the cloud. The result of the research will be used to create a cloud concept that will combine as much of the strengths of the cloud as possible. The research will aim towards finding as much of these strengths as possible and finding compatibilities between them.

5.1 Collecting in-house sources

The first step in this research phase was to collect all the in-house sources available. To do that, I contacted a couple of people of who we knew they had done something with the cloud. I had meetings with Christian Luijten, Matthieu Helder and Rob Kersemakers, and a short phone conversation with Merijn Neeleman. One of the first things I noticed was that I was not the only graduate that was involved with the cloud. Qingwen Chen and Kevin Hoes also made prototypes of cloud concepts. Their work was out of my scope so I could not use their research as base for my research. Only the report by Christian Luijten was usable for me, because he collected a lot of data on various cloud services. The only thing that I had to do was to update the information, because the cloud is always changing, moving and growing. This is also the case with my own document, so check the sources for updates on my information.

In a meeting with Rob Kersemakers, he told us that they were busy trying to get access through the corporate firewall to cloud services. Because of the firewall in the Océ network, a lot of these cloud services are blocked and in order to deploy a product, we have to have access to at least one of these cloud services. He and his team had a meeting planned with the IT management department of Océ. That meeting still has to happen at this moment, but any result from that meeting will be communicated towards me. However, I could not wait for that to happen, because I simply do not have the time to wait 6 weeks on the result of a meeting, and than probably another 6 weeks or more for an actual change in the firewall or network, so we set that aside and took our own action. More on that will follow in the next phase. Rob also manages the Amazon server that Océ holds for test purposes, so we might contact him about that later on again.

Because I got access to a number of Océ documents, I have quite a bit of information at my disposal, but only a very limited amount can be used as research input. The reason why I first tracked down the sources that were available is because we do not want to do research twice. It is good that I have chosen to do so, because otherwise my document would otherwise have shown overlap with the document from Christian.

All the documents that I have mentioned above are internal Océ reports that I cannot provide as appendix because of confidentiality issues.

5.2 Research delineation

To delimit the research scope, we have to make some decisions. First we have to choose what side gets focus; client or server side. Because we already have a V4 printer driver and UI extensions, we have a pretty good idea of the environment that we are working in. That is why the focus of the research has to be on the server side of the solution. I will describe the integration environment in a chapter, but I will not elaborate on the client side technologies that we need to use, or features that we might need to implement.

The next decision is about the cloud services that we want to integrate. I chose to take in only the cloud services that “mattered”. That means that I only take in cloud services with a big user base. This makes it easier for our own customers to use the cloud functionality in the printer driver. Also a cloud service has to be relevant to our concept. We cannot use a service that has nothing to do with printers or printer drivers, like a weather service or music player service. The list with cloud services that we do want to investigate is based on the list of cloud services in Christian’s research. I will use his data where I can and update some facts if needed, and add some services that have emerged since his research.

Now that we delimited the width of our scope, we need to determine the depth too. Keeping in mind that we are broadening our view of the cloud, we don’t want to go into too much detail, but enough to understand the advantages and disadvantages of the technologies that we might want to use. When I don’t mention certain details, it does not mean that I did not investigate further. You might find additional details in my sources. I will describe all levels of the cloud service software, and with that, the protocols, architectures, cloud service types and implementation advices.

5.3 The real research

The first thing that you do when starting a research is asking yourself what you want to learn from the research. From this you can then construct a number of questions that you are going to answer during the entire research. For my research I have formulated a number of questions and supporting questions that I use as chapters and paragraphs in my report. I will now summarize my report here, but for the actual document, see Appendix D: “Cloud Integration Research”.

5.3.1 What research had been done?

In section 5.1 I already wrote about in-house research resources, but for the research I just used one: The report of Christian Luijten. In this chapter though, I don’t explicitly mention him, I just notify the reader that the information of Océ is embedded into my research and I will refer to external document when external sources are used.

5.3.2 What technologies can be used?

This section covers the basic transfer protocols that are used for cloud communication. HTTP and HTTPS are of course obvious technologies. XMPP on the other hand is not so obvious, but it is a protocol that has a broad usability because of its near-real-time properties and origin as instant messaging protocol. The next part of the question is more important: There are 3 cloud “flavours”, IaaS, PaaS and SaaS. These abbreviations will be used throughout the rest of the document.

From the collected data, I can conclude that IaaS and PaaS are useable because of their amount of freedom, and SaaS is not useable because of its lack of freedom. The PaaS services can still be limited in their freedom and therefore IaaS is the best option, because of the nearly unlimited freedom.

5.3.3 What existing cloud services can be used?

This chapter describes what existing services could be useful for integration into the printer driver. I first made a list of the interesting services from Christian's report, updated the information about the services in his report, and then expanded the list with additional services in the categories cloud file storage and cloud printing. With a complete list, I investigated the compatibilities between the services in the list. It would be unnecessary to integrate two cloud services which can already share resource, or it could have added value to integrate 2 service that combined can do more than when separated.

This part of research resulted in a list of services that might be worth integrating. There are a number of services that integrate with other cloud services, but except for Google's services, there are no big cloud services that are deeply integrated with each other.

5.3.4 What interaction should there be with the cloud service?

We take a look at some scenarios that could be realized with integrating cloud services into a printer driver. After that we investigate what technologies the cloud services commonly use, and take a closer look to the API in order to discover what information cloud services normally require. In this part we focus only on the top 5 biggest cloud services, because those are the only ones with a big enough user base to be actually eligible for using. After analyzing what the services need, how to get it from the user, and what existing workflows exist to get data from the user, we take another look at the scenarios. With the knowledge and understanding of the cloud services and their capabilities and needs, we can now analyze whether a scenario can be implemented with these services or not.

The result was that there are number of services that can be integrated when we support a combination of 2 technologies: OAuth and REST. Google Drive and Cloud Print, Dropbox and Microsoft SkyDrive all work on these technologies and that would result in a big common code base which would simplify everything. With supporting all these services, we would be able to achieve almost every scenario.

5.3.5 Where do we integrate the cloud?

In this chapter I investigate the client side environment of the integration. This is where the printer driver, printer extensions and metro applications are. I highlight the frameworks and environments and nothing more, because we have handled that in earlier chapters of this graduation report.

The Metro environment is a good environment to integrate into. Although the environment has strong restrictions, it just makes things a bit harder to implement, but not impossible. The .NET 4.5 environment is very powerful because of the new components and asynchronous call support mechanisms. It might be possible that we need a little help from the V4 printer driver, but the Metro app will be the main integration point.

5.3.6 What do competitors offer?

Reading about other products from other companies really strengthened my cloud concept, and the confidence that I was heading in the right direction. I have seen that some competitors already have products that are similar to my concept. Some other competitors are involved with the cloud on a completely different level, like selling cloud software development, and engaging into a cloud knowledge portal.

5.3.7 Can creating our own cloud service help in integrating more services?

One of my concepts contains the idea of creating our own cloud service. This service will have a REST API and will be able to access multiple other cloud services, like a relay / proxy service. This should help integrating the cloud services that we want more easily and securely than when integrating directly into the driver. In this chapter we investigate whether it is even possible to do this, and how it would be possible to realize the idea.

After analysing the normal integration method, and the cloud integration method and comparing them to what competitors have, I can conclude that having a cloud service really does help with the integration process. I discuss the different possibilities of where to host a cloud service and there it is clear that having just one real cloud service is preferred over having local servers in the firewall of the corporate networks, or even hybrid solutions.

5.3.8 How can we build our own cloud service?

This chapter kind of elaborates on the previous chapter. We take a look at the deployment options of a cloud service, and their advantages and disadvantages. We define a couple of criteria that are vital for our cloud service to have, and then match the cloud service hosting providers to those criteria, to see what service would really be suitable.

For the deployment option, the best choice for my proof of concept is to develop on local machine, and deploy on Amazon EC2. This is because Océ already has a server ready for use there and it will be easy to use that. It also allows maximal flexibility and extensibility, because we could just order a couple of extra VM's. Other options might have worked just as well, but I choose for the existing option, to be safe. I have also checked if I might have to make any compromises on certain areas for choosing Amazon, but for as far as I have investigated, I don't. Amazon comes with total freedom in using operating systems and underlying hardware, which means that we have total control over everything.

5.3.9 What infrastructure should be used?

Last but not least, we take a look at the infrastructure of our cloud concept. This is not about the cloud infrastructure like you would see in an IaaS solution, but about the corporate network at for example Océ, and how to grant the users / printers cloud access without introducing a major security leak. There are a couple of solutions, but we have to analyze what is best.

There are a few options that are possible here, and most of them contain complicated setups with servers managed in by the local IT departments and that would make it hard to implement. There is only one good option, and that is to just make one client and a cloud service. The cloud service would run on the



Amazon servers, so internet access is a prerequisite, but that should not be an issue. Because the service would be using HTTPS for data transfer, the connection would be secure, and the port would not be blocked by the IT departments (it is too common to block). The client app would have unhindered access to the cloud, and that is exactly what I aimed for.

5.4 Findings and conclusions

By taking all conclusions together I get a pretty clear picture of what technologies, methods and services that I want to combine to create my own cloud concept. The best solution would be to build my own cloud service and client. The service would run on Amazon EC2, on a Windows 2008 R2 virtual machine, and would function like a proxy to access multiple cloud service from only one data transfer line to the client. The concept ensures maximal security, and flexibility. For the proof of concept I will use Google Drive as demonstrator cloud service, but once that is implemented, other cloud services can be integrated also very easily, because of the common technologies.

We do have a solid base of technologies, service and methods, but the software design will play a critical role in ease of scaling up to multiple cloud services, and security of the concept. If I would mess up the implementation of OAuth by design, I would introduce a major security leak, so I have to avoid that.

6 Designing a Proof of Concept Cloud Solution

With the research I have gathered a lot of knowledge about the cloud, and I can now start to design a proof of concept. Is it not easy to come up with a concept that will work, and even great ideas have downsides and you can never be 100% sure that it is going to work. In this section I will guide you through the design decisions that I have made for my cloud concept solution. The concept is the deliverable for phase 5 from my planning in appendix A, and this chapter will discuss the complete process of the development of the concept, so the design and implementation.

6.1 The Concept

The idea for the proof of concept that I am going to implement is quite simple: Print to the cloud. The most of the time that our Metro style printer extension is used, is when the user want to print something. The only thing we have to do is to offer other storage possibilities and handle the transaction to them.

The first steps of the implementation will of course involve setting up the communication between the cloud service and the client. When that is done, I will attempt to implement the capability to send a file to the cloud, to have it stored there. If that succeeds, I will start integrating my implementation into my V4 printer driver.

If I have any time left, we might consider expanding the concept with letting the client (the printer driver) download a printed job, and send it to the printer where the driver is installed for. This will enable a primitive form of one of my original scenarios; printing to any printer in the network with just one printer driver. When I still have time left after finishing that feature, I might look into fully automating that feature.

6.2 Direct or indirect cloud access

In the research we have come across the idea to create our own cloud service as a intermediate service to access the cloud. I explained the differences, advantages and disadvantages of having either direct access to the cloud (direct communication with different cloud services like Dropbox and Google Drive) or indirect access, and with that implementing our own cloud service. We have seen that a number of competitors of Océ are already doing such things with success.

I came to the conclusion that having such service would improve the capabilities and flexibility of the entire solution so much, that I decided to create the service as part of the proof of concept. This decision has a great impact on the design of our concept, as predicted. From one point of view, we simplify things for IT management, and on the other side, we introduce the challenge to create our own API to access all other cloud APIs. Especially authentication and authorization is going to give us some additional challenges.

6.3 Choosing the service

The second decision that I had to make was what cloud service I was going to implement. There were a couple of things that needed to be considered. The first requirement was a reasonable customer base. That meant I had to pick one of the top five cloud services from my research document. A second

requirement was that the service has to be compatible. Almost every cloud service from the top five has a lot of clients and applications, but one of them had standard integration with a number of other cloud services, and provided easy steps for developing your own integration, and that was Google Drive.

There are a number of advantages involved in implementing Google Drive. It has a pure REST API, and uses OAuth for authentication and authorisation. This makes the implementation easy to convert to another service, because these technologies are big standards in the cloud. A second benefit is the close integration options with Google Cloud Print, which makes our driver integration of Google Drive more valuable. A third advantage is that Google provides code samples of Drive clients in various languages and platforms. There even are SDKs available with a number of libraries that we could use to simplify the implementation. This is very useful, especially for our proof of concept, but for a commercial product it might be better to implement the cloud access code without help from libraries, because other cloud services won't use the library and might not provide their own. Having common code between those services is highly recommended.

6.4 OCAPI

The service that we are going to create for the proof of concept is named OCAPI which stands for Océ Cloud API. OCAPI will run on the Océ cloud server, rented from Amazon, as part of the EC2 (Elastic Compute Cloud) service. This server is already available within Océ for other projects, and I have access to the server for test and development purposes of this project. However, for fast development and also to avoid network issues, I will develop the service on my local development machine, and deploy to the Amazon server later, if I have time left.

OCAPI will be the name of the protocol. The name of the server side of the concept is OCAPI Service. We also have a client side of the project, which will be named OCAPI Client. OCAPI Client will have common code with OCAPI Service. I will try to keep objects, object interfaces and serialisation code common, for maintainability and readability purposes.

6.5 The Design

With our idea in mind and the components to accomplish that idea ready, we need a design to put the ingredient together in the right order. I have come up with a high-level architecture design, which is the optimal solution for our proof of concept and can be expanded to other cloud services very easily. It combines as much of the strengths and advantages of the researched technologies as possible. Even the disadvantages and weaknesses are limited to a minimum, there will be some issues. This solution is the most acceptable compromise to that.

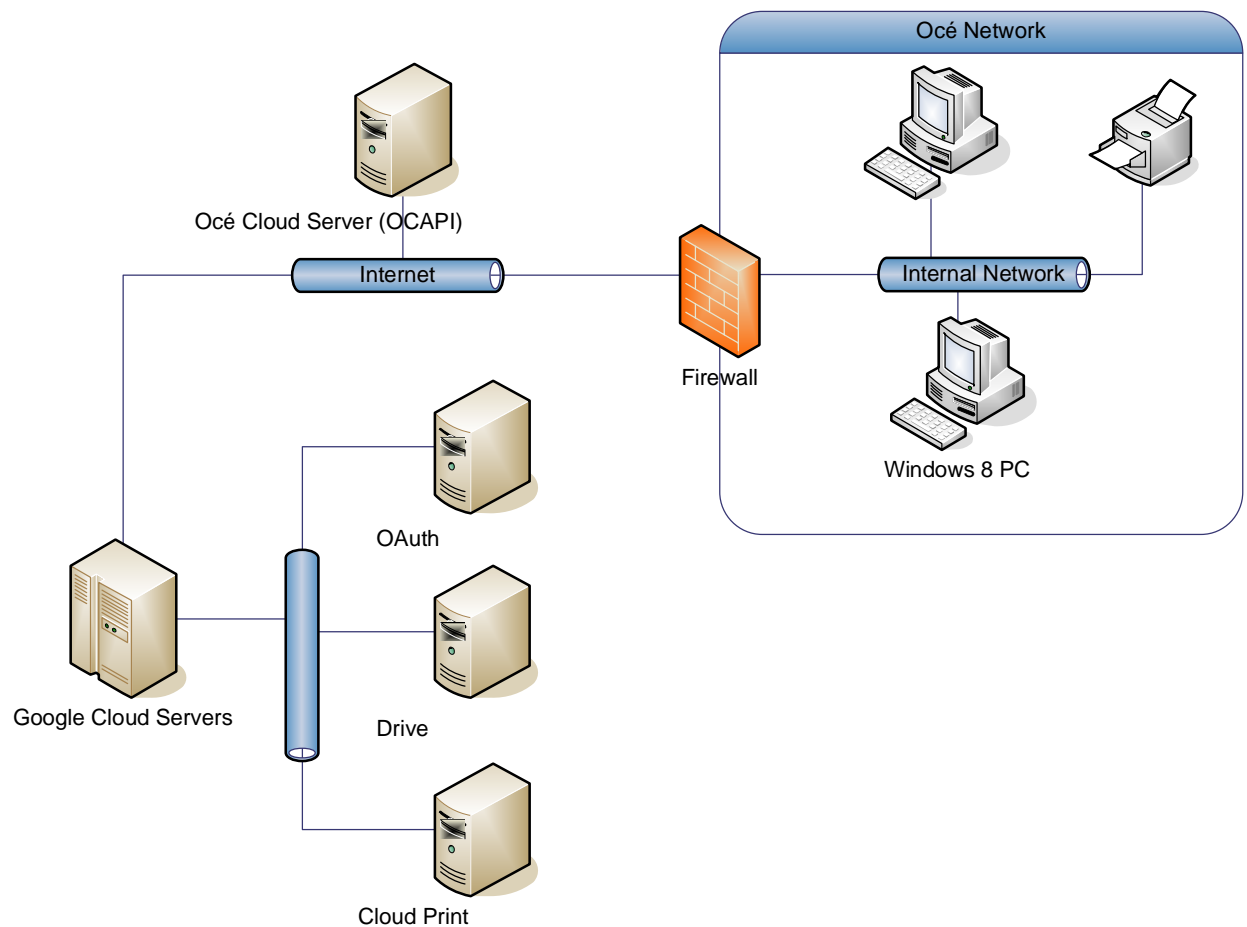


Figure 3 - Network Architecture of Design

The image above is a simplified depiction of the network architecture that I am going to apply. In the image, everything is connected to each other but because of firewalling in the Océ network, not everything is reachable. I shall explain how communication between the servers will run.

The first step from the client in the Océ network (the Windows 8 PC), is to request information about the available cloud services on the Océ cloud server outside the Océ Network. OCAPI will return a list of available services for the client to choose from. After receiving the data, the client can let the user choose what cloud service he wants to use.

The next step is does not involve using OCAPI itself. The client has to authenticate and authorize the use of OCAPI with Google Drive using OAuth. This is necessary because OCAPI is going to require access to personal files. By authorizing the access, the client gets a set of tokens. The client will send these tokens to OCAPI which will store these for later use.

The next step from the client is to request the capabilities of the selected cloud service. Capabilities are the definition of the cloud service API, and the client can call these capabilities on OCAPI. In the capabilities is described how to call them on OCAPI, with what parameters, and what type of data they

return. Once called on OCAPI, the cloud server will execute the request by sending data to, or retrieving data from the selected cloud service, using the authentication tokens that were sent before. The response from the cloud service can either be translated to OCAPI objects and passed on to the client, or just be passed on to the client and parsed there.

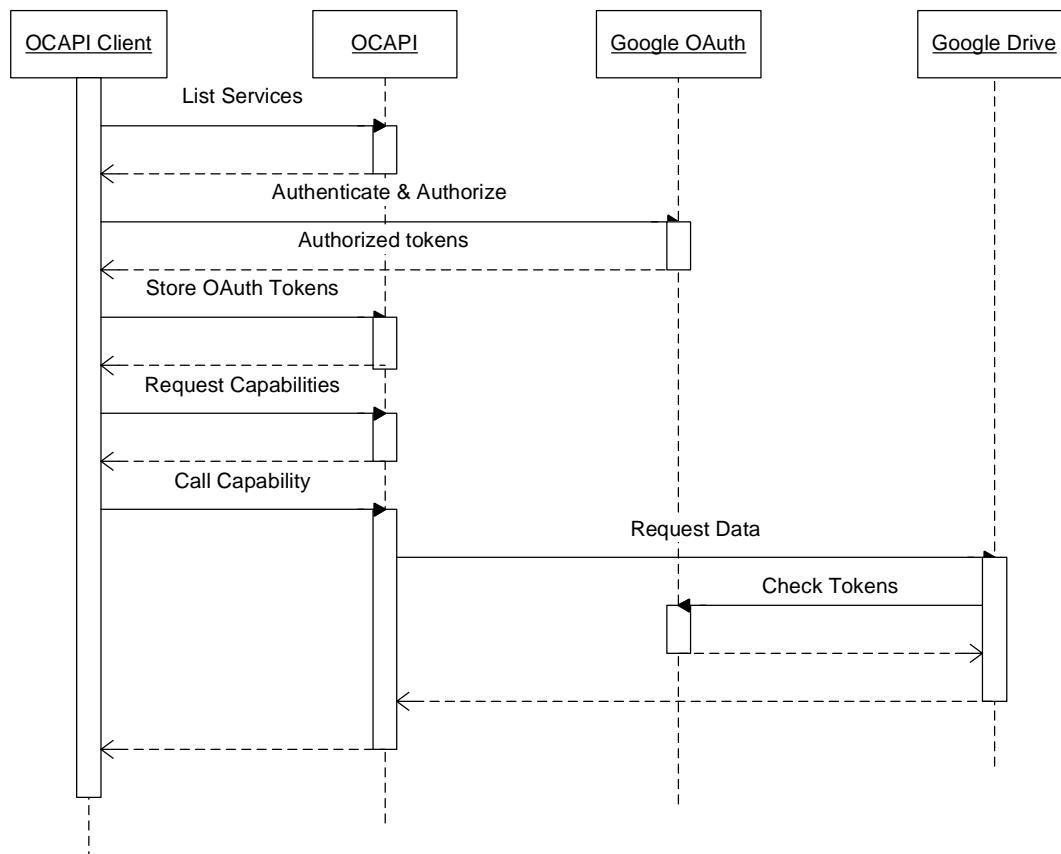


Figure 4 - Sequence Diagram of full data request

I have added a sequence diagram to clarify the steps that I describe above. In this diagram it becomes clearer that the client never gets to access the Google Drive APIs directly. The authentication and authorisation procedure is done in a internet browser, outside the Metro app. That means that I never have to handle the most sensitive information from the user: The username and password. The metro app itself never directly communicates with the third party cloud services.

6.6 The Implementation

The challenge in the implementation of the design is in creating OCAPI, our own cloud service with a custom API that enables us to communicate with all other cloud services. It has to be flexible enough to handle any kind of cloud protocol, even ones that are to be developed in the future, and any available API. For our proof of concept I will try to take this into account as much as possible, but I will undoubtedly run

into errors and design mistakes. It is up to Océ to learn from these mistakes as much as I do, before they start developing a commercial cloud solution.

6.6.1 OCAPI Service

The first part of the implementation was the building the structure of the cloud service, OCAPI, and getting communication up and running, so I could test. As environment of the service, I at first chose ASP .NET, with the .NET 4 framework, but when I started implementing the Metro client, I discovered some very interesting features in the new .NET 4.5 framework that would come in very handy in the service, so I switched. The reason why I chose ASP.NET is because it is a very powerful platform that would take a lot of basic implementation of my hands. I would not have to worry about HTTP handling. I did not yet realize the power of the built-in object serialisation, but I discovered that soon enough. In very little time I was sending complete objects to the client side, and the ASP framework was automatically serializing it for me.

With some basic service and capability data ready on the server, I needed to implement a solution for OAuth. This was the hardest thing to implement, because the Google servers reject anything that is not according to their specifications, and also here I encountered the problem of documentation not being up to date. Google Drive is pretty new, and during my development of OCAPI, they silently released V2 of their Drive API. Is also is quite confusing to be involved with multiple tokens, keys, and IDs, and requesting one token can lead to invalidating others. I did manage to design and implement a solution where the session tokens would be stored on the client side, and the app ID and secret would be stored / hardcoded in the service. The trick was to never let the app ID and secret be passed to the client because that would introduce a big security leak, but let the client provide the additional request data, and let the service do the request instead. Since that this solution was also the way that the request mechanism was going to be implemented, it also was a good practice, and the request mechanism took a little less time to be completed.

6.6.2 OCAPI Client

For the user side, we needed a client that would communicate with the OCAPI service. The target environment is a Metro style app, with the .NET 4.5 framework, because this is the same environment as one of the printer extension environments is, and this would make integrating with the printer driver easier. I was not aiming for a user friendly UI, just a test tool would suffice, so I built a UI with a couple of test buttons and some list boxes to retrieve and display data from the service. The buttons trigger a communication manager to retrieve data from the OCAPI service, and do authorisation through another OAuth manager. To achieve the communication with the OCAPI service, I use specially decorated classes. These classes are contained within 2 projects, of which one is a Metro style class library that uses references to link the sources of the OCAPI class library, which is a real class library (.DLL). These specially decorated classes are serialized to and deserialized from XML automatically by the ASP.NET framework. For the Metro app, I needed to develop a solution that would do the same, without putting any object specific code into the deserialisation. I succeeded in this by using Reflection, and the same deserializer being used by the ASP.NET framework.

For the authorisation, I needed to open a web URL, to let the user log into a website and approve the access that we want. The URL is provided by the OCAPI service. I was looking how I could open the browser from Metro apps, when I found a component that would be able to do this within Metro itself. The component is actually designed and built for doing this, and this made implementing a lot easier. The only thing that I would need to do extra is provide a start and end URL. The start URL would of course be the starting point of the user authentication sequence, and once the end URL would be reached, the component would automatically close, and I could collect the result, and continue the full OAuth sequence.

6.6.3 The combined result

The result of the combined service and client is quite impressive. It is fast and safe, it requires only one time authorisation, users can log out and log in with different accounts. It may not look friendly, but for a proof of concept, it is. The design is even more impressive. I managed to implement Google Drive as cloud service and access the user files, without laying any knowledge of Google Drive in the client. I am able to connect multiple other cloud services without modifying the client code, just by adding connection data to the service. This makes the result very easy to extend. The service does not need any storage, database, or long lasting connections like sessions, and completely complies with the REST architecture. This is a very good demonstrator for developers and designers to show off the capabilities and possibilities with the cloud. Integrating this solution with the printer driver will only enhance the concept even more.

6.7 Conclusions

The design of the concept is what makes the entire solution so powerful, and the implementation a success. The concept is designed so that the client does not have any knowledge on specific cloud services, other than OCAPI. The OCAPI protocol is able to deliver 3rd party cloud services and their capabilities to the user without the client having direct access to the 3rd party cloud service. Also the app never gets direct access to the user credentials, but just authorisation keys which can be revoked by the user. Other authorisation data like the client ID and client secret are hard-coded into the OCAPI service, so even by decrypting the HTTPS connection, the app credentials will not be revealed.

Implementing the cloud concept went pretty smooth, and I came across some pleasant surprises during the development of the code. The only difficulty that I encountered was the implementation of OAuth, because it involved some serious thinking and various considerations were being made towards using libraries to take care of the OAuth implementation. In the end I chose to implement it myself because of the complex situation. My cloud service could be seen as a web service and that would involve implementing another OAuth variant and authentication sequence than when treating the entire solution as an app. I chose to treat my solution like an app because that way, I could store the tokens on the client side and the app credentials, which is something different than user credentials, on the service side. This is the safest solution, and for difficulty it would hardly make any difference.

It was interesting to experience the limitations and also benefits of Metro. The restrictions on the environment and differences in the .NET 4.5 framework are bothering but manageable, but I would not be able to complete my concept the way I did now without using the new features of the framework. I would have been forced to use threading and that would have cost me a lot more time to manage.

7 Integrating the Cloud

This chapter describes the process of realising the deliverable for phase 6 from my planning in appendix A. The goal of the phase is to take the cloud concept from phase 5 and the printer driver and extensions from phase 2 and 3 and merge them together. This will result in a cloud capable printer driver.

The idea to make the printer driver access the cloud was pretty simple: Just integrate the cloud concept that I built into the Metro PrinterExtension. Since both applications run on the same environment (WinRT and .NET 4.5), it should not be hard to accomplish that. There just were some parts missing that I needed to fill in before the prototype would be complete.

7.1 Designing the integration

With the cloud access covered by the OCAPI client, I only needed to figure out a way to access the print output and upload that to the cloud with the configured connection. That is the first challenge of the integration. The V4 Printer Driver Model requires the UI to only run upfront, configure everything and when closed, the printer driver would actually start working. The UI instance is killed 500ms after closing the print slide-out, so that track stops there. I needed to find a solution to this problem, because if it is not possible to execute anything after the printer driver execution, I would not be able to upload anything, bringing down one of my most important use cases.

Luckily there are a couple of options to enable execution after the printer driver. The first option is to open a Metro background task that would check a certain folder for changes, and when a change occurs, act upon it. That would enable me to check the printer driver output folder, and once the print output would be done, upload the file with preconfigured options. Another option would be protocol activation of a program. Metro enables developers to deploy programs that register themselves to defined protocols. There are a couple more options out there, so I figured that I could accomplish the concept, and I would first start the real integration before worrying about the implementation route for the upload function.

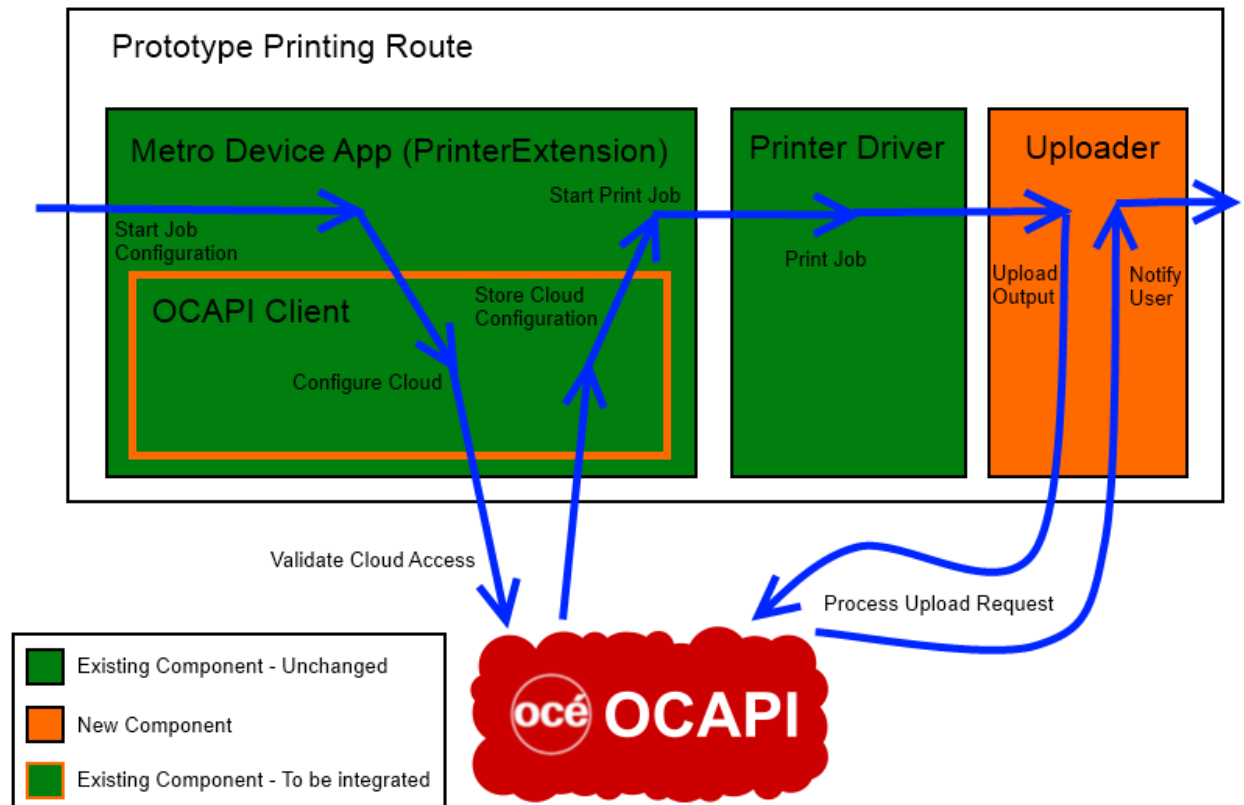


Figure 5 - Printing Route of the prototype in Windows 8

For clarifying the design, I have added a diagram that contains all software parts, new and existing and what function they have, and in what order they are executed. As you can see, most of the parts already exist, and just need to be combined and integrated. The uploader is new though, but should not be a problem to implement, since I would be able to use a lot of the common code from the OCAPI client.

7.2 Triggering upload

I have done a little research in order to understand the mechanisms of triggering other apps from a running Metro application (in my case, the Device App, or Metro PrinterExtension). I have come across a couple of solutions that I would like to explain, before continuing with the rest of the integration.

The first option that I have looked at is the Background task. This seems to be the Metro alternative to a Win32 process. Microsoft removed the possibility to run background processes from WinRT, probably for power consumption reasons during sleep. Instead we get background tasks, which are basically event triggered tasks. The startup of the background process subscribes to system events, and executes code when the event is fired. This allows the system to go into deep sleep without interruption from running processes. The only problem that I have left is when I need to start the background task, and would it be killed or cancelled when my Device app is being killed after doing the print job configuration. This is implementation detail and I decided to postpone researching further until I actually need to implement the feature.

Another option that took my attention was app activation through file or protocol association. This means that I would have a specific file extension or a protocol prefix linked to my app, and by opening such file or protocol, my uploader would be triggered with the correct location of the file. I might even be able to integrate that functionality in the existing device app, so I would not need to create a separate application for that. I have done some short test and I created a app that would trigger be triggered on my protocol, and that was successful. I have not figured out when I would launch protocol, but also that is implementation detail, and it might even be that I combine this option with the background task feature.

There still are other options worth investigating, but because of time-boxing, I have not yet done that. I think that I can accomplish a working solution with the features that I have seen, and if I have any time left, I will certainly look into the other options.

7.3 Integrating OCAPI Client into the PrinterExtension

For starting of with the real integration process, I first dusted off my existing device app, to see if everything was still working properly. There were no problems launching the app, and communication with the driver was also fine, so I started moving source into the app. Because of the structure of the app, I needed to make a new scenario for the OCAPI client, so I started with that before I actually added the OCAPI code. To build the structure was easy, because I could mimic a lot of existing code. It might even have been a little challenge to split the code needed to run the structure from the content code.

As I described earlier, the environment of the code in the device app is the same as the environment of the OCAPI client. I built the OCAPI client deliberately in this environment to make the integration process easier. The actual integration went as expected. It only involved copying code to the correct source folders, and including the sources into the project. I could copy and paste the UI XAML code into the target project and it ran with almost no modification. At UI level, the C# code was not as clean as I had wanted, so integration cost a bit more time there, but still no issue. The issues came when I started to test. The normal entry point was fine, and that was my first reference, but when I discovered the PrintTaskSettings entry point was failing, the integration process took a turn for the worse.

7.4 Problems Arise

The print slide-out or fly-out as Microsoft calls it, was failing. I had seen this behaviour before so it was not new for me and I had solved it before. This time it was different though. While debugging, I cannot find a reason why my app fails to start up properly. The splash screen shows, the initialisation code is running fine and when the splash screen closes, the app terminates with normal exit codes.

I started adding logging, setting breakpoint throughout the entire program, reversing code changes I had made, and enabled various debug options to discover any clues where my app would break. At one point in time I thought that I had fixed the error, because after making a fix the app worked fine, but after making some UI changes in a completely different software component (only accessible through the normal entry point), the app broke again, and I have not had it working since. This made me suspect that there might be something wrong with the platform instead of my code.



After having tried every debugging possibility that I knew, I started to investigate whether the system could be to blame. I created a new virtual machine, with a fresh installation of Windows 8 Release Preview, and all supporting software needed to do some tests. I figured out that only the fresh sample, downloaded from the Microsoft Sample website, would behave correctly and the rest of the apps, even the slightly modified version of the sample, would not. When I copied the working sample to my normal VM, the sample behaves just like the rest of the samples: broken.

While testing some more, I rebooted my original environment and now the app was working fine from both entry points. This means that both test results have become equally unreliable. The good news is that my code is working fine. Because I ran out of time, I was not able to investigate further.

7.5 Further development steps

Because I am out of time to complete this, I decided that I would first finish my report and other documentation that I would need to pass for my graduation, and spend the time left on trying to get a demo for the final graduation presentation.

When I have time left after finishing the obligations for the graduation, I would like to investigate the device app behaviour further, and see if rebooting on a more regular basis works. I will have to time box this strictly because I also want to spend a little time to implement the uploader. This way, I would be able to show the complete use case flow, with or without the Metro print entry point in my app.

My graduation period ends before the official release of the real Windows 8. This means that I cannot test the device app, or any of my software on the real OS. We have seen drastic changes between the Consumer Preview and the Release Preview, and I expect that the actual release will also introduce some changes. There is a chance that the device app will work fine on the release, because of a number of fixes in the OS. The Release Preview was said to be feature complete, but that does not mean that all features are working fine. I therefore strongly advise Océ to test the samples again on the real OS, and see what happens there.

7.6 Conclusions

Again, during this phase I encountered the challenges and risks of bleeding edge technology and products. For what I have seen now, I can only conclude that my primary development and testing environment is unstable. It could be anything from an update of some kind to persistence errors in the framework. With documentation not being there, or not being complete, it makes it even harder to figure out the errors. Also there are no other people reporting the problem online which makes me wonder if I am the only one using the sample, or the only one that is having problems with it. The integration works and the Metro PrinterExtension is cloud capable, just not on the level I would have wanted it to be.

I have done as much as I can to reduce the risk of having problems with integrating the software, but I could not have prevented the issues that I am having now. I will keep trying to at least implement a demo solution. Because the normal entry points of the Metro apps are working fine, that should not be an issue. I could also try to implement the client into the desktop PrinterExtension, and leave Metro for the time being. There are enough solution routes to accomplish my goal; it only needs a bit more time.



Sources & Installable Binaries

Throughout the project I have made a considerable amount of software. For almost every phase of the project, there also is a software deliverable. I have added a index of the software deliverables and where the software can be found.

First V4 Driver (Consumer Preview Version)

This driver is built for x64 Windows 8 systems, and test signed by my Visual Studio 11. If you want to install the driver, and for some reason the test certificate is rejected by your Windows 8, you will have to temporarily disable the certification check by following the guide marked by **DDE01**. You can also build your own driver using the sources (probably less work if you already installed the development environment).

The archives are to be found at the following location:

{CD}\Driver – Phase 2

- V4PSJDFTest_1.0_Stable.zip (Source)
- V4PSJDFTest_Package_1.0_Stable.zip (Installation files)

Desktop PrinterExtension & Metro Device App

These applications are extensions to the V4 printer driver shown above. They will not function (properly) without it being installed on your system, so do that first if you have not done so.

Installing the Desktop PrinterExtension requires some extra attention, and involves making changes to your registry. Backing up your system is advised because doing something wrong here might result in a corrupt system. I have described the steps to register the PrinterExtension in Appendix C. You have to place the binary executable of the PrinterExtension at a static location (or leave it on the debug location) on your harddrive, and fill in the location of the executable into the registry key file.

The Metro Device app also requires some extra attention. I have described the steps to install the Metro Device App in appendix E. The device metadata is included in the zip.

Because these projects are built using the Consumer Preview version of Windows 8, it is likely that they will not run properly on next versions of the OS. Please clean and build the projects before using the binaries. This is also the reason why the binaries are not included. The source can be found on the following location:

{CD}\Extensions – Phase 3

- Metro_Device_App.zip
- Desktop_Printer_Extension.zip

Cloud Concept

This software is built after the research phase and the release of Windows 8 Release Preview. Because of issues with the Consumer Preview version, this software is developed on the Release Preview and therefore it will probably not run on the Consumer Preview (I admit that I have not tested it). In order to run this software, you will have to upgrade to the Release Preview version of Windows 8 and Visual Studio 2012. If you are already running that or newer versions, you do not have to take any action.

The software for the cloud concept consists of 2 parts: An ASP.NET web / cloud service (OCAPI Service) and a Metro client app (OCAPI Client). In order to test the software, you will have to first deploy the OCAPI Service, and then deploy the OCAPI Client (follow the second step in appendix E). When running the client, do not forget to enter the correct URL of the service. When you are running the service on your local development machine, the URL might be something like <http://localhost:51169/>. The URL will be displayed in a balloon near the system tray for a brief moment when you deploy the OCAPI Service.

There are no binaries to be delivered here, because the software has to be deployed to the system by Visual Studio. The source project can be found on the following location:

{CD}\OCAPI – Phase 5

- OCAPI_Service.zip
- OCAPI_Client.zip

Cloud Integration

This software is the result of the integration of the cloud concept that was built in phase 5, into the Metro Device App that was built in phase 3. Some additional tweaks and modifications were necessary to get the solution working, but because of errors during the integration process that resulted into time pressure, the level of the solution is not what I wanted. Nevertheless, I am still proud on what I have achieved with this, because I have made the first cloud capable Windows 8, V4 printer driver extension.

Again the software solution consists of 2 parts. The OCAPI service is the same from the Cloud Concept deliverable that you find in the previous section. For further instructions on the installation of the service, see section 10.3. The other software part is the Metro Device app. For instructions on how to install / deploy this project, see section 10.2. The project can be found on the following location:

{CD}\Integration – Phase 6

- OcéAppV2.zip

Evaluation

During this project I have learnt a lot from a broad spectrum of subjects. At the start of the project I was confronted with a large amount of documentation that I would need to understand before I could start with printer driver development. I needed knowledge about the new V4 printer driver model, as well as a bit of knowledge about the legacy V3, to understand the differences between the two models. In the mean time I had to make a planning for the rest of the project and think of all the steps that I would have to take to bring the project to a success. I made a planning with phases, and every phase would overlap with the previous phase. This planning has saved me because without the overlap, I would not have delivered as many products as I can now. It allowed me to finish the products, or build a quick alternative if I were to get stuck. Then the development of the printer driver started, and that was for me the real start of the project. I knew that I did not fully understand the documentation yet, but as soon as I got to work with the samples from Microsoft, I started filling the gaps in my knowledge. This is also where I proved that my planning was working as I wanted, because I needed extra time to get my driver working. I was struggling with the XML parsing and I eventually had to set a timebox for it. After exceeding that timebox, I implemented a quick alternative. With the printer driver now working correctly, I moved on to building the PrinterExtensions. This is where I entered more comfortable terrain for me, because I still am more comfortable with C# than with the C++ of the printer driver. The first extension for the desktop environment just took me a while to connect to the printer driver. This was caused by the documentation still being in a preliminary version, being unclear and incomplete. This was also the stage that I found out that the extension that I was building was not going to work for the Metro environment, and that I had to build a completely different thing for that. Luckily I had time left in my phase to switch to the Metro apps. It took me a few days to realise how much different these environments actually were, but that did not hinder me so much. The real challenges were the coupling of the app to the printer driver, just like with the desktop version, and the errors that I was getting from one of the app entry points. I reported the errors to Microsoft and while waiting for their response, I continued with the app, and later with the next phase: The research of the cloud. The research in the beginning was quite fun, because I had to meet with a number of people from Océ that had already worked with the cloud. They all showed great enthusiasm and were glad that they could help, and interested in my result when I would finish. As the research continued it almost became boring to do because I was working 6 weeks on a document. In the end the research paid off because it was a great base for my cloud concept design. I did really enjoy building the design and implementing it because this is where I could apply everything about software engineering that I have learned in the past 4 years of my study. Despite that I was working on two different platforms that I hardly knew, I did not run into problems while implementing the design, except for some small issues that I would have had anyway, no matter what environment. The next phase of the project was to integrate the cloud concept into the printer driver. In the beginning of the phase everything went well and without errors, because I made the correct choices during the design of the concept. Soon I ran into problems though, because I could not launch my app from the Metro print dialog any more. It took me a week to discover that it was to blame on the operating system and a reboot would solve the problem. Because of this time loss, I had to timebox the implementation and therefor I do not have the integration on the level of functionality that I would like, but it does work. The remaining time for my graduation I mainly spent on updating documentation and getting things ready for my defence. I accomplished all the goals that I wanted to accomplish and proved that I am ready for the real world of software development.

Sources

- Reference: MS01
Name: Developing V4 Print Driver (developing-v4-print-drivers.docx)
Source: <http://msdn.microsoft.com/en-us/library/windows/hardware/br259124.aspx>
Writer: Microsoft
Reliability: Good
- Reference: MS02
Name: XPSDrv Filter Pipeline (XPSDrv_FilterPipe.docx)
Source: <http://msdn.microsoft.com/en-us/library/windows/hardware/gg463364.aspx>
Writer: Microsoft
Reliability: Good
- Reference: MS03
Name: Developing Metro style Device Apps for Printers
Source: <http://msdn.microsoft.com/en-us/library/windows/hardware/br259129>
Writer: Microsoft
Reliability: Good
- Reference: DDE01
Name: Disable Driver Signature Enforcement in Windows 8 Consumer Preview
Source: <http://laslow.net/2012/03/14/disable-driver-signature-enforcement-in-windows-8-cp/>
Writer: Laslow.net
Reliability: Good (Confirmed working solution)



Canon
CANON GROUP

Appendices

- Appendix A – Project Initiation Document (PID)
- Appendix B – Creating A V4 Printer Driver
- Appendix C – Registry Keys For Desktop Printer Extension
- Appendix D – Cloud Integration Research
- Appendix E – Installing Metro Device App



Canon
CANON GROUP

PID Cloudprinting & Océ

Internship Maurice Wingbermhle

author(s)
Maurice Wingbermhle



Table of content

Document History	3
Foreword	4
1 Background	5
1.1 Context & Motivation	5
1.2 The Current Situation	5
2 Project Definition	6
2.1 Project Goals	6
2.2 Chosen Approach	6
2.3 Scope of the project	6
2.4 Products and Deliverables / Result	7
2.5 Limitations	7
2.6 Dependencies	7
2.7 Preconditions	7
2.8 Assumptions	8
3 Project Organisation Structure	9
3.1 Contractor	9
3.2 Executor	9
3.3 Internship Guidance / Judgement	9
3.4 User Group	9
4 Project Control	10
4.1 Reporting	10
4.2 Progress control	10
4.3 Tolerances	11
4.4 Risk Management	11
4.5 Issue Management	11
4.6 Deviation procedure	12
5 Appendices	13



Document History

Revisions

Version	Status	Date	Changes
0.1	concept	23-03-2012	Set-up document
0.2	concept	29-03-2012	Changes after review from Johan
0.3	concept	2-04-2012	Minor changes to PID, clarification of planning
1.0	Release	5-4-2012	Changes in Planning document

Approval

This document needs the following approvals:

Version	Date of approval	Name	Function	Check
1.0		Jos Verhagen	School Mentor	

Distribution

This document has been sent to:

Version	Date Sent	Name	Function
1.0	5-4-2012	Jos Verhagen	School Mentor
1.0	5-4-2012	Johan Hoogendoorn	Company Mentor



Canon
CANON GROUP

Foreword

This document is the Project Initiation Document for the internship project of Maurice Wingbermhühle at Océ Technologies B.V.. The purpose of this document is to define the project, serve as a base for the management of the project and to enable the determination of the success of the project.

During the project he will conduct research to Windows 8, Windows Driver Kit 8 and the new Printer Driver Model v4, and the possibilities of integrating cloud printing within a new Windows 8 driver.

The main goal of this project is the internship and there no compromises will be made to this target. However, due to the new technologies and possible interesting new features for Océ Technologies, all findings, documentation and other products will be shared with Océ Technologies B.V. and adaptations might be made to the original product to match with their quality standards.

This Project Initiation Document, or PID for short, is a planning document and shall include the following fundamental aspects of the project:

- What are the goals of the project?
- Why is it important to achieve these goals?
- Who are involved in this project, and what are their roles and responsibilities?
- How and when will consequences, that are described in this PID, be realized?

This document will be used...

- ... to be sure that the project has a healthy base.
- ... to serve as a document that provides guidelines and targets to check the progress and success of the project.

This document will contains the following components:

- Background information
- Project definition
- Project organisation structure
- Project control

1 Background

1.1 Context & Motivation

With the introduction of Windows 8, Microsoft introduced printer driver model v4, allowing use of built-in constraint handling, Bidi communication, Metro style user interfaces, etc.

We're interested in finding out exactly how we can benefit the most from the new architecture. In particular we're interested in the Metro style; how this works from an end-user perspective, and if (and how) this opens up new paths for cloud printing services.

Windows 8 will be suitable for tablets. These mobile devices open up new ways of using our printer systems. We're interested in finding out how this can be used best. In this context, one of the questions that we have is how we can benefit the most of cloud printing, and that question can be divided into multiple questions:

- What should this interaction to cloud services be?
- What do competitors (Apple, Android) offer?
- Which customer workflow exist?
- Can we support these?

The motivation for the start of the project is the internship of Maurice Wingbermhühle at Océ Technologies B.V.. The project is designed and adapted for Maurice Wingbermhühle, so it will fit his profile.

The assignment description and a test case are added as appendix to this document. The contents of the assignment description are also integrated in this document.

1.2 The Current Situation

At this moment the driver team has everything implemented in the V3 print driver model, and are still continuing on active development of v3 drivers and supporting current drivers. The team is also working on a family driver for all the devices. This means that there will be 1 driver for all the devices. Currently there are separate drivers for all devices.

Since Windows 8 will be suitable for tablets, Microsoft made changes to the print driver architecture to improve user experience, increase system stability and potentially safe battery life. In order to achieve that they created a couple of new API's and removed or limited some other functions. Because of this, it is likely that current V3 drivers won't work anymore on Windows 8, so new drivers have to be made to fill the void.

2 Project Definition

2.1 Project Goals

The goals of the project are:

- Make a proof of concept printer driver for Microsoft's print driver model v4 (loosely based on one of Océ's printer systems) with a Metro Style UI, built in constraint handling, Bidi communication for printer status information.
- Using Windows 8 and the v4 Print Driver Model, create a metro-style printer driver with a limited subset of job settings.
- Investigate how the new V4 model, and with that the created print driver, can be used for cloud printing purposes.
- Based on the investigation on the printer driver / cloud interaction, extend the driver / job submitter application to use cloud service functionality the best way possible.

Based on those main goals, we will go through a couple of phases, each phase with its own products and deliverables.

2.2 Chosen Approach

The project will be divided into phases. Every phase represents a major step towards the end goals. Every week we will look back at the week and see what we have achieved and what was planned. From that we will determine if the rest of the planning is still feasible. Phases might be dependent on each other, so one phase might be research, and the phase afterwards might build a product out of the findings of the earlier phase(s). Phases might also go parallel or overlap, for example while waiting for feedback or response of some kind. Phases have goals, actions and deliverables.

Later in this document we will describe more details of the planning.

2.3 Scope of the project

The project will be focussed on Windows 8 and V4 Print Driver Model printer drivers. We might include research to other operating systems, to see what other systems already have, and how we could enhance our system. The project will not only be research, but we will be making actual drivers and possibly additional software to support the driver.

2.4 Products and Deliverables / Result

This project will render multiple products and deliverables:

- Research report on Cloud printing integration with the V4 print driver model
- Proof of concept V4 and Metro-style print driver with cloud service integration
- Internship Report

Additionally, more documentation on the software or other concepts might be made, but these won't be deliverables for school, unless specially requested. These documents are as a support for writing the internship report and the Océ driver team, for further research.

2.5 Limitations

Since Windows 8, Visual Studio 11, Windows Driver Kit 8 and Print Driver Model V4 are all not final, and still under active development, it might be possible that we encounter a defect or unimplemented feature that obstructs our research or development. We might not be able to find a workaround, so this way our work might be limited. Some of the products above are beta, others are stripped releases.

Océ Technologies B.V. has access to some solutions for these problems. We can contact Microsoft with our findings and problems and they will check if they have a solution for that which we could use. Océ is also part of the Tier 2 feedback program for Windows 8, so Microsoft would appreciate if we contact them with our findings / problems. In the case of actual blocking problems, we will resort to these resources. This might mess up our planning, and our product might change.

2.6 Dependencies

Due to the experimental nature of the project, with all new technologies and experimental / beta software, we might encounter defects. Our product might depend on a feature that Microsoft still needs to release to public, before we can finish our product. Up to now, no reasons have been found to assume we cannot finish our products, but research during the project might prove otherwise. Also in this scenario we will contact Microsoft, and they might provide us with a preliminary solution. Worst case scenario is that there is no solution yet, and we have to develop it ourselves or fall back to older existing technologies.

On the subject of cloud services, there is already knowledge within Océ, and we will search for this knowledge and go talk to some people of Océ when necessary.

2.7 Preconditions

All software needed for this project has already been released and are available for download. We will not be needing any other preconditions to start this project.



Canon
CANON GROUP

2.8 Assumptions

For documentation of the project and its products, the assumption will be made that the reader has some knowledge of the Windows driver development. For the sake of the readability of the document and relevance to the product, we cannot explain every term and aspect that can be considered known for a driver developer throughout the entire document. Therefore we will create a terminology and abbreviations list. We expect the reader to have a ICT background and have at least basic understanding of programming, and the internals of a operating system, such as drivers and devices.

3 Project Organisation Structure

3.1 Contractor

Océ Technologies B.V.
Research & Development

Contact:

Johan Hoogendoorn (Company mentor)

Software Engineer

Software Development 1

St. Urbanusweg 43

5914CA Venlo

The Netherlands

+31 (0)77 359 4153

johan.hoogendoorn@oce.com

3.2 Executor

Maurice Wingbermühle (Intern)

Software Development 1

St. Urbanusweg 43

5914CA Venlo

The Netherlands

+31 (0)77 359 2773

maurice.wingbermuhle@oce.com

3.3 Internship Guidance / Judgement

Jos Verhagen (School Mentor)

ICT & Technology

Fontys Hogeschool ICT Eindhoven

Rachelsmolen 1 Gebouw R1 Kamer 4.55

5612MA Eindhoven

The Netherlands

Telefoon: 08778 71189 / 0653175749

j.verhagen@fontys.nl

3.4 User Group

Software Development 1 – Driver Development Team

Océ Technologies B.V. – Department R&D

4 Project Control

4.1 Reporting

During the internship, there will be several reports from the intern to the mentors. In this section we will describe how and when this will occur.

Once every 2 weeks, the intern will report to the school mentor, by sending a short e-mail about the progress and process of the project. This will give the school mentor insight in how the project is running and check that with how the project should be running. The school mentor could interfere if he thinks the project is not running well enough, and he will provide feedback to the intern when needed.

When the school mentor will be visiting the company at a minimum of two times. The first time after 2 or 3 weeks from the start of the internship for general internship information and the second time short before the final session, at which time the presentation will be rehearsed.

If (process) problems occur during the internship, the school mentor can be contacted. He can assist in finding a suitable solution to the problem so the internship can be completed successfully.

Feedback from the school mentor will be oral, via e-mail or by visiting the company, depending on the severity of the feedback.

On a more regular basis there will be reporting to the company mentor. There will be a weekly meeting where the status of the project will be addressed and feedback or input will be exchanged. Apart from that, when the intern has questions he can pose them directly via e-mail or oral communication.

Every morning, the intern will participate in the daily stand-up, to have a moment of reflection, to discuss blocking issues, update the rest of the team with his findings of the day before, and activities of that day. Since the company mentor will also be present there, he also will be updated of that. During this stand-up, the intern will be updated of the status of the projects of the other team members, and with that, there might be relevant information for the intern as well.

Documents and report will be made by throughout the internship. These documents have to be approved by the company mentor and legal department before going outside the company, because of the possible sensitive information in the documents. The document can be changed after feedback of the company mentor or legal department. He will assist in determining whether the enclosed information is needed to assess the document and project.

4.2 Progress control

Because of the regular reporting to both mentors, there are more people involved that can assess the situation of the project. If either of the mentors feel the project is not progressing as it should, they will provide feedback to the intern, on which he will take action to improve that. Because of the short intervals, the damage will not be too severe, and therefore can still be corrected.

4.3 Tolerances

We will be tolerating some delay or advancements in the project. Since we have no monetary budget, we can only manage the time aspect of the project.

For the project in total, 1 week delay will be tolerated. A week extra delay of the project would result in a miss of the final session. Since one week is permitted, sickness or taking a day off should not result in problems.

Since we will be working in phases, we can introduce tolerances in them as well to create buffers to reduce the risk of overall delay. Depending on the size and the risks of the phase, we will assess whether a buffer is needed, and how big the buffer needs to be. A preliminary planning will be added to this document in a later section.

4.4 Risk Management

For now, we only now that time management might be a risk, but we have caught that with regular feedback sessions with either mentors. One of the phases, the research phase, might render additional risks, but have to be handled / resolved after that phase.

At this moment we only have the risk that there are unimplemented features in the current release of Windows Driver Kit, Windows 8 or Visual Studio 11, or that there are bugs in either of these software packages. That is the risk of working with beta / bleeding edge software. The likelihood we find such issue is pretty high since all the software we're working with is still under active development. The occasion that such defect will block our research or development is small, so the risk is relatively low. The result of the risk is that we will lose time searching for a solution for the resulting problems of the defect. That is the reason to build in a buffer in the planning to catch these problems. Another tool that will help is time boxing. When a deadline for a task is not met, we should assess whether to continue on the task and shrink another, or drop it and work around the issue. We will assess how much time we need when a defect occurs, and determine what to do then. We cannot predict these defects and their time consumption in our planning.

4.5 Issue Management

Any issues within the project (difficulties with the company or any other problems) should be reported to the school mentor immediately. The issues have to be handled immediately because postponing the issue can be devastating to the project. The issue is best to be resolved between involved parties directly, but when necessary, the school mentor shall intervene.



Canon
CANON GROUP

4.6 Deviation procedure

When drastic deviations from the planning have to be made, a new planning shall be made, and sent to the mentors for approval. Severe deviations from any other project related subject has to be handled accordingly. For example, when the intern is going to deviate from the original project path (like implementing a totally different product than described in this document), this document has to be revised and reapproved.



Canon
CANON GROUP

5 Appendices

Appendix A: Planning

Appendix B: Assignment Description & Test case



Canon
CANON GROUP

Planning

Cloudprinting & Océ

author(s)
Maurice Wingbermühle

1 Phases

Phase	Description	Duration
1	Getting started, reading on the subjects, writing PID	3 weeks
2	Developing basic print driver	4 weeks
3	Enhance print driver with Metro-style UI & Submitter app	6 weeks
4	Research cloud service integration	6 weeks
5	Develop cloud service integration concept	4 weeks
6	Implement concept into existing driver	4 weeks
7	Test solution on a population of users	2 weeks
8	Writing Internship Report	12 weeks
9	Finalizing Internship	4 weeks
10	Enhance / Expand print driver with additional features	4 weeks

We will clarify the phasing, describing the goals, activities and deliverables of each individual phase.

1.1 Phase 1 – Getting Started

Goal: Getting started, retrieving context about the subjects. Throughout the project the context will remain to grow, but there is a certain amount of context needed to get started with printer drivers. The intern has to learn a lot about printer drivers in general and in particular about the V4 print driver model.

Activities: Reading documentation and examining examples are activities that will be done during this phase. Furthermore, the PID and planning (this document) will be made.

Deliverables: The PID and planning. As a special remark, the school mentor will visit the company in this period.

Definition of Done: The PID and planning are ready and approved by the school mentor.

1.2 Phase 2 – Developing a basic printer driver

Goal: Build a basic V4 printer driver suitable for one of Océ's devices, with limited functionality.

Activities: Writing the driver, and documenting the steps made by the intern to accomplish his goal, and difficulties that occurred in order to get to them.

Deliverables: A working V4 driver, for one of Océ's devices, Development Environment Installation Guide, Design Document

Definition of Done: A print job is successfully executed to a Océ printer through the built V4 driver. All Deliverables are ready & approved.

1.3 Phase 3 – Enhance the printer driver with Metro-style UI & Submitter App

Goal: Build a metro style UI for the basic print driver.

Activities: Before we can build the UI and submitter, we will investigate if we can use Océ's new Ocean look & feel to implement our UI. We might contact some people outside of the driver team in order to retrieve existing knowledge about this subject, and see if we can adapt the existing Ocean UI to our needs or we need to invent something of our own. We will first have to research and choose what technologies we want to use, like C#, Javascript, C++, or a combination of them, running on WinRT or Win32.

Deliverables: The Metro style UI for the basic print driver, and a description document of the choices made during the process of developing the UI and submitter

Definition of Done: The UI can be used to issue (submit) a print job, retrieve & change settings with Bidi and do constraint handling. All deliverables are ready and approved.

1.4 Phase 4 – Research cloud service integration

Goal: Research how cloud services can be integrated into our driver.

Activities: Research how cloud services could enhance our print experience. We will first have to investigate what customer workflows exist and could be applied. Océ already has knowledge about this subject and we will contact sources within Océ to see if they can be used in our own research. After that we will investigate how existing cloud services match with the workflows that are chosen, and if they can be integrated with our driver, or that we have to build a cloud service of our own.

Deliverables: Research report resulting from the investigations, describing workflows used and explanation of the choices made. Architectural document on used cloud service(s).

Definition of Done: Completed research report, with enough information that a concept can be built from it. All deliverables ready and approved.

1.5 Phase 5 – Develop cloud service integration prototype

Goal: Develop a prototype for cloud service integration. This prototype is still decoupled from the driver.

Activities: Developing a prototype for cloud integration. This is just discovering how we could use cloud services in a print driver. This might involve coding examples and small proof of concepts. From these examples and prototypes we then choose one or more to implement in the real driver.

Deliverables: The considerations made for making a choice will be documented. That report, together with the examples and prototypes are the deliverables for this phase.

Definition of Done: We have a cloud printing prototype ready for implementing into the driver, matching the desired customer workflow(s). All deliverables ready and approved.



1.6 Phase 6 – Implement concept into existing driver

Goal: Create an actual implementation of our cloud printing prototype, developed in phase 5, in the driver we developed in phase 2 and 3.

Activities: We will use the prototype that already has been written in phase 5 as an example for the implementation. Depending on the outcome of the research we will also be implementing a cloud service ourselves, but the focus will be the driver implementation. The steps of the implementation will be documented.

Deliverables: The report on implementing the prototype (design document), and the finished driver. This driver will be used as backup demo product for school, in the case the driver will not be expanded any further.

Definition of Done: A completed, tested driver with cloud service integrated. All deliverables ready and approved.

1.7 Phase 7 – Test solution on a population of users

Goal: Test our completed driver with cloud integration on a group of users, and document their reactions and findings.

Activities: We will be submitting the finished driver to a group of users, to test what they find of our concept. This will be a usability test, to see if our solution is intuitive and easy to use, and if users like the added functionality of cloud services. The findings of the tests will be documented.

Deliverables: The test report.

Definition of Done: The finished test report, ready and approved.

1.8 Phase 8 – Writing Internship report

Goal: Writing the major part of the internship report. This phase will run throughout almost the entire internship, to prevent stress at the end of the internship.

Activities: Writing the internship report, as complete as possible. Every phase will be a chapter in the final report, so every phase, we will actually write the chapter, based on the other documentation made during that phase.

Deliverables: A preliminary version the internship report.

Definition of Done: Submitting the preliminary version of the report to the company mentor and the legal department for feedback.



Canon
CANON GROUP

1.9 Phase 9 – Finalizing Internship

Goal: Finishing the internship report and finalizing the internship.

Activities: Processing the feedback given by the company mentor and legal department, and resubmitting the report for checking. This first step will be repeated as many times as necessary. After that, the report is sent to the school mentor for review, and also his feedback will be processed, and the document will go through the first step again. When everyone involved is satisfied with the document, the final version will be made and offered to the school mentor, as most important deliverable. Based on the finished report, a presentation will be made, which will be rehearsed during the second visit of the school mentor to the company. The presentation might also be presented in the Cloud Café within Océ.

Deliverables: Internship report, and presentation for defence of the report.

Definition of Done: Maurice Wingbermühle is ready for his internship assignment defence / final session. This means that all deliverables are ready and approved, the presentation is ready and there is nothing more to do other than the final session itself.

1.10 Phase 10 – Optional: Enhance / Expand printer driver

Goal: Enhancing the driver / Expanding the driver with additional functionality.

Activities: After processing feedback on review of documents in phase 9, we extend the printer driver by implementing more features and functionality to the printer driver. If it is possible we will use the result of this phase as demo product during the final session.

Deliverables: Updated driver, and if required, updated documents.

Definition of Done: All deliverables, should these exist, ready and approved.



2 Planning Matrix

We have matrix showing the phases and how they are divided over the available weeks:

Week	Phases									
	1	2	3	4	5	6	7	8	9	10
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										
16										
17										
18										
19										
20										



Canon
CANON GROUP

Assignment Description & Test Case

& Test Case

author(s)

Maurice Wingbermhle



1 Assignment Description

With the introduction of Windows 8, Microsoft introduced printer driver model v4 [allowing use of built-in constraint handling, Bidi communication, Metro style user interfaces, etc...]

We're interested in finding out exactly how we can benefit the most from the new architecture. In particular we're interested in the Metro style; how this works from an end-user perspective, and if (and how) this opens up new paths for cloud printing services.

Goal of this assignment is to:

- Make a proof of concept printer driver for Microsoft's model v4 [loosely based on one of our printer systems], with
 - a metro style UI,
 - built in constraint handling
 - Bidi communication [for printer status information]
- Investigate how this model can be used for cloud printing

2 Test Case

Context: Windows 8 will be suitable for tablets. These mobile devices open up new ways of using our printer systems. We're interested in finding out how this can be used best. In this context, one of the questions that we have is how we can benefit the most of cloud printing services.

- What should this interaction to cloud services be?
- What do competitors (Apple, Android) offer?
- Which customer workflows exist?
- Can we support these?

Test case: Using Windows 8 and the v4 driver architecture, create a metro-style printer driver with a limited subset of job settings. Investigate if the printer driver application can be extended to make it work as a metro-styled job submitter application as well as a driver.

Based on your investigations on driver / cloud interaction, extend the driver/job submitter application to make best use of cloud service functionality.

Optional questions:

- Would it be helpful if we made our own cloud services?
- How could we make this? [prototype]



Canon
CANON GROUP

Creating a V4 Printer Driver

How to start on a V4 Printer Driver - XPS & PS, and JDF tickets

author(s)

Maurice Wingbermhle



Table of content

Foreword	3
1 V4 XPS Printer Driver	4
1.1 Create from template	4
1.2 Editing files	9
1.2.1 GPD configuration file	9
1.2.2 XPSDrv Filter Pipeline Configuration	9
1.2.3 INF File (Setup Information File)	9
1.3 Building the Driver	11
1.4 Testing the Driver	12
1.4.1 Installing	12
1.4.2 Printing on the virtual printer	14
1.4.3 Results	14
2 V4 PS Printer Driver	15
2.1 Creating from template	15
2.2 Editing the files	16
2.2.1 PPD (PostScript Printer Description) Configuration file	16
2.2.2 XPSDrv Filter Pipeline Configuration	16
2.2.3 Create MSxpsPS.ppd	17
2.2.4 The Manifest file	18
2.2.5 INF File (Setup Information File)	18
2.3 Building the Driver	18
3 Creating Filters	19
3.1 Custom Filter: Print Ticket Parser	19
3.2 The other side: JDF Insert Filter	19
3.3 Configuration of the pipeline	20
4 Debugging the driver	21
4.1 XPS spool file	21
4.2 Common system errors	21
4.3 WPP Logging	21
4.4 Visual Studio Debugger	21
5 Testing	22
5.1 Tools	22
5.1.1 Notepad++	22
5.1.2 GhostView	22
5.1.3 Open XML Package Editor Power Tool	22



Canon
CANON GROUP

Foreword

This guide describes how to start a printer driver with the Microsoft Printer Driver Model version 4, for Windows 8, and Windows Driver Kit 8. This guide assumes you have a development environment as described in the document "Development Environment Installation Guide.doc".

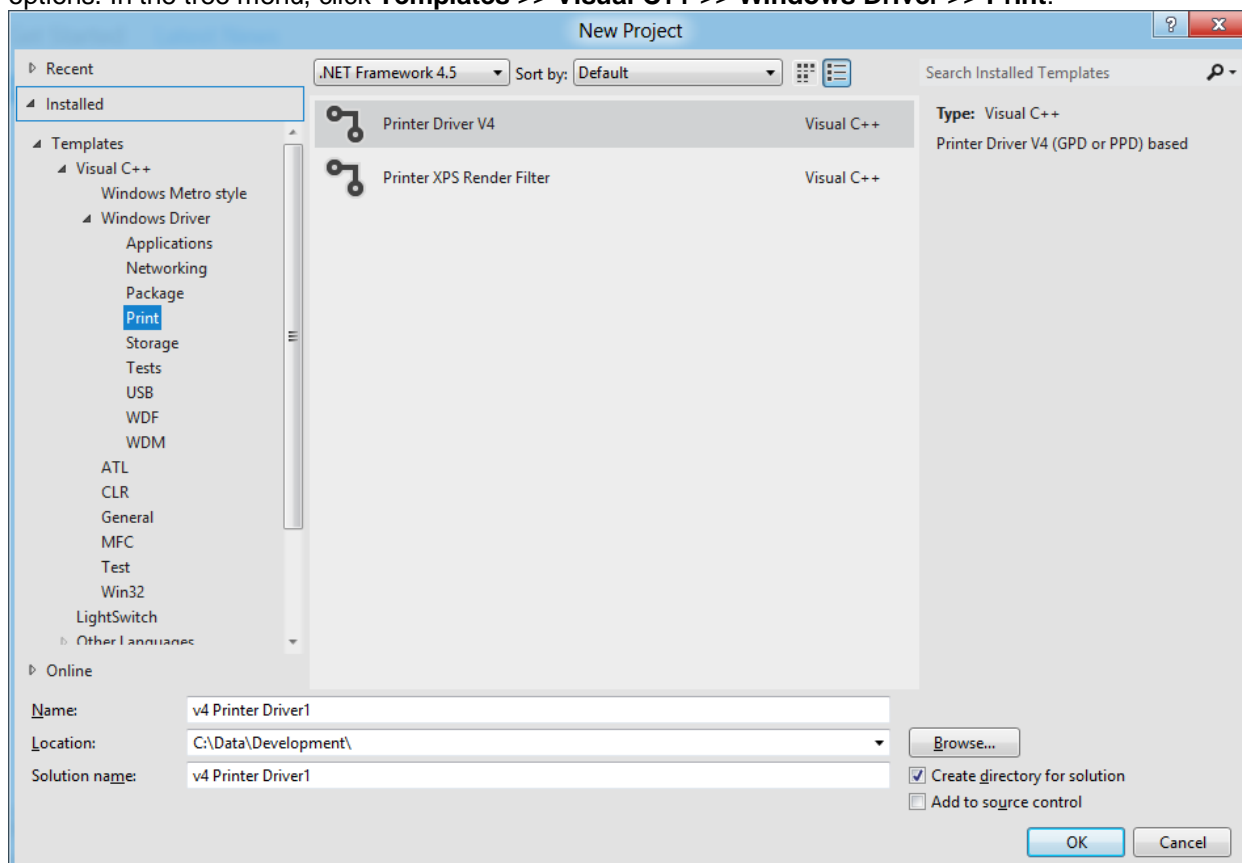
We will describe how to create an empty V4 XPS printer driver, an empty V4 PS printer driver and how to install and test them on your development machine.

1 V4 XPS Printer Driver

To create a empty XPS printer driver in the V4 printer driver model, we will use the default V4 Printer Driver template that Visual Studio 11 has to offer. Because Microsoft has provided us with a lot of default functionality in the new V4 printer driver framework, our “empty” printer driver will actually print XPS output by default, and still has a decent set of options in the printer driver. All we to do is set up a project with the correct configuration files.

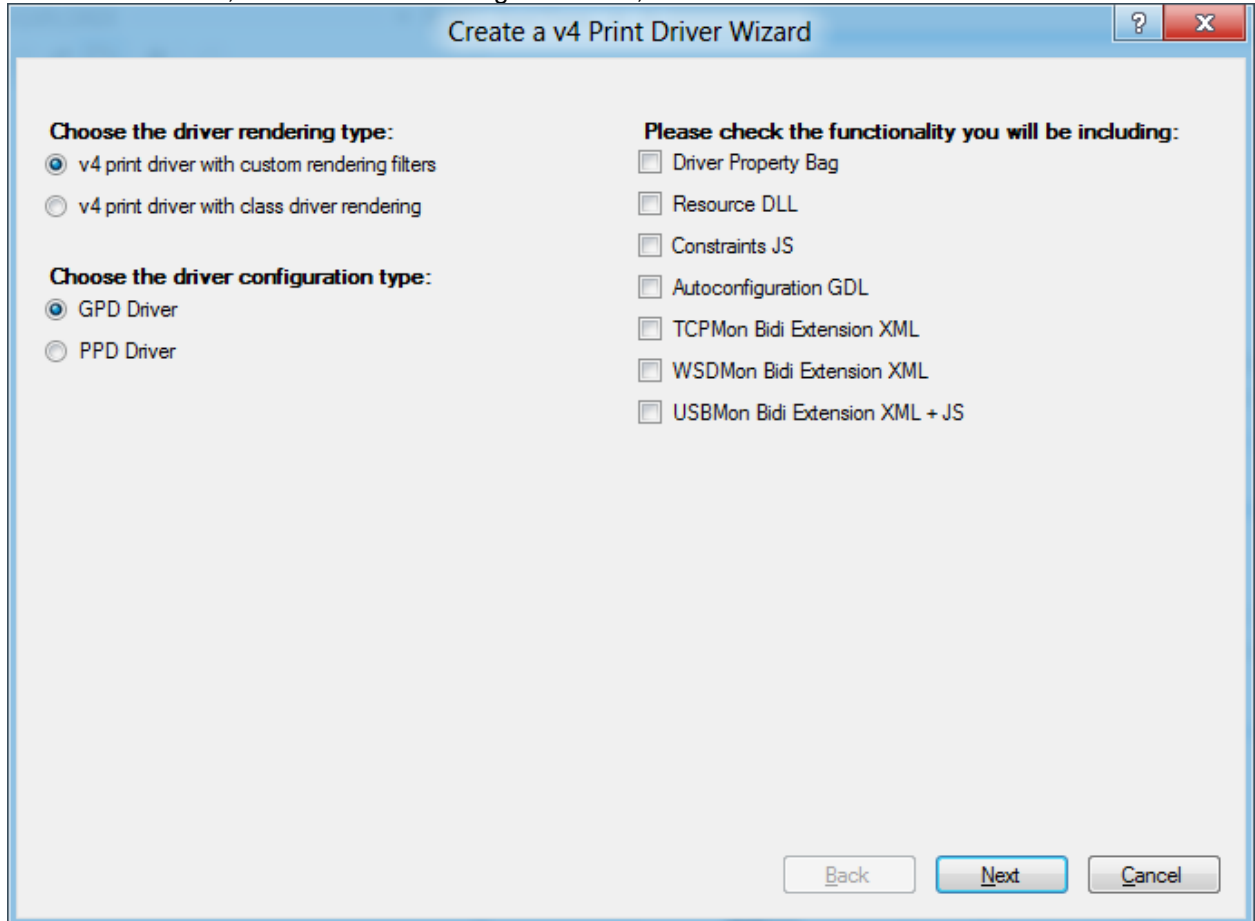
1.1 Create from template

Let’s start with Visual Studio 11. Create a new project and select “Printer Driver V4” from the template options. In the tree menu, click **Templates >> Visual C++ >> Windows Driver >> Print**:



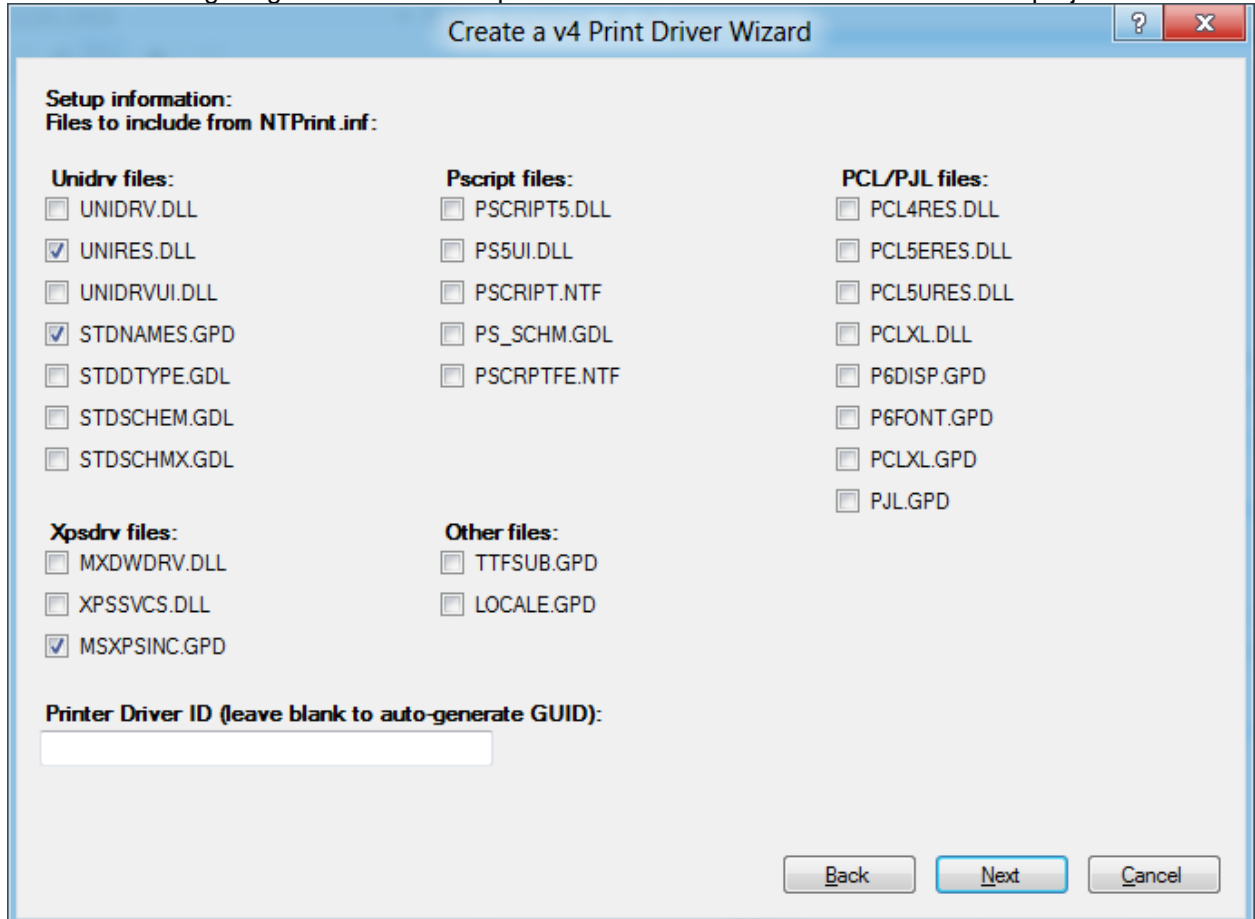
After selecting your desired project location and entering your project name, click “OK”. A couple of dialogs will appear.

For the XPS driver, we need a GPD configured driver, as shown in the next screenshot:



I won't elaborate on any other options than we need. The options as above are default, and that is how we want them for now. Click "Next".

In the next dialog we get to choose what provided resources we want to include in the project:



Create a v4 Print Driver Wizard

Setup information:
Files to include from NTPrint.inf:

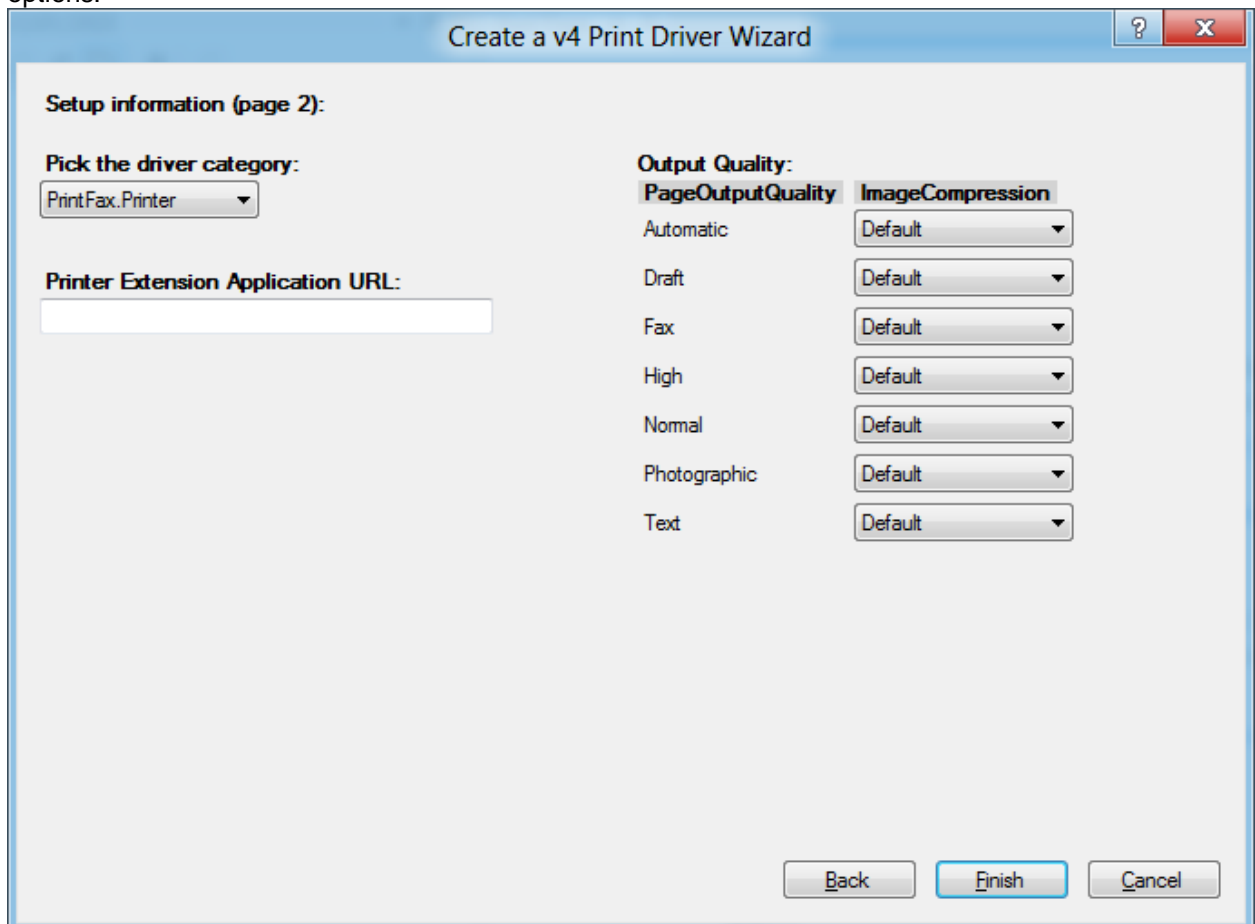
Unidrv files: <input type="checkbox"/> UNIDRV.DLL <input checked="" type="checkbox"/> UNIRES.DLL <input type="checkbox"/> UNIDRVUI.DLL <input checked="" type="checkbox"/> STDNAMES.GPD <input type="checkbox"/> STDDTYPE.GDL <input type="checkbox"/> STDSCHEM.GDL <input type="checkbox"/> STDSCHMX.GDL	Pscript files: <input type="checkbox"/> PSCRIPT5.DLL <input type="checkbox"/> PS5UI.DLL <input type="checkbox"/> PSCRIPT.NTF <input type="checkbox"/> PS_SCHM.GDL <input type="checkbox"/> PSCRPTFE.NTF	PCL/PJL files: <input type="checkbox"/> PCL4RES.DLL <input type="checkbox"/> PCL5ERES.DLL <input type="checkbox"/> PCL5URES.DLL <input type="checkbox"/> PCLXL.DLL <input type="checkbox"/> P6DISP.GPD <input type="checkbox"/> P6FONT.GPD <input type="checkbox"/> PCLXL.GPD <input type="checkbox"/> PJL.GPD
Xpsdrv files: <input type="checkbox"/> MXDWDRV.DLL <input type="checkbox"/> XPSSVCS.DLL <input checked="" type="checkbox"/> MSXPSINC.GPD	Other files: <input type="checkbox"/> TTFSUB.GPD <input type="checkbox"/> LOCALE.GPD	

Printer Driver ID (leave blank to auto-generate GUID):

Back Next Cancel

Also in this screen you have to leave everything default, unless you really know what you are doing. Click "Next".

In the next dialog we can select additional driver information, a couple of descriptive settings and default options:



Create a v4 Print Driver Wizard

Setup information (page 2):

Pick the driver category:
PrintFax.Printer

Printer Extension Application URL:
[Empty text field]

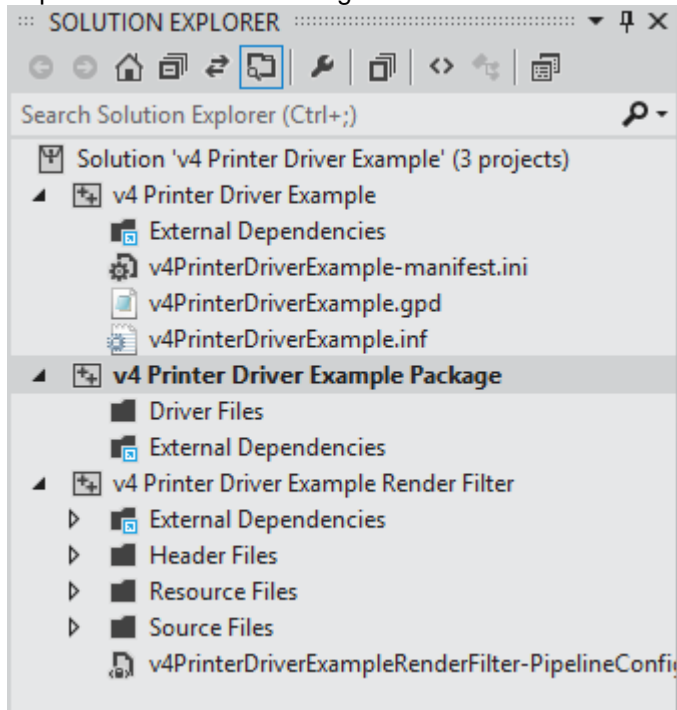
Output Quality:

PageOutputQuality	ImageCompression
Automatic	Default
Draft	Default
Fax	Default
High	Default
Normal	Default
Photographic	Default
Text	Default

Back **Finish** **Cancel**

We only want to change 1 option on this page, and that is the Driver Category. I prefer **PrintFax.Printer** to any other option, but we will use the printer as a file printer later on. I have not seen any influence on the behaviour of the printer driver, but it might affect the image Windows displays in the Devices and Printers screen. Leave everything else as is, and click "Finish".

After Visual Studio finishes with creating the project, you get a solution with 3 projects. Your Solution Explorer will show something like this:



1.2 Editing files

1.2.1 GPD configuration file

The current state of the solution will build but it won't create a installable package, in order to do that, we have to make some changes to 3 files. Let's start from the top, with the gpd file.

The GPD file is the main configuration file of the printer driver. It contains information about the capabilities of the printer and their default options, and the model name. You don't have to modify much here, just fill in the model name with anything you like (preferably something descriptive), and remove the `*PrintSchemaPrivateNamespaceURI` directive because we don't have such thing yet. Save the file.

1.2.2 XPSDrv Filter Pipeline Configuration

Next up is the PipelineConfig.xml file. When you open the file, you see that there is an entry for 1 filter. This filter is included in the project. Because we want the driver to be empty, we have to remove the filter from the pipeline. Delete the `<Filter>` entry, so the xml file only contains "`<Filters></Filters>`". If you plan on adding the filter again later on, you probably want to save the `<Filter>...</Filter>` snippet in a text file somewhere safe. When done, save the file.

1.2.3 INF File (Setup Information File)

The last file we need to edit, is the most important file, since this is the file Windows reads to determine what files to copy where, and what printer is installed. It's the inf file. Open it and start from the top. The first directive you need to edit, is the driver version **DriverVer**. For my test drivers I personally use driver versions under 1.0, so something like 0.1.

Next we skip a few lines and go to the `[Standard.NT$ARCH$]` indicator. Under this indicator we need to declare the printer driver models that can be installed with this driver package. For now, just do one, like this:

```
[Standard.NT$ARCH$]
"Clean V4 XPS" = CLEANV4_XPS, CLEAN_V4_XPS_Printer_HWID
```

I shall explain a bit what we are doing right here. The first section "Clean V4 XPS" is the name of the model of the printer that Windows will pick up and use as printer name. After the equation sign I have declared a indicator to the installation instructions, and after the comma, I have placed a fake hardware ID. You place a Hardware ID here, or a GUID, so when Windows is installing a real printer, it will use these IDs to match your driver with a certain printer model. Remember that this field is mandatory, and you will need to enter a complete entry. If you don't add the Hardware ID, the INF file will not even pass validation and your driver build will fail.

I have just declared a indicator to the installation instructions for our printer model, but that indicator does not exist in the file yet. Let's do that now.

Create a new indicator, and add the CopyFiles directive to it, like this:

```
[CLEANV4_XPS]
CopyFiles=CLEANV4_XPS_FILES
```

It is possible to add a lot more directives to this indicator, but we only need one. As you can see I've made up another new indicator after the CopyFiles directive. The directive instructs the installer to copy the files mentioned under a certain indicator.

Now also create the file copy indicator, and list all files you need to copy to the DriverStore, like this:

```
[CLEANV4_XPS_FILES]
CleanV4XPS.gpd
CleanV4XPS-manifest.ini
CleanV4XPSRenderFilter-PipelineConfig.xml
```

Now the installer knows what files to install, but not where. To make that clear to the installer you have to add all the files to the SourceDisksFiles indicator, that already exists a few lines up.

```
[SourceDisksFiles]
CleanV4XPS.gpd = 1
CleanV4XPS-manifest.ini = 1
CleanV4XPSRenderFilter-PipelineConfig.xml = 1
```

You can see I've added " = 1" to all lines. This is a reference to the following lines:

```
[SourceDisksNames]
1 = %DiskName%,,,
```

You can leave those lines, they indicate the installer where to put the files. The "1" in the SourceDisksFiles section refers to the "1" in the SourceDisksNames section. Do not forget the " = 1" after the filenames in the SourceDisksFiles section, because that will result in a validation error and the build will fail.

With that we're almost finished. You only need to fill in the manufacturer name, in the Strings section, like this:

```
[Strings]
ManufacturerName="Océ Technologies B.V. - R&D Dept."
DiskName="CleanV4XPS Installation Disk"
```

Save the file and you are done, concerning the editing. Next step: Building.



Canon
CANON GROUP

1.3 Building the Driver

This should not be difficult. Before you hit the magic button, be sure to select the right processor architecture. If you are developing and testing on a 64 bit machine, you will have to change the build target to x64, instead of Win32, in the Solution Platforms box in you toolbar. With that right, hit [F7] or click Build >> Build Solution. If you did everything right, the solution should build without any further problems, and you get a package as result. Now we are ready for testing

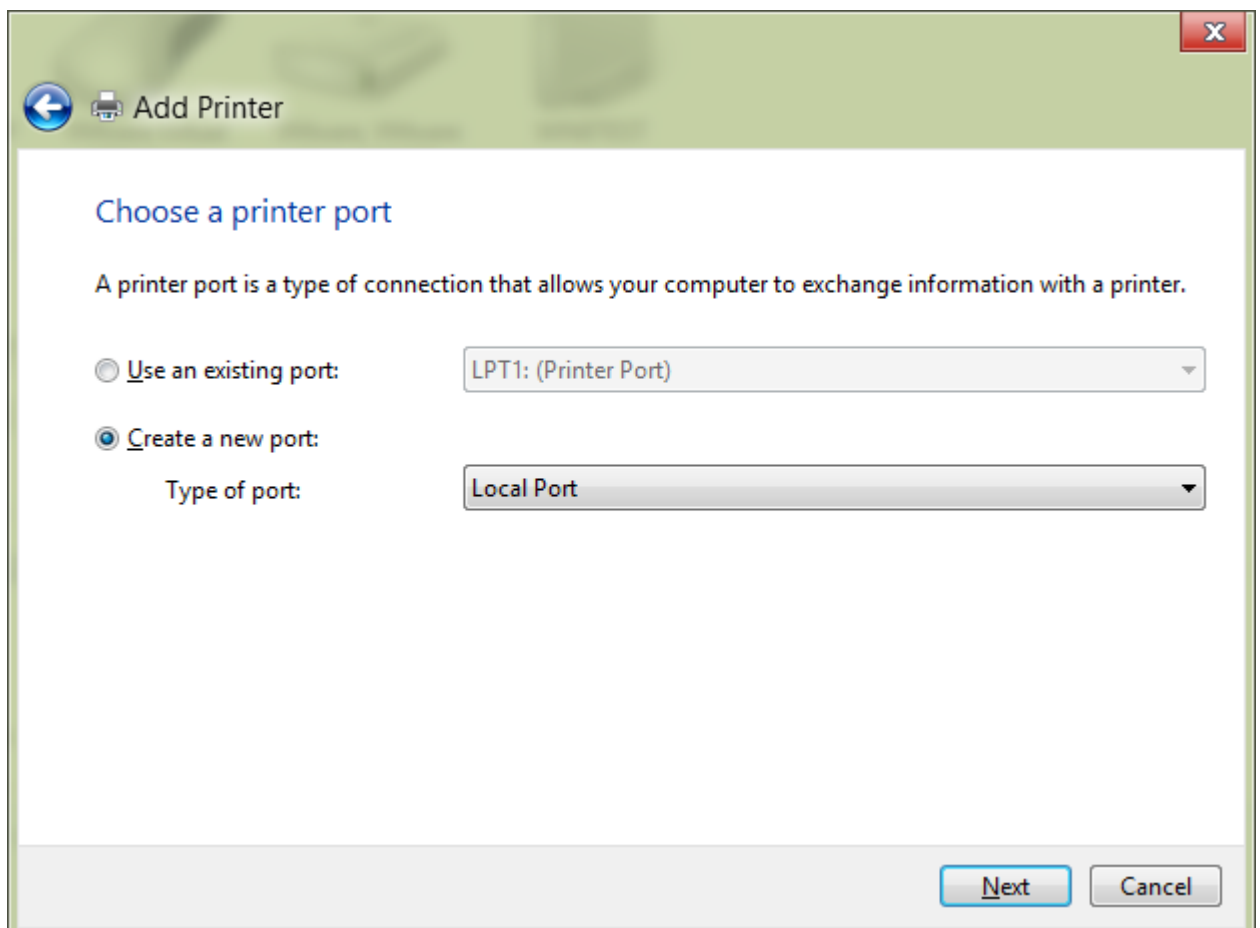
1.4 Testing the Driver

The test consists of 2 parts, installing and making the first print.

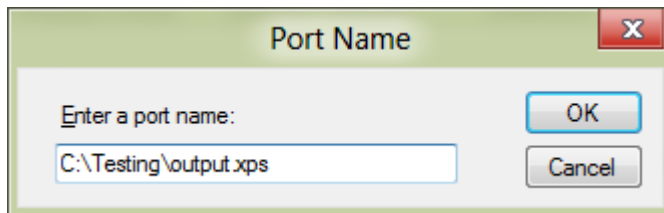
1.4.1 Installing

Hit the Win key on your keyboard (or open start anyway you want), and type Printer. This will invoke search. Select the “Settings” tab to see the search results in the Settings. Open the Devices and Printers screen. Click “Advanced printer setup” to install a new printer with our new driver.

Windows will start searching for new printers on your system and the network. Since our printer does not even exist, click “The printer that I want isn’t listed” on the bottom of the dialog. The next dialog will appear, asking how you would like to find another printer. Select the last option “Add a local printer or network printer with manual settings.” and click Next.

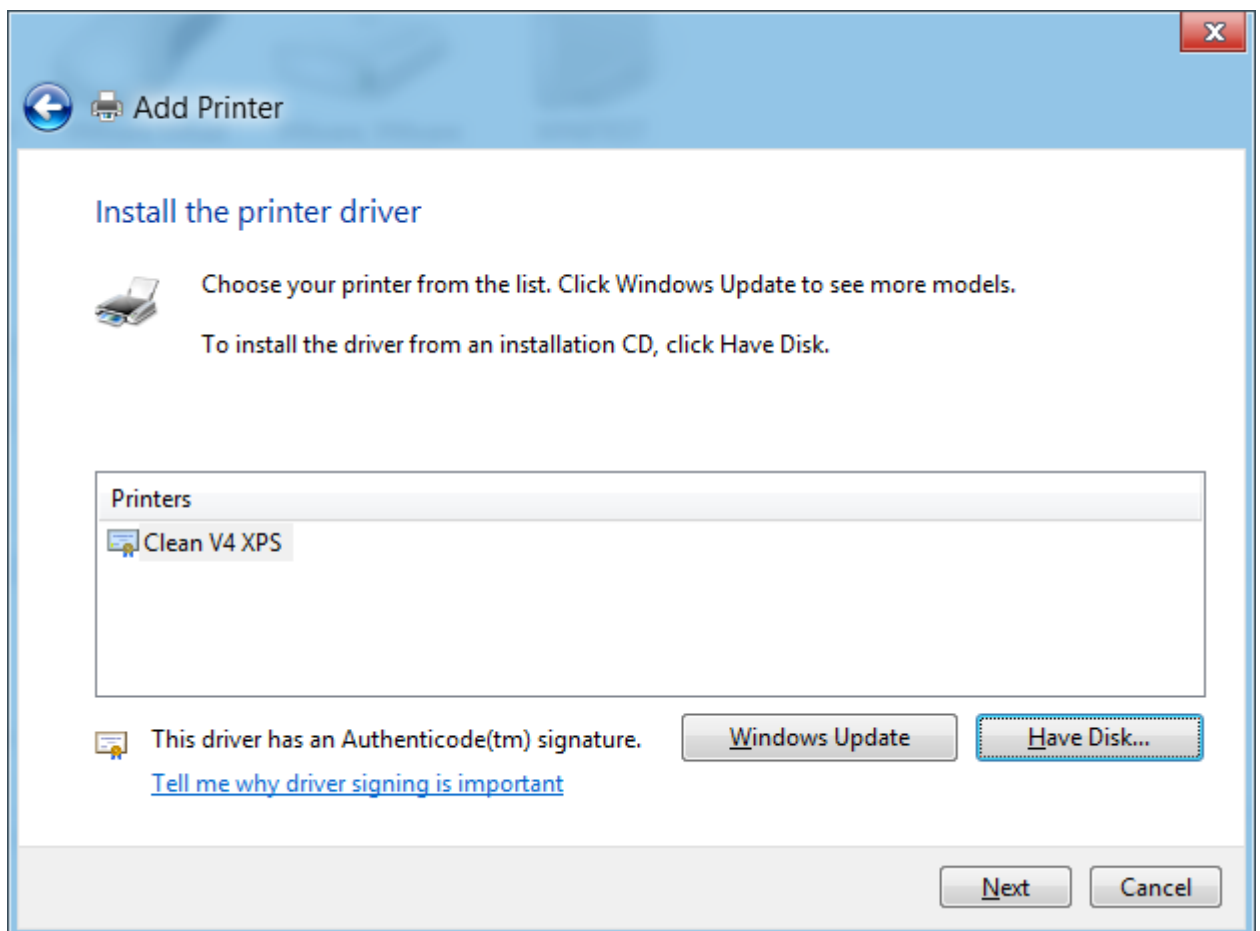


The dialog as in the image above will appear. Since we do not yet have our own printer port yet, select “Create a new port”. We want to create a new local port, so make sure that is shown in the dropdown box. Click Next.



The installer will prompt for a port name. You can do 2 things here. Either type a full file path (does not have to exist yet), like I did, or just type in a short name. The short name will result in a file with that name in your personal documents folder on the machine you're testing on. The full path will result in a file on the location you wish. Note that this file will only exist after the first print job, and not yet after installation. Make sure you have writing access on the location you enter. Click OK.

In the next dialog, click "Have Disk" because we manually want to select the location of our printer driver. The installer will prompt for a location. Enter the location of the built driver. This should be the location of your project, and then in the folder x64 >> Win8betaDebug >> Package. The x64 part is of course only for x64 machines, Win32 drivers are directly in the Win8betaDebug folder. You will notice that there is also a INF file in the Win8betaDebug folder, but it is important that you select the one in the package folder. This is because only the package folder has a signed catalog (.cat) file. Click OK when finished.





Canon
CANON GROUP

In the next dialog you can see that the installer has already read the INF file, because it shows the printers it can install from the INF file. We have only added 1 printer, so there is not much choice. Click Next.

The installer will ask for a name, edit the name, or leave it as is. Clicking Next will install the printer. When the installer is done with the operations and does not encounter an error, it will show the final page of the installer dialog. Set the driver as default if you want, and finish the dialog.

1.4.2 Printing on the virtual printer

The next step is to print something on the printer we just installed. Open notepad, enter the text "Hello World!" and hit Ctrl + P or File >> Print. Leave the print options default, and click Print. Make sure you have selected the correct printer, we don't want some rubbish to come out of your real printer.

If everything went right, we just made a print with our own driver. Open an explorer and navigate to the location where your printers local port should output. If everything really did went well, you should see a file with a very recent timestamp, and 22KB of size. Depending on the name you gave, you might or might not have a .xps file extension. If you have, go ahead and open it. If not, rename the file so you have, and then open it. In Windows 8, we have the Windows Reader that can open XPS files. Check if you see a document with our "Hello World!" text and there should be a header and footer as well.

1.4.3 Results

If you did not make it to the end but got an error during either installation or printing (or no print file appeared), check if you did not miss a step of the editing of the files and the building section. If you have verified that all steps are correct, you can check the Windows event and setup logs for any clues.

Windows Event Logs:

Start: Search "event" >> Settings >> View event logs

INF Setup Log:

Notepad: %windir%\Inf\setupapi.dev.log

2 V4 PS Printer Driver

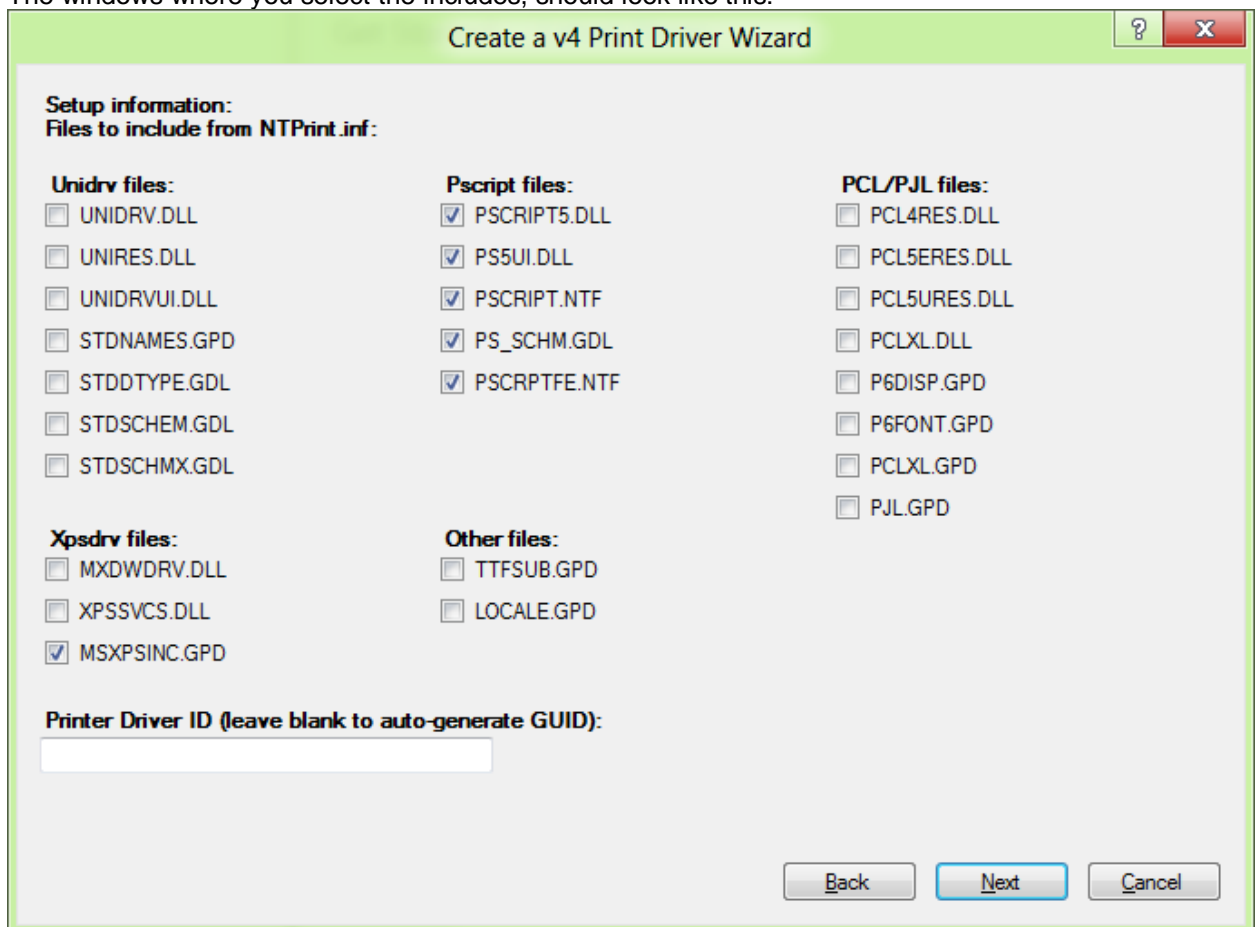
Microsoft provides us with everything to create PostScript or PCL6 printer driver. They have PS and PCL filters ready for use. The clue is to correctly configure them. Due to the lack of decent documentation at this moment, I've decided to expand my guide to incorporate the PS filter in a empty XPS driver.

Before you start this chapter, you have completed the first chapter, because I will skip through some steps here, and refer to some steps from that chapter.

2.1 Creating from template

The first 2 steps are the same as in paragraph 1.1, except for the GPD configuration option, now select the PPD configuration option.

The windows where you select the includes, should look like this:



Create a v4 Print Driver Wizard

Setup information:
Files to include from NTPrint.inf:

Unidrv files:	Pscript files:	PCL/PJL files:
<input type="checkbox"/> UNIDRV.DLL	<input checked="" type="checkbox"/> PSSCRIPT5.DLL	<input type="checkbox"/> PCL4RES.DLL
<input type="checkbox"/> UNIRES.DLL	<input checked="" type="checkbox"/> PS5UI.DLL	<input type="checkbox"/> PCL5ERES.DLL
<input type="checkbox"/> UNIDRVUI.DLL	<input checked="" type="checkbox"/> PSSCRIPT.NTF	<input type="checkbox"/> PCL5URES.DLL
<input type="checkbox"/> STDNAMES.GPD	<input checked="" type="checkbox"/> PS_SCHM.GDL	<input type="checkbox"/> PCLXL.DLL
<input type="checkbox"/> STDDTYPE.GDL	<input checked="" type="checkbox"/> PSRPTFE.NTF	<input type="checkbox"/> P6DISP.GPD
<input type="checkbox"/> STDSCHM.GDL		<input type="checkbox"/> P6FONT.GPD
		<input type="checkbox"/> PCLXL.GPD
		<input type="checkbox"/> PJL.GPD
Xpsdrv files:	Other files:	
<input type="checkbox"/> MXDWDRV.DLL	<input type="checkbox"/> TTFSUB.GPD	
<input type="checkbox"/> XPSSVCS.DLL	<input type="checkbox"/> LOCALE.GPD	
<input checked="" type="checkbox"/> MSXPSINC.GPD		

Printer Driver ID (leave blank to auto-generate GUID):

You have to select all the Pscript files, and the MSXPSINC.GPD file.

The rest of the steps is equal again to the steps taken in paragraph 1.1.

2.2 Editing the files

I will step a bit faster to these steps and not explain the things that I already explained in chapter 1. There is more to edit here.

2.2.1 PPD (PostScript Printer Description) Configuration file

Instead of a GPD file, we now have a PPD file. Let's start with that file.

```

*%*****
*%: The following root-level attributes should be modified to suit your printer
*%*****

*FormatVersion: "4.3"
*FileVersion: "1.0"
*LanguageEncoding: ISOLatin1
*LanguageVersion: English
*Product: "Clean V4 PostScript Printer"
*PSVersion: "(2015.105) 9"
*Manufacturer: "Océ Technologies B.V."
*ModelName: "Clean V4 PostScript"
*ShortNickName: "Clean V4 PS"
*NickName: "Clean V4 PS"
*PCFileName: "CleanV4PS.PPD"

*MSXPSMaxCopies: "999"
*PrintPSErrors: True

```

These are the changes that I have made to this file. You should do something similar. Remember to remove the ***MSPrintSchemaPrivateNamespaceURI** directive. I have also added the ***PSPrintErrors** directive with True.

2.2.2 XPSDrv Filter Pipeline Configuration

First make the same changes to the PipelineConfig.xml file as in paragraph 1.2.2 so we get a empty <Filters> tag.

Next we will add a new filter tag that adds the PostScript filter to the pipeline. The filter tag is a snippet that can be taken from "Developing v4 Print Drivers.docx", paragraph 4.1.5.2.2, a document that Microsoft provides on their developer website:

```

<Filter dll="MSxpsPS.dll"
  clsid="{8636D90A-5E03-4d62-9269-E06493C57473}"
  name="Microsoft XPS to PS">
  <Input guid="{4d47a67c-66cc-4430-850e-daf466fe5bc4}" comment="IID_IPrintReadStream" />
  <Output guid="{65bb7f1b-371e-4571-8ac7-912f510c1a38}" comment="IID_IPrintWriteStream" />
</Filter>

```

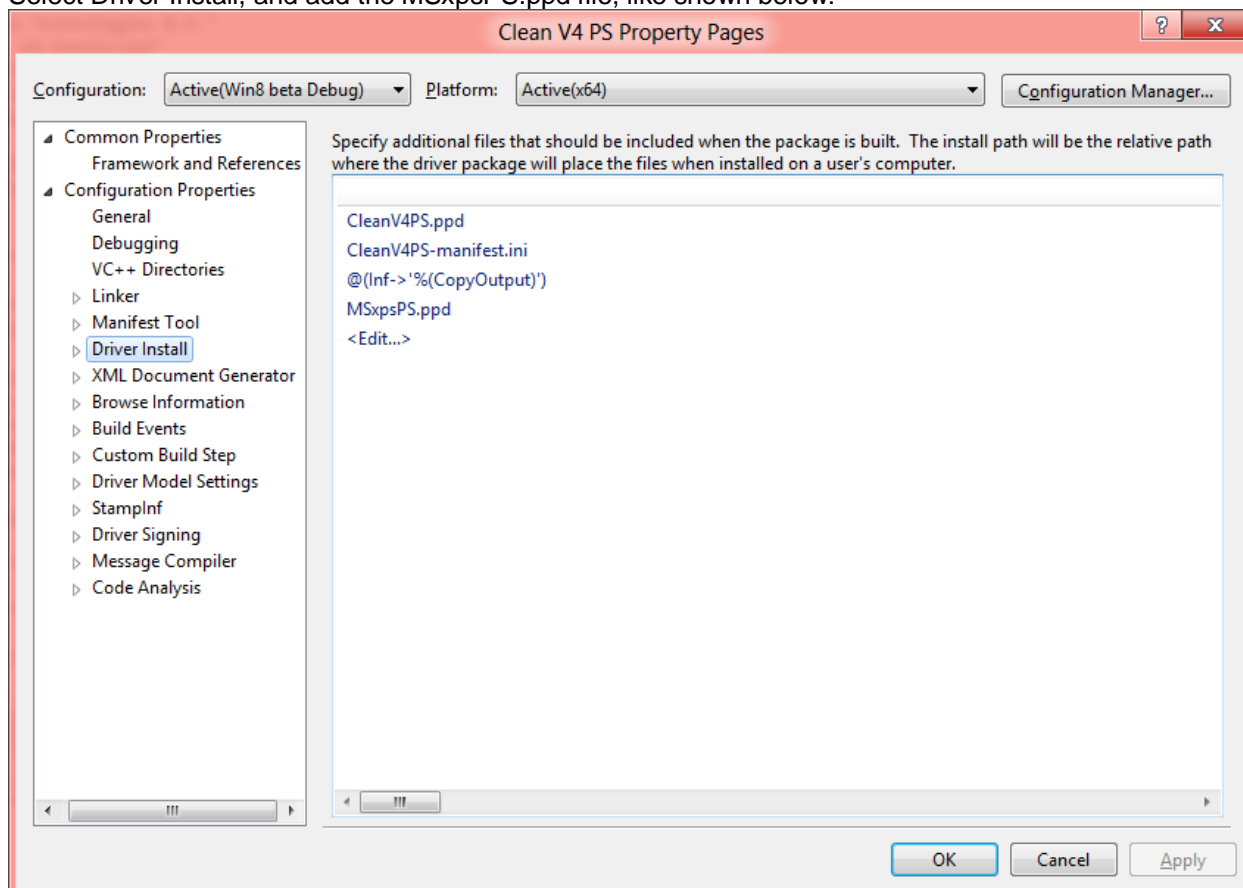
You now have 1 filter in the pipeline again.

2.2.3 Create MSxpsPS.ppd

In order to get the MSxpsPS filter to work, it needs its own PPD file with the correct name and corresponding options to the global configuration. Instead of creating a new file, I copied the existing PPD file and renamed it. Open an explorer and navigate to your project folder, and look for the main PPD file. When you find it, copy and paste it in the same folder, and rename the copy to "MSxpsPS.ppd" (and nothing else!).

Now go back to Visual Studio. We need to instruct Visual Studio that it needs to take the PPD into the project and output directory when building. First add the file to the project, by right-clicking the top project of the solution, and select Add >> Existing Item. Browse to the file and select it. Now it's in the project, but it won't get copied to the Package folder. To accomplish this, right-click the project again and select Properties.

Select Driver Install, and add the MSxpsPS.ppd file, like shown below:



When done click Apply first, to see if it's correctly added, then click OK.



Canon
CANON GROUP

2.2.4 The Manifest file

We also need to make change to the manifest.ini file. This is the file that describes all the includes.

On the RequiredFiles directive, add MSXPSPS.DLL (pay attention, DLL, not PPD). When done, it should look like this:

```
RequiredFiles=PSCRIPT5.DLL,PSSUI.DLL,PSCRIPT.NTF,PS_SCHM.GDL,PSCRPTFE.NTF,MSXPSINC.GPD,MSXPSPS.DLL|  
DriverCategory=PrintFax.Printer
```

2.2.5 INF File (Setup Information File)

You should be able to do this on your own, following the steps from paragraph 1.2.3. You should of course replace the gpd with ppd and do not forget to include the MSxpsPS.ppd file.

This concludes the editing steps for the PS driver.

2.3 Building the Driver

Same as described in paragraph 1.3

3 Creating Filters

3.1 Custom Filter: Print Ticket Parser

To begin with the first filter, the XPS Print Ticket Parser, we start with the PostScript project that we prepared in chapter 2. We'll leave the pipeline configuration for what it is now because we cannot complete it yet. The default template has a lot of code for parsing PrintTicket, so if you we're stubborn enough to actually delete the original code, you'd better start over again. The code that we are looking for is in the RenderFilter project, inside the solution. There are only 2 files inside this project that are really interesting for us at this moment and that are PTHandler.cpp and RenderFilter.cpp. To give you a bit of context: dllentry.cpp handles the start and finish of the DLL calls. It creates a instance of RenderFilter.cpp, and calls the StartOperation function. RenderFilter.cpp is the real pipeline segment. It has the interfaces for the input and output and makes further calls to, among others, the Print Ticket handler PTHandler.cpp.

The template "as-is" is actually working pretty good, but we need to modify a few things to make it useful. For starters, we want our XPS Parser to also have a XPS output, instead of a Stream, so we need to modify that in code, and in the pipeline configuration, because the writer in the template works with the Stream interface. When you try to build and install this printer driver, it will probably succeed, but it will not output any PostScript. That is because the RenderFilter.cpp does not output anything, it just tries to parse and validate the Print Ticket. To change that, we need to go to the code in RenderFilter.cpp where the PTHandler is called to do it's magic on the XPS parts. Apart from sending the XPS parts to the PTHandler, we also want to output these same parts, to make our filter "pass-through" again. Feed the XPS parts to the writer with the appropriate send functions to make the filter output again.

Next, we need to add a function to the PTHandler to retrieve the validated PrintTicket. You can choose how you want to do it, but I made a GetData function which takes a reference to a string and fills that with the raw XML from the validated print ticket. This should not be to hard.

Back to the RenderFilter. After having fed all the XPS parts to the PTHandler, we retrieve the validated print ticket with our home made function, put that into the pipeline PropertyBag. The RenderFilter already has a instance of the PropertyBag interface, so the only thing you need to do is call the AddProperty function with the right parameters.

3.2 The other side: JDF Insert Filter

With that done, we finished the XPS Parser, so let's continue with the JDF Insert Filter.

To create the JDF filter, we need to create another project in the solution, again, based on a template. This time we don't need the entire printer driver template, but just the Render Filter template. For this filter we want Stream in and output so modify the interfaces to match that. The reason for that is quite simple: We're behind the MSxpsPS converter filter, so we cannot expect XPS output from that. We want to output JDF, and then place PostScript output of the MSxpsPS filter behind that.



The first thing you want to do in this filter, is reading from the input, and store that in a buffer for a bit. That is because the V4 architecture is pseudo multi threaded. It will try to do as much as possible in a parallel way, so initializing the filters and variables will happen parallel if the pipeline does not detect data requirements. With placing a read statement, we indicate that we first want the data of the previous pipeline segment to be ready before anything else in our pipeline can happen. This is important because we want to read from the propertybag, and if the previous filter is not yet finished, it could be that the value we want to read does not exist yet.

Read the PrintTicket from the propertybag, which we set in the XPS Parser filter. We then need to read the important value from the XML. We should parse the XML file, but because of time issues, I was forced to choose for the string search option instead. I injected the retrieved values for stapling, simplex / duplex, number of copies, and collation into a static JDF ticket that I copied from another Océ printer driver output. Output the JDF ticket, and then write the saved buffer from the first read, and read and write the rest of the input. This can all happen in the RenderFilter.cpp, so you don't need the PTHandler, or any of the other files here. With this you should already be finished with this filter.

3.3 Configuration of the pipeline

For the configuration part, you should modify the pipeline-config.xml file to include the 3 filters in the right order as displayed in the previous image. You should also make sure your PPD files configure the features you want to be using, because otherwise they won't show up in the Print Preferences window Microsoft generates for you.



4 Debugging the driver

Of course we needed debug tools to analyze the driver and your system in case something goes wrong. There are various places to retrieve information about the printer driver. I'll take you through a couple of them.

4.1 XPS spool file

It is possible to retrieve the raw unprocessed spool file. You first have to enable a checkbox in Printer Properties >> Advanced tab >> Keep printed documents. After that, open an explorer and navigate to C:\Windows\System32\spool\PRINTERS\ and copy the .spl file to, for example, your desktop. You can either rename the file to .zip and open it with any compatible program, or you can rename it to .xps and open it with Visual Studio 2010 (yes, not yet 11) with the Open XML Package Editor Power Tool installed. Get it at <http://visualstudiogallery.msdn.microsoft.com/450a00e3-5a7d-4776-be2c-8aa8cec2a75b>

4.2 Common system errors

To view common errors with the printer driver, or installation of the driver, you can use the Windows Event Viewer. Search for "event" in Windows 8 Start, and you will get 1 hit in "Settings". I usually use the Custom view "Administrative Events".

4.3 WPP Logging

To view the WPP logging made by the printer driver, which is a very useful tool, you need a program, that can view the logging. I'm using ETViewer, a free, standalone executable that can be downloaded from <http://etviewer.codeplex.com/>. You have to open the PDB files that match with the DLL's you built for the driver, in order to actually see results in the logging.

4.4 Visual Studio Debugger

Probably the most powerful tool is the Visual Studio debugger. To start debugging, you have to create a registry key that forces the printer driver pipeline to stay alive for a specified amount of time, so you can attach to the process. The key you have to create is in "HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Print", and is called "PipelineHostTimeout". It should be a DWORD type, and the decimal value indicates the timeout value in milliseconds. So if you want a 2 minute timeout, like I did, use 120000 as value. Print a job to your driver to wake up the printfilterpipelinesvc.exe process, go to Visual Studio, press [Ctrl] + [Alt] + [P] and attach to the process. Set breakpoints in your code, and print another job to the driver. Now your first breakpoint should be hit (if your breakpoints are reachable). Remember that is the easiest if you let the installer copy the pdb files with the dll files. This way you don't have to tell Visual Studio where to find the files. You do need the pdb files for debugging, no breakpoints will be hit if they are not present. For more details about this, see MSDN: [http://msdn.microsoft.com/en-us/library/windows/hardware/ff545076\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/ff545076(v=vs.85).aspx)

5 Testing

To test my driver, I used Notepad++ to just read the print output. Especially for the JDF ticket, it is the easiest way to see if the output is valid. To make things easier, cut the JDF ticket and paste it to a new file, and save it with xml extension. This enables the Notepad++ syntax highlighting. To verify the PostScript, you have to strip the JDF file from the top before opening the .ps file with a program like Ghostview. As final test, I changed the printer port to a remote port of a Océ printer, managed by Testgroup. The first time I tested on these machines, it went wrong, and apparently I had the JDF mimetype directive set to XPS, instead of PostScript.

5.1 Tools

I have used some tools for testing, that are not related to the debugging tools, and I will share them with you here.

5.1.1 Notepad++

Probably the most popular text editor for programmers, because it can recognize various programming language and related configuration files. Notepad++ can apply context highlighting and that is particularly useful for analyzing and validating XML. The JDF ticket is XML, and because that is the part that we are modifying later on, it can be useful to check for typo's.

Open the print output file, select the XML and cut and paste it to a new file, and save that as XML. Notepad++ will that apply highlighting as soon as you save the file with the right extension (.xml). If you want to see the PostScript output, you can save the remaining content of the printer output to a .ps extension and open that file with another tool: GhostView.

You can download the latest version of Notepad++ at: <http://notepad-plus-plus.org/>

5.1.2 GhostView

For viewing the PostScript output of the printer driver you need another tool called GhostView. It uses GhostScript as parser / interpreter, and you will need to install that as well in order for GhostView to work. You can find the latest version of the tools at <http://pages.cs.wisc.edu/~ghost/>.

5.1.3 Open XML Package Editor Power Tool

If you have not yet enabled the PostScript converter filter in the pipeline yet, you should be getting XPS output. If you want to validate that output you can use this tool to browse through the zip structure, and analyze the raw XML files in Visual Studio. Be sure to strip the JDF from the rest of the output, as described in section 4.1.1. If you want to see the visual (rendered) output of the XPS file, you can open the file with the Metro style Windows Reader app.

See section 3.4 for more info and a download link.



Canon
CANON GROUP

Registry Keys for Desktop PrinterExtension

Registering a V4 PrinterExtension for Desktop

author(s)

Maurice Wingbermühle



Table of content

1 Get the Printer Driver ID	3
1.1 From Printer Driver Project	3
1.2 From a preinstalled printer	3
2 Create a Registry Key file	4
2.1 The base of the file	4
2.2 Replacements	4
2.2.1 PrinterExtensionID	4
2.2.2 PrinterDriverID	5
2.2.3 AppPath	5
2.3 Saving the file	5
3 Applying the registry key	6
4 Unregister	7



1 Get the Printer Driver ID

1.1 From Printer Driver Project

If you are developing the entire printer driver package like I am, it is easier to get the printer driver ID directly from the code. Open your V4 printer driver project and open the manifest file. The printer driver ID is placed on the second line by default (from the template). If you modified or created this file by yourself, you will probably know better where to look for it.

The keyword that you are looking for is `PrinterDriverID`, and you will find it under the `[DriverConfig]` directive, in a line like this:

```
PrinterDriverID={xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx}
```

Copy the entire section after the equals sign, including the curly braces { }.

1.2 From a preinstalled printer

If you did not develop your own printer driver, but have a preinstalled printer driver that you want to test your `PrinterExtension` with, you will have to do the following to get the printer driver ID.

1. Open a PowerShell
2. Copy the following text into PowerShell (copy – right mouse click). Replace `<printer name>` with the name of the printer queue that you want to use:

```
$printDriverID = Get-Content ((Get-Printer "<printer name>" | Get-PrinterDriver).DependentFiles | Where-Object -FilterScript: {$_. -like "*manifest.ini"}) | Where-Object -FilterScript: {$_. -like "PrinterDriverId*"} ($printDriverID.Split("="))[1];
```

If you wrote the name of the printer correctly, this should result in a GUID. Copy this GUID, including the curly braces { }.



2 Create a Registry Key file

To register the application, there are 2 ways to add keys to the registry. To make things easy, I will show you how to create a registry file (.reg), instead of using the registry editor.

2.1 The base of the file

Open Notepad, or another suitable text editor. Copy and paste the following text into the text editor:

```
Windows Registry Editor Version 5.00
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Print\OfflinePrinterExtensions]
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Print\OfflinePrinterExtensions\{PrinterExtensionId}]
"AppPath"="c:\\apps\\Printer extension
sample\\C#\\ExtensionSample\\bin\\Debug\\PrinterExtensionSample.exe"
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Print\OfflinePrinterExtensions\{PrinterExtensionId}\{Printe
rDriverId}]
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Print\OfflinePrinterExtensions\{PrinterExtensionId}\{Printe
rDriverId}\{EC8F261F-267C-469F-B5D6-3933023C29CC}]
@="1"
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Print\OfflinePrinterExtensions\{PrinterExtensionId}\{Printe
rDriverId}\{23BB1328-63DE-4293-915B-A6A23D929ACB}]
@="1"
```

The code above creates a number of registry keys with appropriate values. Before it can be used, you will have to modify it to fit your printer driver, PrinterExtension and system. That will come in a moment though. You can modify the base above. The last 2 keys of the base contain 2 different GUIDs. These are statically defined for interfaces on which your PrinterExtension can be called. The first GUID is for Print Preferences, and the second is for Driver Events. If you don't support driver events in your PrinterExtension, then you can remove the last key (from "[HKEY_" to "@=" 1").

To get more information about the registry keys, see "Developing V4 Printer Drivers", section 6.4.1.1.2 <http://msdn.microsoft.com/en-us/library/windows/hardware/br259124.aspx>

2.2 Replacements

You have to replace some values in the text above, to make it suitable for you printer driver and PrinterExtension. Make sure that the format of your input matches the values that are already in the base.

2.2.1 PrinterExtensionID

First replace {PrinterExtensionID} with a GUID that you have generated yourself. You can generate a GUID by using Visual Studio. In the Visual Studio, in the toolbar click Tools >> Create GUID. Select the Registry format option, and click the Copy button to copy the result to your clipboard. Use find and replace to replace all occurrences of the PrinterExtensionID with your new GUID.



2.2.2 PrinterDriverID

The second step is a bit simpler. Replace all occurrences of PrinterDriverID with the printer driver ID GUID that you retrieved earlier.

Note: The GUID that you retrieved is probably in lower case, but you have to convert that to upper case in order to be interpreted right into the registry. I used an online tool to convert my GUID, but you can do it manually if you want.

2.2.3 AppPath

The last value you will have to replace is the AppPath value. Browse to the physical location of your project on your hard disk with Windows Explorer. Find the .exe file of your build, which will probably be in your bin\Debug folder. Copy the address of the folder, appended with the executable name (so the full application path) to the text editor. Pay attention to the double backslashes in the application path as the base sample, and apply them to your path.

2.3 Saving the file

Now that we are done with the contents of the file, we are going to save the file. We want to have .reg file, which in fact is a specially formatted text file. So if you were working with notepad, you can save the file as a .reg file on a location of your own choice. You can also pick a name of your own, but I advise you to name it something like "<printer-name>-<printerextension>-install.reg". Just don't forget the location where you put it. If you were working with something else than notepad, you will have to make sure that the output is correct.



3 Applying the registry key

To complete the registration of the PrinterExtension, we only have to execute the file we just created. Being a .reg file, Windows will recognize it as a registry key file. When you double click on the file, Windows will prompt you with safety warnings and maybe even an authorisation dialog (you have to have Administrator rights to do this).

Please note that after installing the registry key, it can be found using the Windows registry editor (WIN+R >> regedit) on the location specified in the file:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows  
NT\CurrentVersion\Print\OfflinePrinterExtensions
```

However part of the keys will be removed from this location after running the PrinterExtension for the first time using the Print Preferences button in the print dialog. Pressing Print Preferences will trigger a service to check the registry key above and parse and apply the new child key(s). So if you think you made an error, you can fix that before running the PrinterExtension.

Instead of using the print dialog and Preferences button, you can also trigger the service on your own. The service is called PrintNotify and you have to restart it if it is running, or start it if it is not running yet. PrintNotify will then walk through the registry keys and process any unprocessed keys.



Canon
CANON GROUP

4 Unregister

To unregister your PrinterExtension, you do not have to do much. Take the install registry key file, open it with notepad, and replace the 2 occurrences of `@="1"` with `@="0"`.

Save the file under another name (I advice the same name, but with "uninstall" instead of "install" in the name) and run the file to apply the key. To have the key processed by Windows, open the Print Preferences in the print dialog. You do not actually have to print a document by the way, just cancel the print dialog when done.



Canon
CANON GROUP

Cloud Integration Research

What technologies and workflows exist in the cloud and how to integrate them in a V4 printer driver for Windows 8.

author(s)
M. Wingbermühle

Table of content

Foreword	5
1 Research Objective	6
1.1 Supporting Questions	6
2 What research has already been done?	7
3 What technologies can be used?	8
3.1 Data Transfer Protocols	8
3.1.1 HTTP	8
3.1.2 HTTPS	8
3.1.3 XMPP	8
3.1.4 Data transfer: Which protocols should be used?	9
3.2 Cloud Flavours	10
3.2.1 SaaS – Software as a Service	10
3.2.2 PaaS – Platform as a Service	10
3.2.3 IaaS – Infrastructure as a Service	11
3.2.4 SaaS, PaaS or IaaS: Which one to use?	11
4 What existing Cloud Services can be used?	12
4.1 A list of cloud services	12
4.2 Update on Dropbox	12
4.3 Google Drive integrates Docs	12
4.3.1 Comparison with Dropbox	13
4.4 Amazon Cloud Drive	13
4.5 Mimeo Connect Cloud Print	13
4.6 Compatibility between services	14
4.6.1 Google Drive and Cloud Print	14
4.6.2 Box.net and Google Docs	14
4.6.3 HP ePrint and Google Cloud Print	15
4.6.4 HP ePrint and Mimeo Cloud Print	15
4.6.5 AutoCAD WS and Google Drive	15
5 What interaction should there be with the cloud service?	16
5.1 What do we want from a cloud service?	16
5.1.1 Print to Cloud storage	16
5.1.2 Print to PDF	16
5.1.3 Print to PDF and send / share	17

5.1.4	Print to another printer	17
5.2	What are common technologies for cloud services?	19
5.2.1	Dropbox	19
5.2.2	Google Drive	19
5.2.3	Google Cloud Print	20
5.2.4	Box.net	20
5.2.5	Windows Live SkyDrive	21
5.2.6	Conclusion	21
5.3	What information do the cloud services require?	22
5.3.1	What information is commonly needed by the major cloud services?	22
5.3.2	How can we request the needed information from the user or the system?	23
5.3.3	What customer workflows do already exist?	24
5.4	Which scenarios are achievable?	24
5.4.1	Print to Cloud Storage	24
5.4.2	Print to PDF	24
5.4.3	Print to PDF and send / share	24
5.4.4	Print to another printer	25
6	Where do we integrate the cloud?	26
6.1	The Filter Pipeline	26
6.2	The Desktop PrinterExtension	26
6.3	Metro Device App	27
6.4	Combination Approach	27
7	What do competitors offer?	28
7.1	HP ePrint	28
7.2	Xerox	28
7.3	Brother	29
7.4	Konica Minolta	29
7.5	Canon	30
7.6	Kyocera Document Solutions	30
7.7	EFI	30
7.8	Epson	30
7.8.1	Google Cloud Print	31
7.8.2	Windows 8	31
7.9	First steps and stable concepts	31
8	Can creating our own cloud service help in integrating more services?	32
8.1	The normal way	32
8.2	Examples from the real world	32
8.3	Hosting the service	32
8.3.1	In the local network	33

8.3.2	A real cloud service	33
8.3.3	Hybrid options	33
8.4	The benefits win	34
9	How can we build our own cloud service?	35
9.1	Cloud Deployment options	35
9.1.1	Amazon Elastic Compute Cloud (EC2)	35
9.1.2	IBM SmartCloud Enterprise	35
9.1.3	Joyent	36
9.1.4	CloudFoundry	36
9.1.5	Google App Engine	36
9.1.6	Microsoft Windows Azure	37
9.1.7	Force	37
9.2	Accessibility of third party services	38
9.3	Availability towards clients	38
9.4	Platform dependency	38
9.5	Scalability	39
9.6	Advices and Choices	39
10	What infrastructure should be used?	41
10.1	The active portal	41
10.2	The passive relay	41
10.3	The open gate	42
10.4	The solution	42
11	Glossary	43
12	Sources	45



Canon
CANON GROUP

Foreword

This document describes the research for cloud integration in a V4 printer driver for Windows 8. The research is focussed on the cloud; its capabilities, existing services, technologies, workflows and platforms are the main subjects of investigation.

This research is a major part of my graduation on V4 Printer Drivers in Windows 8 and cloud integration. I am graduating from the study "ICT & Technology", which I am attending at Fontys University of Applied Sciences. I am fulfilling my graduation at Océ Technologies B.V. in Venlo, The Netherlands. At the time of writing this document, I have a basic V4 printer driver working in the Consumer Preview and Release Preview of Windows 8. I also have Metro applications and desktop printer dialog extensions that can set a limited number of settings to the printer. The printer driver can output to almost every Océ printer available.

The world of the cloud is already huge, and still expanding. I highly recommend checking the sources that I have used when you read this document, because it is very likely that things have changed. This research has been just a exploration of what is out there, and I have just taken the items, resources and technologies that could be useful for integration into a printer driver.

My vision about has changed during this investigation, and I now have a better view of the available technologies and possibilities of the cloud, but also the limitations and problems that you might encounter while developing for the cloud.

I hope that this document helps you build your own vision on the cloud and get a better perspective on how to us the cloud in a printer driver. I have enjoyed researching these subjects and I hope you will enjoy reading my questions, problems, and above all, answers and solutions to that I found.

1 Research Objective

The main question of this research is:

“How can one or more cloud services be integrated with a V4 printer driver to deliver an enhanced experience in Windows 8 printing?”

The objective of this research is to investigate how Cloud integration in a V4 printer driver for Windows 8 could enhance the printing experience for the user. We will be using the latest techniques in Windows 8 and the new V4 Printer Driver Model to accomplish a rich user experience. To go even further, we want to integrate cloud services to lift the user experience to the next level.

We already have a basic V4 printer driver in which we want to integrate cloud. The driver can print on pretty much every Océ printer, because of the limited subset of functionality. Because I already have experience writing printer drivers and extension software, I will not investigate the integration itself. For more information, I refer to my other documents about developing printer drivers and extension.

1.1 Supporting Questions

In order to answer the main question, we will compose a set of supporting questions. By answering those questions, we will be able to build a concept that answers the main question.

- What technologies can be used?
 - Which cloud technologies are suitable for our application?
 - Which technologies can be combined?
 - What are the advantages and disadvantages for each technology?
- What existing Cloud Services can be used?
 - Which technologies does the service use?
 - What about security and privacy?
 - How can we implement / integrate the service into the driver?
- What interaction should there be with the cloud service?
 - What information do the cloud services require?
 - How can we request the needed information from the user or the system?
 - What customer workflows do already exist?
- Where do we integrate the cloud in the printer driver?
- What do competitors offer?
- Can creating our own cloud services help in integrating more services?
- How can we build our own cloud service?
- What infrastructure should be used?
 - What infrastructures can be used in the Océ network environment?
- What research has already been done by Océ employees, and could we use that knowledge?

We will handle these questions in a better order, as seen in the table of content.



Canon
CANON GROUP

2 What research has already been done?

The first step that I have taken in my research, is to collect as much information as a possibly can about the research that has already been done within Océ, to prevent double work. I've spoken a couple of people, and gained access to the report they have written on their own research.

I won't describe the results of my search for information in detail. The information that I have used is embedded into my report. There will be no referencing to Océ sources for confidentiality reasons.

3 What technologies can be used?

3.1 Data Transfer Protocols

To really start with our research, we start at the bottom, with technologies. It is hard to precisely define the term “cloud”, but one thing we know for sure: It has to do something with the internet. That is the most important part for a cloud concept; possible communication with a lot of other computers, and not only within your own local network.

3.1.1 HTTP

The most important technology in the internet is HTTP, the Hyper Text Transfer Protocol. The technology uses TCP as transport layer, and every time you enter a website in your browser, the protocol is used to send data to a server, retrieve the website, and do so with little overhead. The protocol and its default port (80) is used so much that almost every firewall has the outgoing port open by default.

3.1.2 HTTPS

There also is a secure version of the protocol, HTTPS, which uses SSL to encrypt the datastream transported by the protocol. Because HTTPS is more secure than HTTP, it is also open in most firewalls. Because of the higher overhead on the protocol, it is only used when necessary. Both HTTP and HTTPS are very good technologies to use in cloud concepts, because of their existing footprint in the internet world.

3.1.3 XMPP

Based on XML natively using TCP as data carrier, the Extensible Messaging and Presence Protocol might also be a contender to be used as a cloud protocol. The open source protocol began its life under another name and purpose: Jabber was a near real-time instant messaging protocol, suitable for chat clients. Now, it has already found its home in the device communication protocol family and for the future maybe more. It is still being used in chat clients like Google Talk and Facebook Messenger, but because of the real-time aspects its use is expanded to a lot more. Google already uses it for notifications and job control to the printer in their Cloud Print solution. XMPP can also be transferred over HTTP. A disadvantage of this is that it usually runs on a webserver that also serves websites, so the port on which to access the XMPP messaging is different from the default HTTP port, and might therefore be blocked by a (corporate) firewall. An opposing advantage is that the XMPP clients do not have a server socket and therefore can't receive incoming connections. XMPP tries to use HTTP binding, and when that fails, it uses simple HTTP polling to communicate with the server.



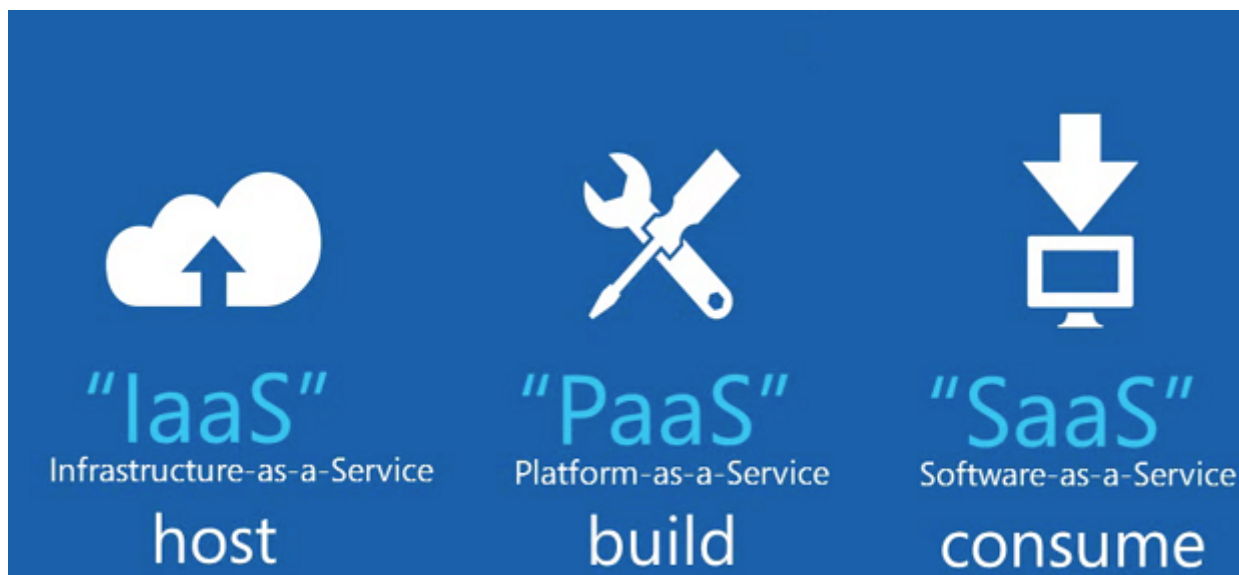


3.1.4 Data transfer: Which protocols should be used?

HTTP and HTTPS are without doubt safe to use in the cloud. In fact, they kind of define the cloud and the internet how they are now. Any technology using these protocols is relatively easy to implement, compared to using other protocols. XMPP is such a technology. XMPP is a very stable standard at the moment, but some considerations should be made involving the security of the corporate network. It would be wise to either use the default HTTP port, but you can also change the port and make adaptations in the corporate firewall, if needed. The first option is probably the best for most scenarios.

3.2 Cloud Flavours

Apart from protocols, there are also other technologies that are important for cloud. As a matter of fact, HTTP is the only protocol that is used in virtually every cloud solution somehow. So let's talk about the cloud itself. There are three different flavours of cloud: SaaS, PaaS and IaaS.



3.2.1 SaaS – Software as a Service

The first one, SaaS, means Software as a Service. This means that a cloud service provider sells a software service and maintains that software for its customers. Users can access the application software from cloud clients. The advantage of this concept is that the application is not installed on a local machine and so there is no local data that has to be stored securely. Instead everything is installed in the cloud, and when the local computer crashes, there is no data loss, the computer reboots and the user continues where he left. If you do that for every application in a laptop, you get a so called thin client. This means that you have a computer with just enough storage for a operating system, and a decent internet / network connection. The user can use the system as he is used to, but he won't be saving any data on the local system; everything happens in the cloud.

Examples of SaaS are big web-based email providers, like Gmail, Yahoo mail, and Hotmail. There are also storage services like Dropbox, Google Drive and Apple iCloud.

3.2.2 PaaS – Platform as a Service

With PaaS you buy more than just a managed application in the cloud. The P stands for Platform, and that is exactly what the cloud service provider is managing for you. Most cloud service providers have their own PaaS system, since there is no real standard for it yet. The use of PaaS is that users of the service can write their own cloud applications and host them on the platform. The platform is managed by the cloud service provider, so the user does not have to worry about that. A PaaS system package usually consists of a programming language execution environment, database system and / or some

other form of data storage capabilities and a web server. However, because the platform is managed for the user, the user is bound to use the programming language and APIs of the providers' choice. For the provider, making these restrictions is necessary to properly manage their systems, like in every big network environment.

Two big examples of PaaS are Google's App Engine and Microsoft's Windows Azure. With App Engine, you're limited to Java, Python and Go (experimental). Windows Azure is a lot more compatible; you can program in .Net (including C#, C++, and Visual Basic), Java, JavaScript, PHP, Python. Microsoft even claims that Azure will run nearly everything.

3.2.3 IaaS – Infrastructure as a Service

If you still want more than just a platform to program on, there is one more alternative: IaaS, Infrastructure as a Service. With this cloud service you rent complete virtual machines, servers and storage space, and you can do pretty much everything with it. You can install your own operating systems, or create a virtual machine image and run that parallel on a dozen of virtual machines. Even load-balancers and complete networks can be rented. With this variant, the sky is the limit. The cloud service provider makes sure that your machines keep running when they should, and your data is safely stored in multiple data centres, and is always reachable for the user.

Probably the most popular example of IaaS is Amazon EC2 (Elastic Compute Cloud). Océ has a virtual machine hosted by Amazon, for test purposes.

3.2.4 SaaS, PaaS or IaaS: Which one to use?

For our small project of integrating cloud into our printer driver, it would be best to keep it as simple as possible. The less we have to manage, the better, so the best solution for us would be to integrate one or more SaaS services. However, letting one or more custom cloud services through the firewall of a corporate network like Océ's is a possible security threat. One solution is to create a proxy within the corporate network that serves as a gateway to the cloud services. An other alternative is to write our own cloud service, host that on a PaaS system, and integrate all SaaS there, and let our own service pass through the firewall on a secure connection. There are various other options, including a few hybrid solutions between the options mentioned earlier. So PaaS is not out of scope yet. IaaS however is. There simply is no need for a complete infrastructure in the cloud, we just want to use software from the cloud, and maybe host our own, but that is it, and no self-managed virtual machines will be needed to accomplish that.

Our primary focus will be on the SaaS services hosted by other companies, or at least existing systems within Océ. When there is time left, PaaS might come into the picture.

4 What existing Cloud Services can be used?

A lot of this question has already been researched by Christian Luijten from Océ. Although the report is just a year old, the cloud development has not stopped since then, so I will add and update the information where needed.

4.1 A list of cloud services

Name	Service Type
Dropbox	Storage
Box.net	Storage
Cortado	Storage / Office
Windows Live SkyDrive	Storage
Google Drive (previously Google Docs)	Storage / Office
AutoCAD WS	Storage / Office
Google Cloud Print	Print
HP ePrint	Print
PrinterShare	Print
PrinterOn	Print
PrintPOD	Print
Amazon Cloud Drive	Storage
Mimeo Cloud Print	Print

The list you see above is just a small list with cloud services. There are a lot more out there, but the services mentioned above are relevant to this research, and important / big enough to bother to integrate. If you take a careful look at the services Christian Luijten mentions in his report (only for internal use by Océ), you will notice that I have let some services out of my list. Microsoft Sharepoint is mentioned in Christian's report because of the PaaS capabilities. The services shown above are all SaaS services, and therefore they are not in and not relevant for us.

4.2 Update on Dropbox

The only thing I have to update to Christian's report. The first update is just minor: Dropbox claims to have more than 50 million users nowadays (may 2012) and they save 500 million files a day. It is truly astonishing that the company doubled the number of accounts in about a year.



4.3 Google Drive integrates Docs

The second update is a lot bigger. Google Docs does not exist any more as it did before. Google launched its own implementation of a file storage service like Dropbox, called Google Drive. They integrated Docs into Drive, so it still exists, but the when you go to the original home of Google Docs, you will be forwarded to the new location of Google Drive. They changed the user interface, to comply with the rest of their new style services like Gmail and iGoogle.



4.3.1 Comparison with Dropbox

Google Drive could be a serious competitor to Dropbox, especially for businesses. Google offers bigger amounts of storage space for free, and buying more space is cheaper than with Dropbox. Users that have Dropbox and are facing space problems with their free account, will probably switch to Drive if they know about it. Dropbox has the advantage that the user is already accustomed to Dropbox, and the user won't bother to move that fast. Another advantage is that they have a client for every major operating system, and a good web interface for the users that can't or don't want to use a regular client. Google of course offers seamless integration with their Android mobile operating system, and Chrome OS, their netbook OS. They also have a PC and Mac client, and the iPad and iPhone version are coming soon. The question is if Drive is really as good, and will become as popular as Dropbox. The integration with their other services like Google Docs, and Google Cloud Print is a plus, and because of the enormous amount of existing Google accounts, the registration is hardly noticeable. But if the same happens to Drive as what happened to Google+ (Google's version of Facebook), it will have a lot of accounts, but no usage of the service.

So Dropbox and Google Drive are two big players to monitor and consider for integration into our concept. Drive has the advantage of the integration with Google Cloud Print, which also is a good service to integrate.

4.4 Amazon Cloud Drive

Something that Christian did not mention in his report is Amazon's Cloud Drive storage service. This is a simple storage service, no way near as extensive as Dropbox, Google Drive, or any other. They do have a desktop application for Windows and Mac but there are no mobile clients (yet), so they will have to work with the online web interface. It was of course a matter of time until Amazon would come with its own cloud service for individual use. They had the capacity long before, because they are hosting Dropbox on their servers, along with many other big company's services. They offer 5GB of storage for free (per account), and if you want more, there are several packages you can buy up to 1TB. The price for all packages is US\$1 per GB per year.



4.5 Mimeo Connect Cloud Print

Another unmentioned service is the Mimeo Cloud Print service. This however is something entirely different from other print services. Mimeo is a print service provider, like an online print kiosk. They print all kinds of things for individual and business clients, on demand, and send it via postal services. They provide several APIs for different kinds of printing media, and different customer groups. Is this interesting for us? No, because we want to keep as many of our clients' print job on our machines, and not outsource them to another company. Apart from the APIs, they also have a Windows Print driver, so you can set up your print on your PC, and print it directly to Mimeo. That driver will probably make use of the same APIs, so when interested, it might be smart to look into that first.



4.6 Compatibility between services

So I already mentioned that Google Drive and Google Cloud Print are integrated, but can we combine more of these services to work together? Well if we address every service separately, there should be no problem, but we are looking for built-in compatibility or integration within the services themselves. A common API or protocol can be helpful as well.

4.6.1 Google Drive and Cloud Print

For clarification, I shall describe what the integration between Drive and Cloud Print involves. There is bi-directional integration in these services. Google Cloud Print has a default printer installed that prints PDF format to your Google Drive storage. That printer is available from the Cloud Print API, so we can access both Cloud Print and Drive from one API, without making special modifications in the code. You actually integrate both services, whether you want it or not.

Drive also has a Print button in the web interface, as you would expect to have inherited from Google Docs. However, the Drive or Docs API show no sign of a print command or anything like it. The Drive and Docs APIs are about managing the files inside the storage, and the Docs API adds features like OCR and some other document specific features. That means that the integration is not fully bi-directional.

4.6.2 Box.net and Google Docs

Box.net claims to have integration with Google Docs. This integration is unidirectional because Google Docs does not have integration with Box.net. Unfortunately I had to be a developer for Box.net to see the actual API (that means creating an account and requesting an API key). Luckily there are articles and publications on Box.net to see how it works and what it can do. They have made a pretty deep integration with Docs; The Box.net user can now create new Google Docs and spreadsheets and edit existing ones on the Box.net storage without even leaving Box.net. All the files that are created and edited are stored on the user's storage account. Considering that they released this somewhere in June 2011, they then already made what Google Drive is supposed to be. Maybe Google copied the concept, and made a full integration, instead of a partly integration as Box.net did.



To summarize things: Box.net has a pretty good integration with Google Docs, but they probably won't touch each others storage, and it's not clear whether we can access anything from Google Docs from the Box.net APIs. For integration purpose, we can probably not profit from this feature in Box.net, but we don't have to. It might be enough for the user to access the Box.net files, because the rest will happen online.



4.6.3 HP ePrint and Google Cloud Print

HP ePrint is HP's solution to the question that we are also facing: Do we integrate separate third party services into our application or create our own service and integrate the other services into that? HP obviously chose the latter. Because HP already had their ePrint cloud solution in place when Google released the Cloud Print service, HP was able to integrate Cloud Print into their solution very fast. They also integrated the Apple AirPrint protocol, and with that they have made their printers available to both Android and iOS, the two biggest mobile operating systems on the market now. The integration is bidirectional, in the sense that Google can add the HP ePrint capable printers to their cloud printers. Integrating the HP ePrint service into our printer driver would be a bad move for us though, and probably one that is impossible to do, because the ePrint API is not available for public. The only ePrint API that is available is the app API for developing printer apps, but that has nothing to do with the cloud service. So we can scrap that of our list of possibilities.

4.6.4 HP ePrint and Mimeo Cloud Print

At first glance, it seems like a foolish move for a printing company to integrate a third party printing company services into your own printers, but there is a catch. Mimeo is a HP Gold Partner, which probably means that Mimeo uses HP printers and printer supplies for their jobs. That means that HP does not lose clients or income if they print via Mimeo, but only profit more from it. Printing a poster, for example, is not possible with a home printer like HP's ePrint capable printers, and with integrating Mimeo, they make it easier for the HP user to do so via Mimeo. This is a smart move for HP, but not one that we can apply as well.

4.6.5 AutoCAD WS and Google Drive

Just new of the shelf is the AutoCAD WS integration for Google Drive. This allows AutoCAD WS users to save their creations on their personal Google Drive storage, but also lets Google Drive users open the DWG and DXF (AutoCAD file format extensions) files that are in their storage with the AutoCAD WS app. This is a nice integration and adds value to the user workflow if we decide to integrate Google Drive into our driver. As stated before, it is not possible to access any applications from the Google Drive API, because it is a file access only API. The integration is not complete yet. At the moment the user can only view drawings from Google Drive. There is no save or export to Google yet, but that will come soon. There is nothing to blame, because Google Drive is pretty new and the API has just been released not even a month ago.

5 What interaction should there be with the cloud service?

To integrate a cloud service into an application, you need to know what you want to do with a potential service, you need to do what a selection of services can do and what information they need, and you need to find a match between your needs and the offering services. In this section we will pick up the first part, and give a general view on cloud service information needs for the second part of the question.

5.1 What do we want from a cloud service?

Because we want to integrate a cloud service into a printer driver, we already have a pretty specific scenario in which we use the application. The printer driver application is only seen when we want to print a document, or in a special case for Windows 8, select the app from the start menu. Since we are developing for the printer driver and not specifically for a cloud service, we leave the start menu entry point of our app aside for now, and go for the printing scenarios.

So assuming that the user wants to print a document and our interface pops up, what would the user want to do with the cloud? We have thought of a couple of scenarios which we will list here.

5.1.1 Print to Cloud storage

One of the most obvious scenarios is instead of printing to paper, printing to cloud storage. Most drivers already have a option to print to file on a local disc. Printing to cloud enables the user to open it again on his mobile devices and share it with other people. Other than that, cloud storage is in most cases safer than your local disc drive, because when it is in the cloud, your file will be stored multiple times on multiple locations.

The downside of cloud storage is that storage providers (especially the ones that provide you with free storage) always have some kind of access to your files. That makes it very unattractive for businesses to place any documents in the cloud, because they won't risk that any sensitive and confidential material will fall into the hands of a third party without their explicit approval. A lot of users don't know what their cloud providers do with their files, and with a bit of awareness, this can be solved for the most part.

The most desired functionality for this scenario is access to the private cloud storage service that the user might already have, or can easily subscribe for. The contenders being integrated are providers like Dropbox, Google Drive, being the biggest and most useful / compatible services on the market now.

5.1.2 Print to PDF

There are various PDF printer drivers already in the market, and there also are a couple of programs that can export to PDF, but an integrated PDF printing functionality in a normal printer driver is something new. What makes it extra special, is that the PDF conversion will not happen in the driver itself, but in the cloud. We can use a cloud service to generate the PDF file for us, and if desired, store it in the cloud.

The benefits are that we will not have to maintain the PDF conversion code in our driver. PDF is a very widely accepted document format and there is a PDF viewer for every major operating system with a graphical user interface.

The downside might be that again, it might not be used for any confidential documents, because the documents are handled by a third party service, and it is not clear what rights they have once you order them to process your document.

There are multiple providers for online PDF conversion, but the most versatile solution might be Google. By integrating Google Cloud Print and Google Drive, we have the complete functionality needed to accomplish the scenario described above, and more.

5.1.3 Print to PDF and send / share

The next step after printing to PDF is to send and share the document to other people or devices. This can be done by placing the document in a certain folder in a storage cloud service, e-mailing, or even share the file via social media networks.

The benefits from this functionality would be that the user can print and share from one screen. This means that instead of going to the steps of printing, downloading the file and sharing / sending it to people, the user will configure everything from start and the system handles the rest.

Again confidentiality is a big issue here. When posting to a social network, the user will have to be extra careful when selecting the security options, so the document does not go public, but even then it might not be safe from the social media network itself. E-mailing is safer on the other hand, and that in many cases is the way that this type of documents is being sent to other people.

In order for this functionality to work, we need the PDF printing functionality. This scenario has lower priority because it can have added value, but it is not vital for sharing. The user is perfectly capable of sharing the file with an extra step in third party applications, without the help of the printer driver.

5.1.4 Print to another printer

Within a corporate network like Océ, there are a lot of printers from Océ, for use by every company employee. They are installed by a lot of PCs and when a user wants to print to another printer, he will first have to install that printer.

This can be solved by creating a cloud of the printers and PCs that have the printer driver installed. If a PC does not have the printer installed that user wants to print on, the driver retrieves the printer specific information from a different PC or from the printer directly, so the user can configure the job for the desired printer. There are various possibilities to retrieve print settings, and places where to render the job to the correct PDL. Before implementing it has to be clear which implementation should be used and in what infrastructure it should be used.



The benefits of this concept are that the user can print on other printers more easily when a printer is already installed. When implementing this concept, the best solution would be to create a server that handles traffic and can provide temporary storage. This server can later on be used to integrate other cloud services and serve as a gateway to cloud service. This would solve the issue of letting the cloud services through the corporate firewall.

The concept does not have direct disadvantages. All disadvantages are implementation dependent, and many of these issues can be solved by design.

This concept is quite big, and implementation would consume quite a bit of time. Though current estimates show that this can be a successful implementation within the provided time, and we would have time left to integrate other services into the server. When designing the server, the integration of (a lot of) other cloud services should be taken into account, so one common API to the server can be used to address as many services as possible.

5.2 What are common technologies for cloud services?

To get by this information we have take a look at the API for every possible service that we want to integrate, to see what data they want.

We will treat the cloud services that have the most potential to be integrated. These services are Dropbox, Google Drive, Google Cloud Print, Box.net and Windows Live SkyDrive. I will handle every service separately, so we can draw conclusions from that.

5.2.1 Dropbox

Dropbox uses a number of open standards to enable you to access their APIs. They use OAuth for authenticating the user, SSL for encrypting the connection and UTF-8 as character encoding. The responses that you get on your request are primarily formatted in JSON. The dropbox APIs comply with the REST architecture, and they have made a few SDKs for a limited set of languages and platforms (Android, iOS, OSX, Python, Ruby, Java) and they all use those REST APIs. Luckily for us there also are some third party libraries for C#.NET.

What you would need to communicate with the Dropbox APIs is a “app key” and “secret”. This combination identifies your app to Dropbox. You have to request the app key from the Dropbox developer pages. On the first use of your app with a user account, the user will have to authorize your app to have access to the user’s Dropbox folders and files. This is done by sending the user to a webpage and let him / her accept the authorisation proposal. When the user returns from the webpage, you can try to retrieve the request token. If that fails, the user denied you access and you cannot access the API.

This all means that you don’t need to ask the user for credentials directly, but redirect him to a page that will handle that for you. This ensures a bit of extra security for your users. However, this could be difficult to accomplish this from a remote server, because the session will not stretch to a remote server or PC.

The APIs provide simple access to the files and metadata, without the need to send the request token as a parameter. The request token is embedded in the session and therefore there is no more need for further authorisation. With some HTTP POST, GET and PUT request you can read and write files to dropbox.

For the entire REST API, see DBOX03.

5.2.2 Google Drive

To make things easy, Google uses 1 authentication system for all their APIs and they use scopes to let the developer identify the APIs he wants to use, so Google can prompt that to the user. That system used OAuth v2 for authentication and authorisation with the user, and the developer only gets tokens to identify the app and its users to Google. When the authorisation and authentication is complete, your so-called app has a session with the Google API server and you don’t have to authorize anything again until the session expires. The API is RESTful, and the server usually replies on the requests in JSON. The



connection with the server (containing the HTTP session) is encrypted with SSL (so in fact HTTPS is used).

The Google Drive SDKs include a much larger set of supported languages. For example, they do have a C#.NET implementation for at least the authentication modules, and they try to be platform independent by only using pure language code, and nothing OS specific like user interfaces.

Once the session is ready, you can access the API, which consist of only 4 commands, through a RESTful interface. These commands provide basic file access to download and upload the files from and to Google Drive. That API can be found at sources GDRV02 and GDRV03.

5.2.3 Google Cloud Print

Same as with Google Drive, also Cloud Print is a service from Google that uses the same authentication and authorisation system, and when authorized you can access the Cloud Print API.

The Cloud Print API is far more extended than the Drive API. They have split the API for job submitting, and job receiving. Submitting is the interesting and only relevant part for us, because we are not yet looking at the receiving part, and within Océ there already is a Cloud Print proxy that we might use. There are only 5 commands, and especially the submit command has a lot more parameters compared to any Drive command. Of course this API is also RESTful.

The API can be found at source GCP02.

5.2.4 Box.net

While Box.net does not explicitly mention it on their developer website, they provide a REST API, encrypted with HTTPS and they seem to use a form of OAuth. I see a very similar authentication mechanism as with Google and Dropbox, but they don't speak of OAuth on their website, and what version that it might be. It could be that they have made their own implementation based on OAuth, or it accidentally resembles it very much. Anyway, the techniques and architecture is pretty much the same as what we have seen with the services above.

There is a major difference: The Box.net API does not use the session to store the authorisation, just the authentication of the user. With every request you do to the API, you have to provide your API key and authorisation token.

The Box.net API relies much more on the parameters in the requests. Their API is pretty complex. At first sight it does not have a lot of commands, but those are not commands. They show you the URLs of where to post those request, and every URL can be approached with a variety of HTTP methods and parameters. The documentation of their V2 API is still in beta phase, but I expect that there will not be any major changes, and that is a bit worrying because the documentation is weak in readability, and lacks a clear structure.

For as far as I can decipher, they do give you complete control over all features in Box.net, so they did not only include the file access, but also things like comments and discussions. They also make a distinction between folders and files in the request, something that previous services don't.

The Box.net API documentation can be found at source BOX03.

5.2.5 Windows Live SkyDrive

Also SkyDrive uses REST architecture in their APIs, and they use a similar system like Google does concerning service authorisation. Using scopes, your software has to identify which resources of the user account it wants to access. There is a difference: Microsoft requires a app key, or client ID as they call it, in the Windows Phone environment, but not for all the other environments like Metro or JavaScript. They don't seem to force you to use SSL because all their REST call examples use plain HTTP.

Microsoft provided libraries and SDKs for a lot of environments, and they all include a default mechanism for signing the user in, without letting the developer handle the user credentials. For example, in Metro XAML you can include the Microsoft Live Controls namespace, which includes complete user controls for logging in, and preformatted and styled buttons. Though all libraries should cover the same functionality, the use and naming of the libraries can vary a lot, which can make it very confusing for a developer.

At the bottom of one of the developer pages, it shows that signing in is also possible without help of the libraries. As developer, you will have to send the user to a certain webpage, with a couple of parameters like a client ID. Analyzing the URL that has to be used for authentication, we can see that also Microsoft uses OAuth v2 as authentication and authorisation system. That would explain the use of a session in the library samples.

The SkyDrive API documentation can be found at source MSSD03

5.2.6 Conclusion

Surprisingly enough, pretty much all selected cloud services use the same technologies. Dropbox, Google and Microsoft use OAuth, only the version that they use is different for Dropbox. Using another version of the protocol will only mean an adapted login method, but will not effect the session that is a result of that login. That means that if we implement the OAuth authentication well enough, it should work fine with the next step, REST API requests, regardless of the version of OAuth.

Another common technology between Dropbox, Google and Microsoft is the use of the REST software architecture. This is a complete architecture for both server and client side, and if we follow that architecture with enough abstraction, it should work fine for all the APIs from Dropbox, Google, Microsoft and even Box.net. This would make integrating Box.net just a bit more difficult, because we would have create a new login method and possibly adapt the session code.

Taking the limited time into account, I think it would be the best decision to start with the Google Cloud Print and Google Drive integration. The combined use of Google Cloud Print and Google Drive enables the most usability scenarios like PDF Printing, and sharing. Because of the common technologies, we

have to design a implementation that will take into account that maybe in future also Dropbox and the Microsoft APIs might be integrated. This should not be a problem, but it will take a bit of extra abstraction in the design.

5.3 What information do the cloud services require?

After taking a look what we would want from a cloud service, it is now time to see what cloud service would want from us, and how they intend to get by that information. This is to get a better picture of the possible security issues and it helps in developing a common API for all services.

5.3.1 What information is commonly needed by the major cloud services?

In section 5.2 we took a good look at the APIs from a selection of cloud services. Now we will analyze what data is needed by those services. We will narrow down the analysis to OAuth and some of the REST APIs.

OAuth is a commonly used authentication and authorisation mechanism for cloud services. There are a lot of benefits to OAuth. It is safe to use. It runs on the cloud service provider's servers, and that means that third party developers have no need to access the user credentials themselves. The OAuth system is scalable, and working with scopes enables the user to grant access to only certain parts of his account to applications. This all helps to sharpen the security and compatibility with third party software. As said, OAuth serves 2 purposes: Authentication and authorisation. Authentication involves handling the login of the user and checking the credentials with the account in the cloud service provider's database. Authorisation involves letting the user know that the app wants access to his account, and to what parts of the account. The user can either accept and grant access to the account information, or it can decline the request and block access to the app.

OAuth needs a few things from the developer side to initiate the authentication and authorisation sequence. First, as a developer, you need a app key, or client ID to identify your application to the cloud service provider's servers. In the case of Google you also need an "app secret", a sort of password that accompanies the app key. As result from a successful sequence, you get one or more tokens. One token is for direct access and can be embedded into the session so you don't have to worry about that any more. With Google, you can also get a token that allows you to refresh the access token when the session expires.

Once we have our access key, we can access the actual API. Most of the REST APIs that we have seen are very simple, but effective. The information that we can request from them is usually in JSON, or in the case that we want to download a file, binary.

For file storage services like Dropbox or Google Drive, we can access all information just by accessing the right URLs and parsing the JSON results properly and use the retrieve information to request more information about files and folders. You will only need outside information when you are looking for a certain file, because you will need the location about that file. When uploading a file, the location of the file on your local drive and the destination of the file in the cloud are needed. All other information can be



retrieved from the file metadata. When renaming, moving or creating empty files or folders, you will also need little bits of outside information, like new names and destinations.

For the Google Cloud Print service, we would of course need different data than with the file storage services. API is not as autonomous as the other APIs because it needs much more data from the client / user. Looking at the submit command in the API, there is just two parameters that have to be filled with data from a previous request, and that is the printer ID and the capabilities. The rest of the parameters have to be entered or generated by the user or app. That means that you have to provide the content type, which can be JPEG, PNG or PDF, the title, which can be anything because it just for internal use in Google Cloud Print, the content itself of course, and a optional tag to simplify tracking by the code.

We can safely conclude that only the authentication truly uses about the same set of information for all services. The needed information is too dependent on the API of the provider of the service and the type of service. File storage services require entirely different information than a print service. Also file storage service tend to be a lot more autonomous, meaning they need no or nearly none information from the user or the outside world in order to retrieve information.

5.3.2 How can we request the needed information from the user or the system?

As we have seen, for OAuth we just need to direct the user to a web page and retrieve an access token afterwards. We do need some keys, but they have to be hardcoded into the software, and they are not system or user dependent.

That means that the information we need depends on what services we want to integrate, and what the user wants to do with that service. We have a general picture of what the services can do and what information they can deliver and need.

For a file storage service, we want access to the hard drive or specified files. In order to upload files, we have to have access to the file on the hard disk, and after downloading a file, it has to be stored on the hard drive or some other storage medium. Maybe we also want the user to be able to rename, delete or move files, and we will need additional information to accomplish that. In that case we need additional ways to retrieve that information.

For a cloud print service, we need access to a file that is accepted as printer input content. We also need a print job title, and a printer ID, which can be retrieved from the cloud, but the user has to choose 1 of the available printers in the cloud. We have to be able to show those available printers to the user and let him make a choice. This all has to be considered when developing a design for our own cloud protocol / interface.

Considering that we are developing for Windows 8, with focus on Metro, we have to take into account that the Metro VM is heavily sandboxed, and it might not be easy to get hard disk access without using an extra user dialog. The rest of the data exchange with the user has to be done through the user interface, which should pose any problems.

5.3.3 What customer workflows do already exist?

Apart from delivering a quick and secure implementable solution for developers, OAuth also includes a rich customer workflow for our users. The workflow is quite simple for the user. It will handle the authentication if needed. If the user is already logged in, OAuth will initially skip the authentication part. If the user wants to change the account, he can do that by logging out and back in again. After authentication, the user is posed the authorisation question, to grant the requesting app access to (selected parts of) the user's account. After authorisation, the mechanism will redirect to a specified URL, on which you possibly could trigger your app to continue working, and try to retrieve the access token.

There are no further standardized customer workflows. The cloud service providers all have their different ways to let the user interact with their data, and that workflow even varies between mobile, desktop and web-based clients. For example: The Dropbox mobile client uses a forced dialog to upload a file, while on the Windows desktop client, you don't even need the UI to upload the file.

Because we are planning on using a service that uses OAuth for authentication, we are forced to comply to that workflow, but we don't have to do much in order to achieve that. In fact, we can't even control the UI part of OAuth, let alone the entire workflow. For the rest we will have to design our own workflow. That workflow has to be suitable for working on a tablet, because that is the main focus for Metro.

5.4 Which scenarios are achievable?

Now we have a lot more information about the cloud services that we might use. If we look back at the scenarios that we described in section 5.1, we can now start analyzing if the scenarios are achievable.

5.4.1 Print to Cloud Storage

This scenario would be achievable by integrating a file storage service like Dropbox or Google Drive. It probably is the simplest example possible with a printer driver and cloud services. The difficulty lies in having different output than XPS. The input stream for the printer driver already is XPS, and we cannot access the original file. We will either have to convert the XPS to a different format in the printer pipeline, or accept the XPS output as valid.

5.4.2 Print to PDF

This option seemed viable for a long time until I discovered that there are only 3 valid input formats for Google Cloud Print (GCP). The idea was that with implementing GCP, we would get access to the GCP PDF Printer, which of course outputs PDF to a folder in Google Drive. With implementing Google Drive alongside GCP, we would be able to retrieve that file from the cloud and act like we have converted the file ourselves. Sadly enough the input formats for GCP do not include XPS or PostScript, so we will have to come up with something else. If we could convert the XPS to either PNG or JPEG format, this functionality still has a purpose. When we need to convert the XPS to PDF ourselves in order to feed it to GCP, the functionality loses its advantage.

5.4.3 Print to PDF and send / share

If we take the PDF out, we get Print and send / share. This would only involve uploading the print output to a cloud storage service like Google Drive, Dropbox or Microsoft SkyDrive. After that, the user would be



Canon
CANON GROUP

able to use third party software from the cloud service provider to share the file with other cloud users. To send files, it would also be a possibility to e-mail the file directly, but that would go out of our scope to implement.

5.4.4 Print to another printer

The main goal of this scenario is to be able to print on all Océ printers within a (corporate) network, with just having the nearest printer installed. There are several ways to implement a solution like this, but the best solution would involve creating our own version of a cloud service, to create a cloud of all the printer drivers and / or printers within the network. There are various other implementation routes that involve implementing cloud storage services or writing our own server. This makes this scenario very achievable, it is just a matter of making a design that is expandable and will render a demo solution within the time boxed 4 weeks.

6 Where do we integrate the cloud?

Up until now we have only been looking at the cloud side of the integration. We do of course also have software, in this case a V4 printer driver, on which we should integrate the cloud. There are various software parts in a V4 printer driver, and they all have a different environment. We will discuss these parts and their environments and the possibilities and the effects of cloud integration.

6.1 The Filter Pipeline

The heart of the V4 printer driver is the XPSDrv filter pipeline. This is where the spooled print job (in this case always XPS) is converted to a Page Description Language (PDL). In order to do this correctly and to let the print come out of a printer correctly, we need to configure the job with a number of properties. These properties are combined into a PrintTicket, and this PrintTicket is created with default values, and has to be modified by the user before the filter pipeline is even started.

The filter pipeline is however not very suitable for cloud integration. Though it would be able to access the cloud resources, it is not possible to launch a UI from the filter pipeline any more, so we cannot ask the user anything, and everything should be entered up front. Since we would be configuring the cloud access up front, it would be much easier to directly access the resources than and already download the necessary data. After the print job has passed the pipeline, we might want to upload the result. This is something that might be done by either a pipeline filter, or a different software component with or without a UI. Anyway, the use of the filter pipeline for cloud integration remains limited.

The pipeline filters are written in native C++. V4 printer drivers only run in user mode, and cannot have a kernel part any more, because that is now replaced by Microsoft components.

6.2 The Desktop PrinterExtension

This is a customized piece of software that should replace the Microsoft default dialog for print preferences. This part of the driver is one of the only 2 software parts where we can have full access of the UI, and that makes it very suitable for cloud integration. The original goal for this application is to give the user the opportunity to make advanced adaptations to the print settings because that is not possible in the default Microsoft print dialog. It gives the developer the opportunity to create a UI that is consistent along different operating systems and platforms.

The desktop PrinterExtension is a Win32 application, built on the .NET 4 framework. Because of this, the application can have full access to the computer's network connection, and all other resources within the system. The system access combined with the UI capabilities results in having the perfect environment to deliver a customized user experience that integrates perfectly with the cloud. The only issue might be that the PrinterExtension runs and closes before the filter pipeline starts, and therefore it might be hard to handle the output from the printer once the filter pipeline is finished. This might make uploading files harder than expected, but with a background process, this might already be resolved.

6.3 Metro Device App

Then there is one more option to be eligible for cloud integration and that is the Metro Device App. This is the Metro style equivalent of the desktop PrinterExtension. Microsoft made 2 different implementations for this goal because Windows 8 has to run on tablets as well, but tablets will run a special version of Windows 8, Windows RT, which does not include the Win32 environment. That means that the Metro Device app is the only way to provide the advanced features on a tablet PC, while desktop PCs running Windows 8 will be able to run both applications.

Metro apps run on WinRT, and are built on the .NET 4.5 framework. The Metro environment is a sandbox with a rights management system. An app has to provide capabilities to gain access to for example system hardware, the internet or the local network. Because of the difference between the frameworks, the implementation of the desktop and metro PrinterExtension might differ a bit. Also, while the desktop PrinterExtension is to be designed for desktop usage, with a mouse and keyboard, the Metro Apps have to be touch compatible.

Despite of all the possible downsides and limitations, this still is the only way to provide a custom UI experience in Metro printing. This also makes it the only possibility to do cloud integration for Metro, with a UI for user input and feedback. It might also be harder to handle cloud IO after the print job is finished, because where the desktop PrinterExtension had a background process by default, the Metro app does not have that, and it will be harder to implement. The Metro app is not as good for cloud integration as the desktop PrinterExtension, but it is good enough.

6.4 Combination Approach

So some software components are better suited for cloud integration than others, but we will probably need a combination of multiple parts in order to make a successful integration between the printer driver and the cloud. This also means that we probably have to get creative to get certain parts of the integration work well with all system components. One thing is sure though: We have to treat the Metro app and the desktop PrinterExtension as 2 completely different things, and they can never depend or use each other. In the most favourable solution, these 2 applications have as much common code as possible.

7 What do competitors offer?

To come up with some ideas, and have a better sight of where we stand with our products, we also take a look at what our competitors are doing in the cloud and Windows 8. On the first sight, it looks like there is just one printer manufacturer that is really involved into either one. The rest of the printer companies are either busy with developing their own solutions and not showing anything to the outside world, or just are not interested enough to start developing (mainly for the cloud). The truth is that there are multiple companies involved in the cloud, but because there are so many ways to do something with the cloud, it might be difficult to track them down.

On the cloud part, there are a lot of printer producing companies that are involved. Pretty much all big printer companies have at least one cloud related product. For Windows 8, it is a completely different story. There is one company that already has gone public with an app, and the rest is either still researching and developing or waiting for the official release of Windows 8. Let's see what they have done already.

7.1 HP ePrint

HP ePrint is a self developed cloud platform that is like the solution that we are thinking of ourselves. They have made a cloud solution that integrates with their printer directly, and communicates to the cloud over just one protocol. In order to integrate other cloud services, they use their existing cloud service to communicate with other services. This is a very powerful solution, because this way HP has total control over the cloud service access software, and when integrating a new third party cloud service, they don't have to update the client software, just the software on their cloud servers.

Along with cloud services, they also implemented a application platform for their printers. That means that users can download apps onto their printers. These apps can then retrieve data from internet and print that, or generate puzzles for example and print those.

The third use of the ePrint is mobile printing. HP has created apps that enable mobile devices like tablets and smartphones to print on ePrint capable printers. This can either be done through existing cloud printing service like Google Cloud Print or Apple Air Print, or through the app with direct communication to the ePrint cloud service. They have a broad variety of access possibilities to their services, and everything is routed to the printers through just one protocol. That is the strength of their solution. This makes it very manageable, even for business networks. HP did not target businesses for their ePrint solution though, because all ePrint capable printers are All-in-One printers for home use. It is even possible to send a e-mail to the printer and the printer will print the contents of the e-mail or attachments if supported.

7.2 Xerox

Xerox has gone into a completely different cloud segment. Their main focus is on providing infrastructure to their customers. That means that a customer can rent entire servers, or just backup storage room. They also have a solution called Mobile Device Management, which manages mobile devices in a

corporate environment by technologies like over-the-air. That is about the complete package of cloud services that Xerox delivers.

However, they are still developing new services and have their own Cloud Consortium, a group of developers from several different companies, dedicated to building state-of-the-art cloud solutions and services for prepress, publishing, printing, print job management, automation, mobility and scanning of documents. By contacting this group with a specific request, I think they will also do tailored projects.

What is remarkable with Xerox, is that the cloud services have no really clear link to printing. The services are related to IaaS-like services and data storage hosting, which is pretty strange for a printer producer.

7.3 Brother

Brother has 2 cloud products which are ready for sale. One is a solution for online meetings and video conferences. It's called Brother OmniJoin, and it delivers a complete software solution for web conferences, like Microsoft Office Communicator and Skype. Since such tools already exist, Brother has to come up with something to compete with the popular alternatives. They have done so by integrating document and video sharing features. Brother also claims that the quality of the sound and video is higher than with other tools.

The other service is pointed towards scanning workflow management. This is more like the service we expect from a printer producer. They are working with third party vendors to implement a simple scan functions that enables the user to press the scan button, select a cloud location, and the document will be scanned towards the cloud. Brother is targeting the small and medium businesses separately from the enterprise level businesses. It is not clear whether Brother has separate solution implementations for those targets, but more information is coming soon on their website.

7.4 Konica Minolta

Also Konica Minolta has realised that the cloud is a hot topic nowadays and that they have to be in as soon as possible. They have done so by creating 5 applications for printing from the cloud, and most of them do so without intervention from a printer driver.

Konica Minolta has defined 2 types of cloud: public and private. The public cloud is a cloud service that allows multiple companies and persons on the same infrastructure. The data is protected by account access, but for companies this might not be enough insurance that their data is safe. That is where the private cloud comes in.

The private cloud is infrastructure managed by a third party, in one or more data centres somewhere in the world. This infrastructure is specifically created for businesses and tailored to their demands. They can be scaled up on demand and instead of paying for the infrastructure directly, the customer pays on a subscription base.

Konica Minolta has 2 applications for printing from public clouds and 3 for private clouds. They think that most of the companies will choose the private cloud, for the sake of data safety. Exact specifications on



their software and its capabilities are not published on their websites (yet), and you will have to contact them to get it. The only thing that we can discover from other Google hits, is that they also implemented Google Cloud Print, and that will most likely fall in the public cloud category.

7.5 Canon

Canon has picked up the cloud in 2 different ways. On the one hand, they have implemented Google Cloud Print into some of their Pixma desktop printers. Secondly, Canon has equipped a imageRUNNER Advance, with Cloud Portal. This office grade all-in-one printer can access files from the cloud and scan to the cloud, without the need of a computer. The user can just access the files from the cloud directly on the printer system.

7.6 Kyocera Document Solutions

Also Kyocera Document Solutions recently (April 13th, 2012) introduced their new cloud printing application Kyocera Cloud Connect. This application lets the user print from and scan to Evernote, a cloud service in which the user can store text, lists, images, sound files and a lot more. It is not like a file storage service, more like a document management service, but a bit more extended. Evernote is very cross platform with clients for Windows, Mac, Google Chrome, Android, iOS, and more. They also have a rich web interface, just in case that a native client app is not installed, not available, or blocked by a firewall. I suspect that this is merely a first step into the cloud, because there are a lot more interesting cloud services, and although Evernote has 25 million users, not much of them will have access to a Kyocera printer.

7.7 EFI

EFI developed something called PrintMe. PrintMe is a cloud printing solution that allows users to print anywhere, anytime on an internet enabled printer without cables or printer drivers. PrintMe is aimed at mobile professionals that need 24/7 access to print documents, and businesses that want to increase revenue streams and customer satisfaction.

PrintMe was launched 9 years ago, and that was well before the cloud era. This means that this might be one of the oldest cloud printing solutions around. The problem is that despite the age, it still is unpopular and not many people have heard of it. EFI claims they have 3000 installations worldwide, with tens of millions printed pages.

I was even more surprised when I found out that PrintMe is also available for Canon imageRUNNER Advance. That means that EFI is not targeting the PrintMe software just for their own machines, but also for other companies with printers outside their market segment (EFI builds big printers).

7.8 Epson

Epson is the only company that I found that has software in both cloud printing and Windows 8. They implemented Google Cloud Print, and made a Metro app for Windows 8, but these are 2 completely separated solutions. I would have expected that Google Cloud Print would have been integrated with the Metro app somehow, but that is not the case (yet).

7.8.1 Google Cloud Print

So they implemented Google Cloud Print as cloud solution. Not shockingly new, because other companies have also done so. The point is that Epson was one of the first to introduce it to their printers, because the press release stated in August 2011 that their new models from autumn 2011 are Google Cloud Print ready.

If you take a look at the product pages of Epson, it gives the impression that Epson Connect is not just a name for the compatibility with GCP, but rather is the name of the intermediate service that provides the connection to GCP. Details are missing, but if my analysis is correct, than they have built a cloud platform where they can integrate more services into one own protocol, like HP did.

7.8.2 Windows 8

Epson is the first printer manufacturer that created and published a Metro app for their printers. The app can show printer ink status when a correct driver is installed and the printer is connected. When that is not the case don't worry, because you can see a lot more than that. They can show you how to replace the ink cartridges in your printer and let you order new cartridges directly from Epson. They also added a showcase of other products that they build, like beamers and scanners. Everything is wrapped inside a very nice looking UI, which seems very tablet friendly. This certainly is a good example to other printer manufacturers, like Océ and Canon.

7.9 First steps and stable concepts

What we have seen from the competitors is that some companies have chosen the same approach when it comes to cloud solutions. Especially Epson and HP have solutions that very much resemble the wild ideas that we have. That means that our first steps are going into the right direction and with realizing such concept correctly, you have a stable base which could be expanded very rapidly if needed.

We also see that some companies are very careful with developing cloud solutions. Canon has some very small scale solutions with just one printer model. This almost feels like they have taken a proof of concept solution, worked it out a bit, and put that to the market. It is not really a complete solution. That feeling is strengthened because it is just one printer with those capabilities, and not an entire line with the same software. It is like they are not confident that it is going to work, and therefore they start with one, and if that sells, than they will consider expanding the line.

We cannot really test the solutions by other companies, but the general idea of the concept is sufficient for our research. Our goal is to develop our own solution, and not implement an existing one.

8 Can creating our own cloud service help in integrating more services?

During a brainstorm, I had the idea of creating our own cloud service to help integrating other cloud services. The concept would greatly simplify the integration process, architecture and maintainability and safety of cloud integration. Now we are going to investigate whether this really would be a better solution, and what the possible downsides would be.

8.1 The normal way

First we will take a look at the normal integration process. We are integrating cloud services into a printer driver, and we have selected a number of cloud services that we want to integrate, say Microsoft SkyDrive, Dropbox and Google Drive. When integrating, we would implement all three cloud services into the driver, and because their APIs are accessible through the same techniques, this would result in a common code base, and specific implementation would be derived from that common base.

This all looks pretty nice but there are some risks involved. Regarding network safety, it is a risk to a corporate network to have 3 outgoing connections, possibly on 3 different ports. In our scenario these 3 outgoing ports would be accessed by all printer drivers that are using the cloud integration and that would pose a security issue and a potential point of interest for hackers or other cybercriminals. Another issue would be bugs. When we have a bug fix and release that to the public, it has to be distributed to a lot of users and computers that are using the driver. Especially when we have a really deep integration into the printer driver, it can be hard to update the driver.

8.2 Examples from the real world

In the previous chapter we have seen that our competitors already have cloud integration, and especially 2 of them (HP and Epson) have solved this problem in a very similar way that we are looking into now. They have created their own cloud service and communication protocol to integrate that with their printers and instead of integrating the third party cloud services into the printers, they integrate the cloud services into their own cloud service, and the connection to the third party servers is initiated from their own servers.

By doing this, they reduced the security problem, because their printers only need access to 1 domain, instead of several. They also reduced the problem when releasing new features or bug fixes. The client software will only have to be updated with major changes and bug fixes, but the biggest part of the cloud access code and new features can be implemented into their own service and since they are able to manage those servers themselves, they can roll out new features completely under their own control, without losing availability to the users.

8.3 Hosting the service

A downside to having a own cloud service is that it needs to be hosted somewhere. Coming up with a solution is no simple task. The service can be hosted on several locations, even at the same time. Finding the right solution, but first we want to explore the possibilities.

8.3.1 In the local network

Hosting in the local network delivers a couple of advantages and disadvantages. Having a local server is easier to manage for the IT department when in a corporate network. Having the server under their own control, the IT department can manage which cloud services they want to enable for their users and which they want to block. They might even implement their own service and integrate local fileserver storage.

For a home environment, this might be a bit of a hassle to install a complete server just for the couple of PCs in home to accomplish a bit of advanced printer sharing, and for most home users cloud access is not a problem because they control their own firewalls. Cloud service providers usually have some documentation about how to get their synchronisation application running and provide some form of troubleshooting options in their apps.

Having local installations of these servers does reintroduce the problem with updates and new features. Because the server is in hands of a third party, those users will probably need to update manually sometimes or a really tight update implementation should be brought in place, and still than needs to be rolled out from another location, but that could be a simple file or web server.

8.3.2 A real cloud service

Second option is to make a real cloud service and host them on our own servers. This would give us complete control over every aspect of the cloud service, down to the hardware and operating services they run on. Those servers need to be placed in a data centre, with backup management, and colocation needs to be managed also, because in the event that your data centre goes down, or gets hit by a disaster, you don't want to loose all your system capacity and moreover, your stored customer data.

For companies who already have such systems in place, this should not be a problem though, and this might be the preferred solution for them. For new companies however, this is a completely different story. The best solution for them might lie in the cloud already. Using a IaaS or PaaS cloud service provider will solve these issues altogether because they will manage your hardware, availability, backups and everything else you need. You only need to provide the software.

When we take a look at the advantages and disadvantages of the real cloud option, towards the direct customers, we see that their side has simplified greatly. Home users will barely notice that there is just one protocol that communicates with their printer / printer driver. For corporate environments, this is a solution that takes away the hassle of having a extra server under their management, but possibly introduces a extra open port. They can firewall this by letting only users connect to the outside instead of also let them back in, and they could restrict the domains accessible through the open port. This is all manageable and probably requires less installation configuration and time to maintain than a local server.

8.3.3 Hybrid options

Than there are hybrid options that combine the use of local servers with real cloud services. The local server could act as a portal towards the real cloud service, or implement a part of the third party cloud services directly to reduce network loads. Having a local server, as mentioned earlier, has the advantage



that the IT department can completely control what services they want to be available to the user, and they can apply specific firewalling and virus scans to both outgoing connections to the real cloud and connections to the clients.

The local server could also be just a relay, like a proxy, without any clever solutions. This could be done without any software from us, so IT management can do this, maybe just documentation describing how this needs to be done, or some third party software for creating a tunnel. The best solution would be pure DNS or proxy solution, without tunnelling, because that could cause a major security leak.

For home users, this also would be a good solution, because they don't need a local server but can connect directly to the real cloud server. It would not make any difference between the hybrid solution and the real cloud solution.

8.4 The benefits win

After looking at a number of scenario's and even real life examples, we can come to the conclusion that implementing our own cloud service as some sort of intermediate format will help in implementing third party cloud services into our printer driver. Whether the final solution has to be a pure cloud solution or a hybrid solution with or without a separate piece of software for corporate environments has yet to be determined.

The final cloud solution will most definitely be different from the proof of concept that I will be creating during this graduation. I will try to incorporate as much of the aspects as possible within the time limits, but that might involve leaving this ideal path of having our own cloud service and just accessing a third party cloud service directly.

9 How can we build our own cloud service?

We have discovered that having our own cloud service would greatly increase the possibilities and flexibility of our cloud integration. Now we are going to have a look at what technologies are suitable for building such a cloud service. We have a number of things that we have to take into account:

1. Accessibility of third party services
2. Availability towards clients
3. Platform dependency
4. Scalability

Since we don't have an extensive web server environment with automatic backups and colocation, we are looking to build a solution targeted towards a cloud deployment environment. Within Océ we already have an Amazon hosted virtual machine, that I could use, and this seems a suitable environment for deployment. But let's not jump to conclusions yet, because we have not seen the rest.

9.1 Cloud Deployment options

The first thing we have to do is to build an index of available IaaS and PaaS services to see what their capabilities and limitations are. In this first section, we only want to list the services, and not explore the details of every provider, because that will come later on.

9.1.1 Amazon Elastic Compute Cloud (EC2)

Amazon, the big webshop company, has invested in building their own data centres for cloud services, and they have become one of the biggest companies in the cloud at the moment. Probably the most popular service that runs on the Amazon servers is Dropbox. Though Dropbox primarily uses another Amazon service (S3), it does not change that EC2 is a better option for our cloud service. Especially since we don't directly intend to provide storage to our clients, but rather let them use their own storage through our portals.

EC2 is an IaaS service, which means that Amazon takes care of the hardware, backups, and availability and we still get a virtual machine with total access to the operating system. Amazon has several preconfigured Amazon Machine Images (AMI) that we can use, or we could create our own image with our applications on them, and launch that on the Amazon servers. We can select the network access and security options for our machine, instance type, the use of colocation or not, and a lot more.

9.1.2 IBM SmartCloud Enterprise

IBM, as one of the world's leading manufacturer of servers and mainframes, of course also has cloud infrastructure for rent (IaaS). They guarantee a uptime of 99,9%, which is pretty high, and important for big companies that rely on internet services for sales and income. With IBM you know your data is safely stored and distributed over multiple data centers. The cost of this tends to be higher than with Amazon, but you pay extra for the quality. With IBM you get a couple of (data) security features by default, that would cost you extra with Amazon.



Setting up a cloud with IBM SmartCloud Enterprise is as simple as with Amazon. Simply select an image type, configure the hardware and network connectivity and provision the server. Everything can be managed through an online portal without the intervention of an IBM employee.

9.1.3 Joyent

Joyent is a full blown cloud hosting provider, which serves several types of hosting. The product that provides the most options and freedom is SmartDataCenter, where you can build a virtual data center of your own. You can host multiple virtual machines in this environment, and manage their internal and external networking configuration. Just like having your own server park or room, but virtual, remote and backed up safely. When you design your solution on this product correctly, it should be easier to scale up your system and make it more secure than with other products from Joyent.

Another product of Joyent is SmartMachines. This is one step higher than having a data center. You have full control over your machines and their operating systems, but not over the underlying connection systems and you have less scalability and security possibilities.

The third product is called SmartOS. This is Joyent's own operating systems that also power the SmartDataCenter. According to Joyent, this product combines the best from hardware virtualisation and operation system virtualisation. It can deliver high quality storage, complete virtualisation and analytics.

Joyent's SmartDataCenter is a typical IaaS solution, and a rather extended version of it. SmartOS still balances on the separation line between IaaS and PaaS, and SmartMachines is in between those two products. With these three solutions, Joyent has a very solid cloud service platform as an IaaS business.

9.1.4 CloudFoundry

With CloudFoundry, we enter the world of PaaS. The greatest advantage of CloudFoundry is that developers can download software that simulates their real cloud environment. This means that developers can develop their app locally and run and test it within their companies on test systems and developer machines, and when ready, deploy it to the real servers at CloudFoundry. They claim that they are the first in the industry that could do that.

CloudFoundry supports a number of standard frameworks for different languages, so you are not bound to one framework and / or language. They also deliver the possibility to communicate with a number of third party applications like database engines (PostgreSQL, MySQL). Combine that with some architecture that makes your apps ready for the future and you have a decent solution for creating your own cloud service. CloudFoundry also has a open source community that can help you with implementing your application and integrating other frameworks into CloudFoundry, so if you get stuck, you can take a look there.

9.1.5 Google App Engine

The next provider is Google, with their App Engine. Being a product from Google, means that it is stable, powerful and easy to use. This PaaS system leaves no sign of the original operating system, and you are

bound to a limited number of languages (Python, Java and Go). In return to that, you get a number of powerful APIs from Google and the possibility to have a deep integration with other Google services.

Google does not only allow you to run apps, they can also host your website and store your data. Combined with Google Apps (which is e-mail, calendar and all other Google services for businesses and has nothing to do with the Google App Engine), you can host your entire company in the cloud for sharp prices, or even for free.

9.1.6 Microsoft Windows Azure

Microsoft also has come up with a solution for cloud service hosting: Windows Azure. When we analyze the name, we see Windows, which would indicate a relation to their operating system family. That would indicate that Windows Azure would be a PaaS system, but it also provides IaaS solutions.

Microsoft sent an Azure expert here at Océ, to give a presentation and demonstration on Windows Azure. She mainly talked about the PaaS side of the system and how to manage that. She showed us that management was really easy, and you could do everything yourself, from deploying your application to a couple of instances, and how to scale up your system. With Azure, you pay for what you use, and because the PaaS is so easy to use and fully automated, the cost of using Azure is pretty low. With the Azure PaaS system also lets you control where (in what countries) your application is available.

There are some limitations though. Up until now, Azure only has a relational database that the developers can use. They do support a reasonable number of languages like .NET (C#, C++), Node.JS, Java, PHP and Python. You can even configure custom environments with unsupported languages like Ruby.

The Azure IaaS system lets the consumer control everything from virtual machines, networking, data storage facilities and even integration with your local Active Directory to provide your users with one-time login through the entire system. Somewhat surprising is that Microsoft provides Linux as a default operating system alongside the Windows Server line to run on virtualized servers.

The IaaS solution is not as well promoted on the Microsoft website as the PaaS system, but it almost seems like they are directing the customer to use the PaaS environment, instead of the IaaS. I would have expected a hybrid solution too, so customers would be able to host their business in the IaaS system, and create apps for own use, that integrate with their business software.

9.1.7 Force

Force is a PaaS system from Salesforce. It focuses on enterprise level companies to extend their current systems to social networks and mobile platforms. They offer integration options for enterprise level management systems like SAP and cloud systems from Amazon (both EC2 and S3).

A big disadvantage is that the Force environment only supports one programming language for business logic (Apex, a Java-like language optimized for using databases), and one for user interfacing (VisualForce, obviously a self developed language). This could be a serious limitation for companies with legacy code, which is very likely for enterprise level companies.

9.2 Accessibility of third party services

The possibility to access third party services is a very important factor for us, to choose a suitable deployment environment for a cloud services. Without proper access we cannot integrate the services the way we want and than we would lose important functionality.

If we take a look at our list above, and match the requirements against the capabilities, we can safely conclude that all the providers that have provide an IaaS solution, are suitable. For all IaaS solutions, it is vital that they can have access to a full internet connection. With an IaaS solution, we can exactly control which machine and which part of our system has what access to the internet.

For a PaaS solution, the level of access to third party services might be limited by a number of factors. A factor could be the PaaS system itself. To protect the platform, a firewall might be in place that blocks any unauthorized access to the platform, even from within. This is very system and provider specific, and before starting development, this has to be clear. Another possibility is a limitation in the APIs of the platform. If the system does not support the communication resources of a programming language, framework or API, than we could get into trouble. This is could be hard to figure out, because the API's might be sealed for non-developers. It is even possible that only certain domains, ports or content types are blocked. Therefore it is vital that we know what limitations there are, if we decide to start implementing on a PaaS system.

9.3 Availability towards clients

Availability to clients is also vital to our cloud service, as for every cloud service. The reason why we are still looking into this point, is because apart from being online as much as possible, we need to know what happens if we run out of bandwidth, the data center goes down, or our storage space is full.

With an IaaS system, you have full control of your system resources, but it dependent on the capabilities of the provider's systems and your settings what happens when you are (almost) running out of them. Some systems allow on-demand growth, with or without (user configured) limit. Some providers even have their billing models adapted for that (pay as you go, pay for what you used). With other providers, you will have to monitor this yourself and make changes to your system in order to keep it running smoothly.

With PaaS systems, this is generally done for you, but there are a lot of business models for this, and every PaaS provider has multiple models for managing resources and billing it to the customers. With some systems it is even possible to automate the scaling in the app code, so if you are detecting a lot of traffic, you can spin up a extra instance, and / or report it to the administrator.

9.4 Platform dependency

In order to create a cloud service we have to take in account that we can have a lot of difference in platforms that we might encounter. From the point of the cloud service, the platform is stable, but the clients can all be different. This means that we cannot use any platform specific output and have to stick to either standardized web techniques like HTTP or use a widely supported open source solution. Of

course we can also choose to implement all our clients' connectivity code ourselves, but that requires a lot of work, and if the chances of our product becoming a success are diminishing pretty fast.

In this first project we are only focussing on a Windows 8 printer driver as primary cloud client. When this becomes a success, and development is expanded to a lot of other client platforms like Mac and maybe even mobile platforms, we do need a common API that is accessible for all the targets.

Fortunately this requirement does not limit the choice of the cloud platform. All cloud platform providers and systems have the capability to use pure web techniques. Though with IaaS systems, where we have to implement these ourselves, we do have to pay extra attention to this. With PaaS systems it is more likely that a standardized communication is part of the platform.

9.5 Scalability

An important term in the cloud business is scalability. A cloud platform provider that does not support or deliver fast scalability is doomed. Where websites still have static capacity for a month or even a year subscription, with cloud services it is normal that the capacity can scale up within 15 minutes.

With an IaaS system it is easy to add raw computing power to your system, but you have to design a system that can cope with dynamic scaling, otherwise you can add a lot of computing power, but it won't have any effect. Depending on the chosen provider, you can get APIs to implement dynamic scaling in the code itself. That would for example enable you to create a master program that monitors load and traffic, and controls scaling if the load goes outside certain boundaries. And with learning from history (daily pattern) you could spin up extra instances and create extra capacity before a peak arrives.

With PaaS systems, you don't need to worry about instances of virtual machines and complex designs to cope with scaling. A good system will force you to implement certain features to prepare your application for scaling by the hosting system. Instead of running multiple virtual machines as with IaaS, the PaaS system will run multiple instances of your program and distribute them over multiple virtual machines without you noticing anything. Also with PaaS systems it is even more likely that you get mechanisms to automate the scaling process.

Because we are not hosting data (or at least not permanent but just caching) we don't need to focus on data storage space. Computing power and especially bandwidth is our main focus for scalability and we should pay attention to this when making a decision about the platform we are going to use.

9.6 Advices and Choices

For our test project, we are going to use the Amazon EC2 server that Océ already has for test purposes. This is the solution with the most possibilities and least costs, and because it is already up and running, we will be able to use it faster and could start development right away.

This does not mean that when the final concept is ready, the solution above is the way to go. At that point, it has to be investigated what is the best solution and adapt the design to that hosting solution. Both IaaS



and PaaS have different advantages and disadvantages. IaaS comes with the greatest freedom to implement whatever you want, but you pay for that freedom as well.

IaaS solutions require more implementation because you only hire virtual machines, storage space and bandwidth. The complete software system, including the complex aspects of scaling and thread safety has to be implemented by the customer.

With a PaaS solution, it may vary what is already done for the customer. With closed systems like Force, it is likely that the implementation part of the customer is considerably lower than with a system like Azure. The more freedom you get, the more you have to implement, and finding the correct balance between is difficult and different for every cloud project.

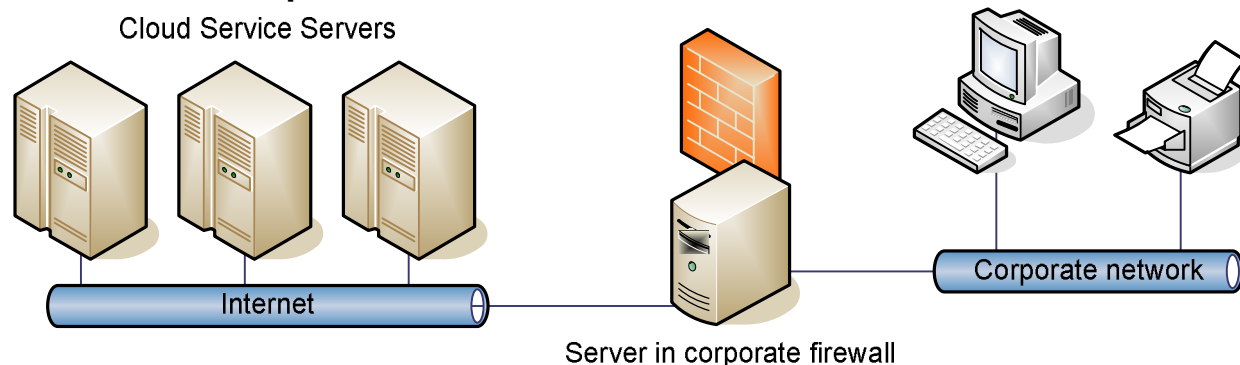
So it may not be much of an advice, but it really is dependent on the final concept of the cloud service, and the abilities that it needs to have. The concept that we have not is not even sure of implementation of our own cloud service, but for experimenting and finding the boundaries of the cloud, the IaaS solution of Amazon EC2 is a great opportunity and we will be able to create a great proof of concept with it.

10 What infrastructure should be used?

For a corporate network we need to take infrastructure very seriously. A corporate network can be seen as some sort of worst case scenario, and if we are able to deal with it, we can handle every other client as well.

There are a couple of scenarios to provide cloud access to every client within a corporate network. Some solutions require extra software for managed servers, and other solutions are just recommendations and instructions for IT management departments.

10.1 The active portal



In this scenario we are dealing with an extra piece of software that runs on a server in the corporate firewall environment. Instead of connecting to the real cloud service, the clients connect to the local server that takes care of the connections to the real cloud. This includes the connection to the own cloud service, but also integrated cloud services, to relieve load of our own cloud service connection.

The advantages of this solution is that IT management can control what services they want to let through to their users, load is reduced from our own cloud service, and the speed of the solution is higher. The disadvantages are that a extra piece of software is needed, we need to take in account that on the client side, certain cloud service might be blocked, and we lose a bit of purpose of our own cloud service.

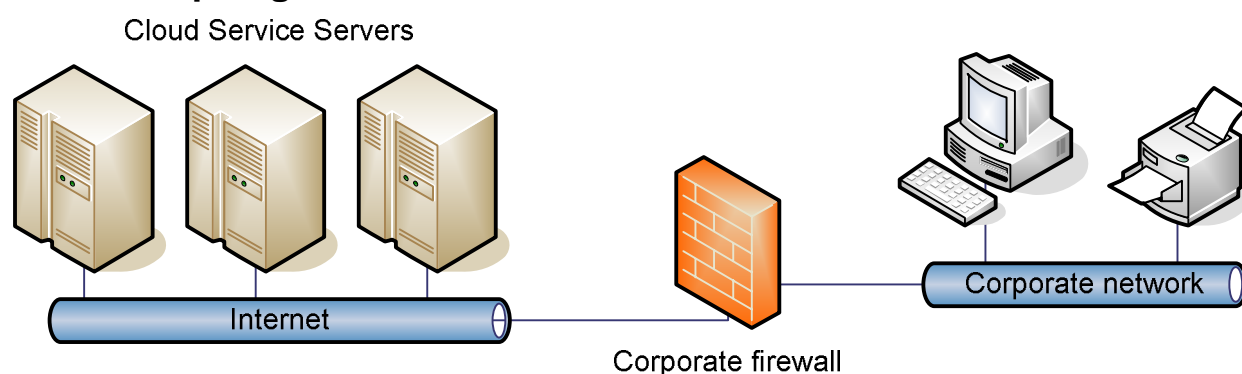
10.2 The passive relay

This scenario uses the same infrastructure setup as with the active portal, except the software running on the systems is different. The server in the corporate firewall serves only as a relay server. This can be achieved by using a proxy structure, or a tunnel to our own cloud service server combined with simple DNS rules in the corporate network. This means that we do not need extra software for the local server.

Advantages of this solution are that the server is very manageable and can be monitored and firewalled strictly by IT management, and there is no need for extra software development by us. Also in the event of a crash of the server, there is no loss of data, just configure another server and put it back online. Disadvantages are that the load on our own cloud service is higher than with the active portal, and IT

management cannot control the content of the cloud service stream, and the server needs to be monitored and configured. The load on the cloud service is just a relative issue though, because otherwise we would have had the load on the local server. Only with very busy networks and printing traffic, we would have a problem, but for now that is not the case.

10.3 The open gate



The third scenario is a lot simpler than the other scenarios. IT Management just has to open an outgoing port and make sure that the domain is not blocked, so the clients can connect directly to the own cloud service. An incoming port is not needed when the service provides a pure REST API.

Advantages of this approach are that it is pretty safe, virtually no maintenance for the IT department, and configuration for corporate users would be the same as with home users. As an added advantage the IT department can decide to block all unnecessary domains from the opened port. This would exclude the possibility of hackers and viruses using the opened port to sneak data in or out. Disadvantages are hardly there, except that the load on the cloud service is also higher than when having a local server, but that issue has been addressed already.

10.4 The solution

The most sensible solution seems to be the last option, where we ask IT management to open a port in the firewall for the entire network. This option can later always be expanded with an active local server if the traffic to the cloud service becomes too high, and we want to relieve that connection. This solution requires the least configuration and maintenance, and no extra software components and is therefore also the cheapest option. This does require REST technology to be implemented on the service, but that is a technology that we were going to use anyway.

11 Glossary

<u>Word / Acronym</u>	<u>Meaning / Explanation</u>
V4 Printer Driver	A version 4 printer driver. V4 or version 4 is the latest version of the printer driver framework / model from Microsoft. Designed and built for Windows 8.
GB	Gigabyte, unit for amount of data. Equal to 1.000.000.000 bytes
TB	Terabyte, unit for amount of data. Equal to 1.000.000.000.000 bytes
API	Application Programming Interface. A collection of definitions on base of which applications can communicate with other applications or application parts.
PDF	Portable Document Format. Developed by Adobe. Standard format for exchange of electronic documents.
PDL	Printer Description Language. Data format for document description, used to communicate to printers.
OAuth	Open protocol to allow secure API authorization.
URL	Uniform Resource Locator, special form of URI (Uniform Resource Identifier). Structured name that points to a piece of data. The name contains all needed information to access the data.
JSON	JavaScript Object Notation. Open text data format for human readable data exchange, derived from the JavaScript scripting language.
JPEG	Joint Photographic Experts Group. An image compression method.
PNG	Portable Network Graphics. Lossless image compression method.
Metro	Design language, created by Microsoft. First real focus in Windows 8 UI.
VM	Virtual Machine. Virtualized computer for running an operating system within another operating system.
XPS	XML Paper Specification. A PDL originally developed by Microsoft
PostScript	A PDL developed by Adobe.
XPSPDrv	Pronounced: "XPS Drive". Printer Driver architecture, using XPS as main document format
PrintTicket	Set of (user configured) print settings, specific for one print job.
UI	User Interface. A way of letting the user interact with a electronic system.
C++	Pronounced: "C Plus Plus". One of the most versatile and most used programming languages.
Win32	Windows 32bit API. The Windows API to access system resources, built for 32bit operating systems.
.NET	Pronounced "Dot Net". Software framework, developed by Microsoft. Extension to the Windows API.
WinRT	Windows Runtime. Actually the same as Windows API, but suitable for ARM and Intel core processors. Windows 8 uses this runtime for all Metro related software.
app	Short for application. Popular word and mostly used to refer to an application for a mobile operating system.
DNS	Domain Name System. Hierarchical distributed naming system for computers, services, or any resource connected to the internet or a private network.
Android	Google Android. Mobile operating system from Google. Mostly used for smart phones.
iOS	Apple iOS. Mobile operating system from Apple. Used for iPhone, iPad and iPod.
OSX	Apple Mac OSX. The 10th version of the Apple Mac operating system.
Python	High level programming language. Main characteristics are the dynamic type system and multi paradigm support.
Ruby	Programming language with strong resemblance to Python

author(s)

M. Wingbermühle

Java	Object Oriented programming language originally developed by Sun Microsystems. Mainly runs in a virtual environment called the Java Virtual Machine (JVM).
C#	Pronounced "C Sharp", Object Oriented programming language originally developed by Microsoft within the .NET initiative. One of the programming languages for the Common Language Infrastructure. Also needs a virtual machine environment to run in.
SDK	Software Development Kit. Package of software tools and libraries to enable software development for a certain platform, API or framework.
UTF-8	UCS Transformation Format 8bit. Variable width character encoding that can represent every Unicode character. Designed for backward compatibility with ASCII
SSL	Secure Socket Layer. Cryptographic protocols for communication security over the internet.
proxy	Short for Proxy Server. A computer or application that acts as an intermediary for client seeking resources from other servers.
JavaScript	Scripting language. Dynamic, weakly typed and multi paradigm. Frequently used in websites, though server side and desktop uses of JavaScript are also gaining popularity.
sandbox	A security mechanism to separate running programs. Often used to protect systems from untrusted code.
REST	REpresentative State Transfer. Standardized software architecture. Used for implementations of web and cloud services.

12 Sources

Source:	Google
Location:	https://developers.google.com/cloud-print/docs/sendJobs
Shorthand:	GCP01
Source:	Google
Location:	https://developers.google.com/cloud-print/docs/appInterfaces
Shorthand:	GCP02
Source:	IBM Zurich
Location:	http://www.zurich.ibm.com/~cca/talks/metis2011.pdf
Shorthand:	IBMZ01
Source:	Wikipedia
Location:	http://en.wikipedia.org/wiki/Extensible_Messaging_and_Presence_Protocol
Shorthand:	XMPP01
Source:	Wikipedia
Location:	http://en.wikipedia.org/wiki/Cloud_computing
Shorthand:	CC01
Source:	Dropbox
Location:	http://www.dropbox.com
Shorthand:	DBOX01
Source:	Dropbox
Location:	http://www.dropbox.com/static/docs/DropboxFactSheet.pdf
Shorthand:	DBOX02
Source:	Dropbox
Location:	https://www.dropbox.com/developers/reference/api
Shorthand:	DBOX03
Source:	Silverlight Hack
Location:	http://www.silverlighthack.com/post/2011/02/27/laaS-PaaS-and-SaaS-Terms-Explained-and-Defined.aspx
Shorthand:	SLVH01
Source:	Google Drive
Location:	http://drive.google.com/
Shorthand:	GDRV01



Canon
CANON GROUP

Source:	Google Drive
Location:	https://developers.google.com/drive/
Shorthand:	GDRV02
Source:	Google Drive
Location:	https://developers.google.com/drive/v1/reference/
Shorthand:	GDRV03
Source:	Box.net Blog
Location:	http://blog.box.com/2011/06/box-and-google-docs-accelerating-the-cloud-workforce/
Shorthand:	BOXB01
Source:	Google System Blog
Location:	http://googlesystem.blogspot.com/2011/06/boxnet-integrates-with-google-docs.html
Shorthand:	GSYS01
Source:	Box.net
Location:	https://www.box.com/
Shorthand:	BOX01
Source:	Box.net
Location:	https://www.box.com/platform/
Shorthand:	BOX02
Source:	Box.net
Location:	http://developers.box.com/docs/
Shorthand:	BOX03
Source:	Amazon Cloud Drive
Location:	https://www.amazon.com/gp/feature.html/ref=amb_link_362776742_2?ie=UTF8&nav_sdd=aps&docId=1000796931&pf_rd_m=ATVPDKIKX0DER&pf_rd_s=center-A5&pf_rd_r=0FQJ4EBD2JJ3A8AN0NGA&pf_rd_t=101&pf_rd_p=1364264882&pf_rd_i=507846
Shorthand:	ACD01
Source:	Mimeo
Location:	http://developer.mimeo.com/
Shorthand:	MIM01
Source:	AutoCAD WS
Location:	http://www.autocadws.com/blog/autocad-ws-launches-google-drive-integration/
Shorthand:	ACWS01



Canon
CANON GROUP

Source:	Wikipedia
Location:	http://en.wikipedia.org/wiki/Representational_state_transfer
Shorthand:	REST01
Source:	Microsoft SkyDrive
Location:	http://msdn.microsoft.com/en-us/library/live/hh826543.aspx
Shorthand:	MSSD01
Source:	Microsoft SkyDrive
Location:	http://msdn.microsoft.com/en-us/library/live/hh826540.aspx
Shorthand:	MSSD02
Source:	Microsoft SkyDrive
Location:	http://msdn.microsoft.com/en-us/library/live/hh826531.aspx
Shorthand:	MSSD03
Source:	Xerox
Location:	http://www.xerox.nl/digitaal-printen/cloud-services/nlnl.html
Shorthand:	XROX01
Source:	Xerox
Location:	https://acscloud.com/
Shorthand:	XROX02
Source:	Konica Minolta
Location:	http://www.konicaminolta.nl/konica-minolta-business-solutions-nederland-bv/producten-diensten/applicaties/cloudoplossingen.html
Shorthand:	KOMI01
Source:	Brother
Location:	http://www.brothercloud.com/
Shorthand:	BCLD01
Source:	Canon
Location:	http://www.usa.canon.com/cusa/office/products/software/document_distribution_and_management/cloud_portal_for_imagerunner_advance
Shorthand:	CNON01
Source:	Kyocera Document Solutions
Location:	http://www.kyoceradocumentsolutions.com/news/apr1213.html
Shorthand:	KYDC01



Canon
CANON GROUP

Source: EFI
Location: <http://w3.efi.com/en/Fiery/Products/EFI-PrintMe>
Shorthand: EFI01

Source: EFI
Location: <http://w3.efi.com/Fiery/Products/EFI-PrintMe/EFI-PrintMe-for-Canon>
Shorthand: EFI02

Source: HP
Location: <http://www8.hp.com/nl/nl/campaign/photosmart/eprint.html>
Shorthand: HP01

Source: Epson
Location: <http://www.epson.co.uk/Explore-Epson/1228074434969>
Shorthand: EPSN01

Source: Epson
Location: http://global.epson.com/newsroom/2011/news_20110831_3.html
Shorthand: EPSN02

Source: Amazon
Location: <http://aws.amazon.com/ec2/>
Shorthand: AEC201

Source: CloudFoundry
Location: <http://www.cloudfoundry.com/>
Shorthand: CLFO01

Source: Google
Location: <https://developers.google.com/appengine/>
Shorthand: GAE01

Source: Joyent
Location: <http://www.joyent.com/products/>
Shorthand: JOYE01

Source: IBM
Location: <http://www-935.ibm.com/services/us/en/cloud-enterprise/>
Shorthand: IBMC01



Source: SlideShare
Location: <http://www.slideshare.net/sriramk/windows-azure-cloud-service-development-best-practices>
Shorthand: SSWA01

Source: Microsoft
Location: <http://www.windowsazure.com/nl-nl/>
Shorthand: MSWA01

Source: CloudForge
Location: <http://www.cloudforge.com/why-cloudforge/-cloud-deployment>
Shorthand: CLFG01

Source: Force
Location: <http://www.force.com/>
Shorthand: FRCE01

Source: OAuth
Location: <http://oauth.net/>
Shorthand: OAUT01



Canon
CANON GROUP

Installing a Metro Device App

Installation, Deployment & Testing Instructions

author(s)

M. Wingbermühle



Table of content

1 Installation Components	3
2 Device Metadata	4
3 Metro Device App	5

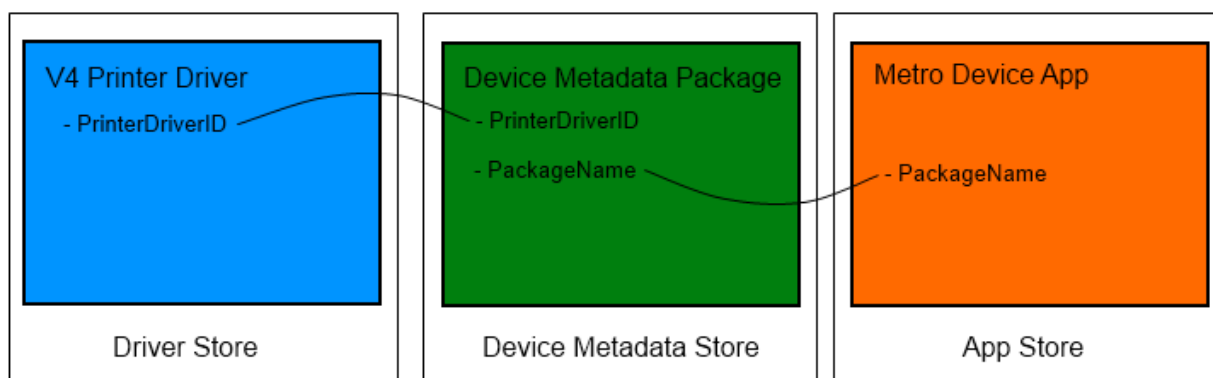
1 Installation Components

To install the Metro style device app for Windows 8, you need 3 software components. The component is the hardware driver, in our case that means the V4 printer driver. I am not going to explain how to install the driver in this document, because Microsoft has clear documentation on that, and using the wizard is pretty straightforward.

The second component is the Metro device app project. Yes, you will need the entire project, and Visual Studio 11 or 2012 as well to deploy it. This is because the Metro apps are actually meant to be delivered from the Windows App Store, but since this software is confidential and for testing purpose only, you will have to install it manually.

The third part is the Device Metadata file, and it connects the device app with the driver by providing Windows with the required data to associate the device app to the printer driver ID.

I have added a diagram showing how the software components are linked together:



The diagram shows which identifiers are used to link the components, and in where the separate software components are stored in Windows. The stores are actual folders on the Windows system drive, and are not to be confused with the distribution platforms, though they are related.



2 Device Metadata

The first step to install the device app, is to provide Windows with the association data in the form of the Device Metadata file. This is a file that has a GUID as name, with the .devicemetadata-ms file extension. You can create and edit these files yourself if you have the Windows Driver Kit (WDK) installed, by using Visual Studio. For more information about that, you will have to look into some Microsoft documentation:

Developing V4 Printer Drivers

<http://msdn.microsoft.com/en-us/library/windows/hardware/br259124.aspx>

Developing Metro style Device Apps for Printers

<http://msdn.microsoft.com/en-us/library/windows/hardware/br259129.aspx>

Device Metadata Package

<http://msdn.microsoft.com/en-us/library/windows/hardware/gg463157.aspx>

<http://msdn.microsoft.com/en-us/library/windows/hardware/hh833794.aspx>

For the installation instructions I will assume that you have a metadata file ready for use, whether you have made it yourself or the file has been distributed along with the metro device app project.

The devicemetadata-ms file has to be copied to the Device Metadata store folder in Windows 8. This folder can be found in the Program Data \ Microsoft \ Windows folder. On your machine, you can enter the following in the address bar of Windows Explorer:

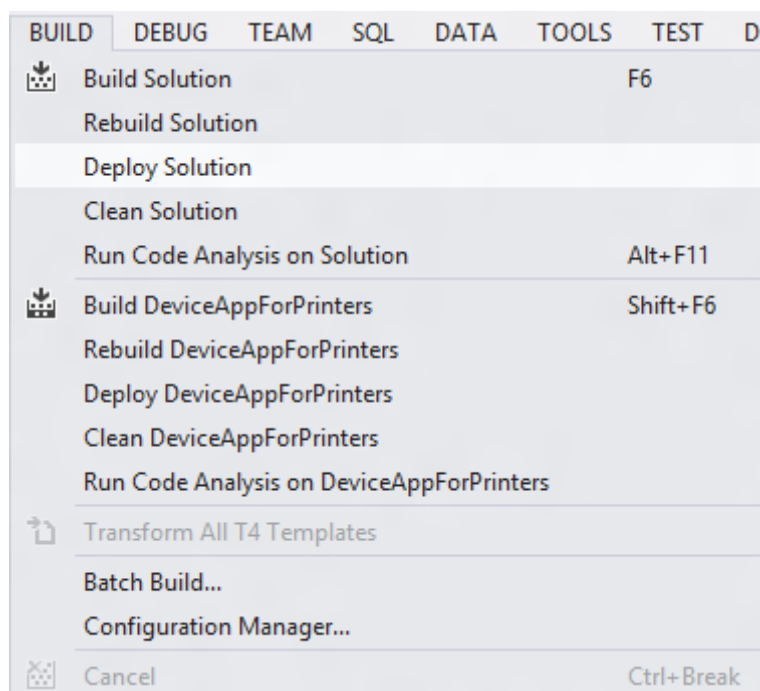
%programdata%\Microsoft\Windows\DeviceMetadataStore

Depending on the locale of the device metadata, you will have to put the file in a subfolder of the device metadata store. If the locale is undefined, place the file in the root, otherwise you will have to place it in a subfolder like "en-US". You will have to create the folder if it does not exist yet. There are a couple of ways to figure out the locale of the metadata package. The easiest way is to open it with the Visual Studio Device Metadata Authoring tool that you get when you have the WDK installed. You could also rename the file extension to .zip and open it with a tool like 7zip and see if there are any locales defined in the XML files contained within the zip. Since Windows 8, a device metadata package can contain multiple locales, while in Windows 7 that is only one.

Placing the file in the correct folder is the only thing that we have to do here, but this step is very important because doing this wrong will cause the association to fail, and your app will not be linked to the driver, and your app will be just another Metro app.

3 Metro Device App

The other thing you have to do to complete the installation is to build the app and deploy it from Visual Studio. Open the solution file (.sln) with Visual Studio, and confirm that you trust all projects contained within the solution. Then go to the Build menu, and click “Build Solution” (F6). This will build the app for you. If there are any errors during the build, check your environment and the project because something is wrong, and you might be missing some libraries, or references in a project are wrong.



When the build succeeds, select the Build menu again and click the third option: “Deploy Solution”. This will place the app to the local App Store on your machine, or whatever debug target that you have defined.

Instead of deploying the application, you can also choose to Debug the application. This will have almost the same effect, except this will also attach the Visual Studio debugger to the app on start, and depending on the project settings even launch the app for you. The app will be deployed during the process of setting up the debug session, and the app will be available in the Metro start menu from that moment on.

When you have completed the steps above, the app is ready for use. In the case of a printer driver and a metro device app for printers, this will mean that you now also can access the app from the “More Settings” button in the Metro Print fly-out, like displayed in the screenshot on the right. Be sure to validate this. If it does not work, and you do see the app in the Metro start menu, you probably have an issue with the device metadata. When the app loads and you can see a splash screen, you know that the association is valid. When the app immediately closes after loading, please try to reboot your Windows, because something might be broken in the supporting software framework.

