**Managing Print Schema Complexity** Using Print Schema System

October 19, 2008

Abstract

Information about Zoran’s Print Schema System (PSS), a facility to manage the complexity of the Microsoft® Print Schema, is provided in this paper.  It provides guidelines and examples for application and print driver developers to use the Print Schema System to access and control printing features quickly and easily.

This information applies for the following operating systems:

Windows® Server® 2008  
 Windows Vista®  
 Windows Server 2003  
 Windows XP

References and resources discussed here are listed at the end of this paper.

For the latest information, see:   
 <http://www.microsoft.com/whdc/>



**Author's Disclaimer and Copyright:** This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Zoran Corporation on the issues discussed as of the date of publication. Because Zoran must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Zoran, and Zoran cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. ZORAN MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Zoran may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Zoran, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2008 Zoran Corporation. All rights reserved. Zoran is a registered trademark of Zoran Corporation.

Windows Hardware Engineering Conference - WinHEC Sponsors’ Disclaimer: The contents of this document have not been authored or confirmed by Microsoft or the WinHEC conference co-sponsors (hereinafter “WinHEC Sponsors”). Accordingly, the information contained in this document does not necessarily represent the views of the WinHEC Sponsors and the WinHEC Sponsors cannot make any representation concerning its accuracy. THE WinHEC SPONSORS MAKE NO WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS INFORMATION.

Microsoft, Windows, Windows NT, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2008 Microsoft Corporation. All rights reserved.

Microsoft, Windows, Windows NT, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners

Contents

[Overview 4](#_Toc212350381)

[Architecture 5](#_Toc212350382)

[PSS Metadata 7](#_Toc212350383)

[Framework Layer 9](#_Toc212350384)

[Consumer Layer 9](#_Toc212350385)

[Consumer Layer API 10](#_Toc212350386)

[Accessing Features without PSS 11](#_Toc212350387)

[Accessing Features with PSS 13](#_Toc212350388)

[Application Usage Scenario 14](#_Toc212350389)

[Printer Driver Usage Scenario 15](#_Toc212350390)

[Conclusion 16](#_Toc212350391)

[Call to Action 16](#_Toc212350392)

[Resources 17](#_Toc212350393)

# Overview

With the introduction of the Print Schema for Windows XP, Windows Vista, and future Windows operating systems, a more robust way of sharing printing feature and configuration information among applications, drivers, and devices is now available. In place of the binary structures (such as DEVMODE) previously used to communicate such information, the Print Schema model provides an open, extensible XML format for communicating printer and print job information among application, driver, and device. The Print Schema defines the structure and content of PrintCapabilities and PrintTicket documents for communicating this information

The Print Schema supports a wide variety of printing features, and is extensible to support features not currently defined. The Print Schema specification is a detailed and thorough specification which provides a clear, consistent, and unambiguous XML format for specifying printing features. However, the Print Schema provides only a structure and format for feature information; the burden of generating and interpreting valid, standards-compliant XML remains with the driver and application developer. Generating valid Print Schema XML and navigating and interpreting it can be difficult and error prone. Such code may be brittle as the Print Schema specification evolves.

To mitigate these difficulties and to manage the complexity of Print Schema XML generation and parsing, Zoran Corporation has developed the Print Schema System (PSS), a COM-based library which application and driver developers can use to create and use PrintCapabilities and PrintTicket documents. The PSS library handles the parsing of PrintCapabilities and PrintTicket XML, encapsulating all detailed knowledge of the structure of the Print Schema and exposing printing features as simple, straightforward C++ data structures.

The Zoran PSS library is designed to be straightforward, consistent, extensible, and agile. The Print Schema System is exposed to the client through a simple COM API, which provides a familiar programming paradigm. Each feature and option defined in the Print Schema is identified in PSS by a simple enumerated keyword, and exposed to the client as a straightforward C++ data structure. The data structures defined by PSS directly reflect the Print Schema’s feature-option-property containment hierarchy.

The PSS facility is designed to be extensible and agile. The Print Schema is extensible through the definition of private Print Schema feature and option Keywords that are OEM specific; the PSS facility makes these Print Schema extensions available to its clients in exactly the same way as regular Print Schema Keywords. Because the data structures of the PSS facility are XML generated, any future changes or extensions to the Print Schema specification can be accommodated without any changes either to client code or to the PSS facility itself. For all the reasons previously mentioned, Zoran is using PSS as part of IPS DDK 3.0 ―a printer driver development kit for the GDI and XPS print paths.

# Architecture

The architecture of the Print Schema System is based on the following three components:

* Print Schema metadata
* Framework layer
* Consumer layer

These components enable a strict separation between the details of Print Schema XML and the client-facing C++ data structures, as well as supporting the ability of PSS to accommodate changes and extensions to the Print Schema.

Please see the following architectural overview of Print Schema System.

**PSS Client  
(Application or Print Driver)**

**PSS Consumer Layer**

**PSS Framework Layer**

**PSS-Generated Data Structures**

<?xml version="1.0" encoding="UTF-8"?><ipsddk:PrintSchemaSystem version="1">

<!-- 6. Metadata Keywords -->

<!-- 6.1. Job Metadata -->

<!-- 6.1.1. JobID -->

<psf:Property name="psk:JobID" ipsddk:validfor="PrintTicket">

<psf:Value xsi:type="xs:string">

( untitled )</psf:Value>

</psf:Property>

**PSS Metadata XML**

**Run-Time Metadata**

**Compile-Time Metadata**

**PSS Internal Format**

**Print Schema System**

<?xml version="1.0" encoding="UTF-8"?>

<psf:PrintTicket xmlns:psf="http://

schemas.microsoft.com/windows/2003/08/printing/

printschemaframework"xmlns:xsi=

"http://www.w3.org/2001/XMLSchema-instance"

xmlns:xs="http://www.w3.org/2001/XMLSchema"

xmlns:psk="http://schemas.microsoft.com/

windows/2003/08/printing/printschemakeywords"

xmlns:ipsddk="http://schemas.zoran.com/

imaging/ipsddk/3.0"version="1">

<psf:Feature name="psk:PageWatermark">

<psf:Option name="ipsddk:DRAFT">

<psf:ScoredPropertyname="psk:OriginWidth">

**PrintTicket / PrintCapabilities XML**

**Figure 1. PSS architecture**

# PSS Metadata

The PSS facility makes use of XML-based metadata not only to support agility and extensibility but also to expose Print Schema Keywords as programmer-friendly enumerations. PSS metadata is provided as a set of one or more XML files, each of which defines and describes a subset of the Print Schema Keywords supported by PSS.

The principal XML metadata file encapsulates all of the standard features, options, properties, parameters, and values defined by the public Print Schema specification. One or more secondary metadata files may also be used which similarly define OEM-private Print Schema Keywords. The full set of metadata files, taken together, defines all of the Keywords which PSS will support.

At the root of the metadata XML is the PrintSchemaSystem element. The PrintSchemaSystem element declares the same namespaces required for a PrintTicket or PrintCapabilities document (Print Schema Framework, Print Schema Keyword, XML Schema, and XML Schema Instance), the Zoran-specific namespace required for PSS itself, and any OEM namespaces within which private keywords are to be defined.

The child elements of the PrintSchemaSystem element are the Print Schema Framework elements (features, properties, parameters, and so forth) to be supported by PSS. Each child element fully specifies the keyword it defines, including the full range of options available for features; data type, default value, unit type, and length for parameter definitions; data type and value for parameter initializations; and so forth.

The following excerpt from the principal XML metadata file illustrates how PSS metadata captures the requirements of the Print Schema specification:

<psf:Feature name="psk:DocumentDuplex">

<psf:Option name="psk:OneSided"/>

<psf:Option name="psk:TwoSidedShortEdge"/>

<psf:Option name="psk:TwoSidedLongEdge"/>

<psf:Option name="psf:Options">

<psf:ScoredProperty name="psk:DuplexMode">

<psf:Value xsi:type="xs:string">Automatic</psf:Value>

<psf:Value xsi:type="xs:string">Manual</psf:Value>

</psf:ScoredProperty>

</psf:Option>

</psf:Feature>

**Figure 2. PSS metadata**

This PSS metadata enables the PSS facility to generate a C++ structure to represent the DocumentDuplex feature, and a C++ enumerated data type to represent the range of options available for the feature.

From the XML metadata listed in Figure 2, PSS generates the following C++ data structures[[1]](#footnote-2):

namespace PSSDocumentDuplex

{

namespace Option

{

enum DocumentDuplexEnum {

UNKNOWN = -1,

UNDEFINED = 0,

PSK\_ONESIDED,

PSK\_TWOSIDEDSHORTEDGE,

PSK\_TWOSIDEDLONGEDGE,

OEM\_000 = 256,

OEM\_001 = 257,

};

enum DuplexModeEnum {

UNKNOWN = -1,

UNDEFINED = 0,

AUTOMATIC,

MANUAL,

OEM\_000 = 256,

OEM\_001 = 257,

};

typedef struct tagData {

DuplexModeEnum enumDuplexMode;

} DuplexModeData;

}

typedef struct tagData {

Option::DocumentDuplexEnum enumOption;

Option::DuplexModeData option;

} PrintTicketData;

}// Type definition(s)

typedef PSSDocumentDuplex::Option::DocumentDuplexEnum

DocumentDuplexOptionEnum;

typedef PSSDocumentDuplex::Option::DuplexModeEnum

DocumentDuplexOptionDuplexModeEnum;

typedef PSSDocumentDuplex::Option::DuplexModeData

PTDocumentDuplexOption;

typedef PSSDocumentDuplex::PrintTicketData

PTDocumentDuplex;

**Figure 3. PSS-generated data structures**

These declarations enable the programmer to work with the occurrence of the DocumentDuplex feature in a print ticket by declaring a structure of type PTDocumentDuplex. The available options for the feature are referenced from C++ code using enumerated values, such as PSK\_TWOSIDEDSHORTEDGE.

In addition to driving the generation of the data structures used by programmers to access Print Schema data, PSS metadata is used at runtime to populate those data structures from the contents of actual PrintCapabilities and PrintTicket documents. Thus the knowledge needed to translate between Print Schema XML and programmer-friendly data structures is external both to client code and to the PSS facility itself. This enables PSS to adapt to OEM-private extensions to the Print Schema as well as to future changes to the Print Schema specification itself.

## Framework Layer

The Framework layer of the PSS architecture is responsible for dealing directly with PrintCapabilities and PrintTicket XML documents. The Framework has no knowledge of the client-facing data structures generated from the PSS metadata; and the Consumer layer has no knowledge of Print Schema XML.

The Framework is responsible for:

* Reading and writing PrintCapabilities and PrintTicket documents (from a stream or data buffer provided by the client).
* Parsing PrintCapabilities or PrintTicket XML.
* Translating the content of the PrintCapabilities or PrintTicket document into an intermediate binary format common to the Framework and Consumer layers.
* Generating print ticket XML from the intermediate binary format.

The client does not use the Framework-layer COM API; it is exposed only to the Consumer layer.

## Consumer Layer

The Consumer layer is the client-facing component of the PSS architecture. It provides a simple COM API which exposes the functionality of PSS to the client. Its responsibilities are:

* To manage the PrintCapabilities/PrintTicket scope hierarchy. PrintTicket and PrintCapabilities documents may exist at system, user, job, document, and page levels. The Consumer layer maintains a list of defined features and options at each level in this scope hierarchy.
* To manage memory for the PSS data structures at each level in the scope hierarchy. Client code declares only pointers to PSS data structures; the Consumer layer owns the memory for all feature data and frees it when it is no longer needed.
* To translate between the intermediate data format provided by the Framework and the PSS data structures used by the client, and vice versa. The PSS metadata drives this translation process.

## Consumer Layer API

The Consumer layer exposes the following methods to the PSS client:

|  |  |  |
| --- | --- | --- |
| Method | Description / Parameters | |
| Request(  PTKeywordEnum keyword,  void \*\* data,  BOOL scope) | Notify PSS of a keyword of interest. PSS allocates and returns a pointer to a data structure for the feature. | |
| keyword | Enum value identifying the feature of interest. Keyword enums are generated from PSS metadata. |
| data | Output parameter receiving the newly allocated data structure. |
| scope | Boolean value indicating whether system and user-level PrintTicket data is to be used. (Not used for PrintCapabilities.) |
| Remove(  PTKeywordEnum keyword,  BOOL scope) | Remove a feature from a PrintTicket. When WritePrintTicketis called, the removed feature does not appear in the generated PrintTicket XML. | |
| keyword | Enum value identifying the feature of interest. Keyword enums are generated from PSS metadata. |
| scope | Boolean value indicating whether system- and user-level PrintTicket data is to be used. |
| ReadPrintTicket(  IStream\* stream,  ScopeEnum scope) | Read a PrintTicket document and populate the PSS data structures for all requested features. | |
| stream | Stream from which the PrintTicket XML is to be read. |
| scope | Level of PrintTicket (job, document, or page) to be read. |
| WritePrintTicket(  IStream\* stream,  ScopeEnum scope) | Write a PrintTicket document. The current values in the PSS data structures for all requested features will be used to generate the PrintTicket XML. | |
| stream | Stream to which the generated PrintTicket XML is to be written. |
| scope | Level of PrintTicket (job, document, or page) to be written. |

# Accessing Features without PSS

The following code sample illustrates the complexity of working with Print Schema XML without the use of PSS. This sample comes from a hypothetical printer driver responsible for handling page order issues for a print job. The driver needs to know whether the job-level PrintTicket is requesting reverse-order printing. This code determines this by parsing and analyzing the PrintTicket XML itself. The code retrieves a simple Boolean value; no complex or nested data structures are involved. Yet the code to determine that simple Boolean value is not trivial.

BOOL IsReverseOrder(IStream\* pPrintTicketStream)

{

BOOL bReverseOrder = FALSE;

VARIANT\_BOOL bSucceeded = VARIANT\_FALSE;

HRESULT hrStatus = S\_OK;

int count = 0;

CComPtr<MSXML2::IXMLDOMDocument2> pDocument;

1 hrStatus = pDocument.CoCreateInstance(\_\_uuidof(MSXML2::DOMDocument), NULL,

CLSCTX\_SERVER);

if ( SUCCEEDED(hrStatus) )

{

2 hrStatus = pDocument->raw\_load(CComVariant(pPrintTicketStream),

&bSucceeded);

}

if ( SUCCEEDED(hrStatus) )

{

CComPtr<MSXML2::IXMLDOMElement> pRoot(NULL);

CComPtr<MSXML2::IXMLDOMNodeList> pChildren(NULL);

CComPtr<MSXML2::IXMLDOMNode> pChild(NULL);

CComQIPtr<MSXML2::IXMLDOMElement> pElement;

hrStatus = pDocument->get\_documentElement(&pRoot);

if( SUCCEEDED(hrStatus) )

hrStatus = pRoot->get\_childNodes(&pChildren);

if( SUCCEEDED(hrStatus) )

{

long count = 0;

CComVariant vName;

pChildren->get\_length(&count);

3 for( long x = 0; x < count; x++ )

{

hrStatus = pChildren->get\_item(x, &pChild);

if( hrStatus != S\_OK )

break;

if( (pElement = pChild) == NULL )

break;

pOption->raw\_getAttribute(CComBSTR(L"name"), &vName);

1. Instantiate a DOM object.
2. Load and parse the PrintTicket XML.
3. Traverse the DOM tree looking for the element of interest.

4 if( vName.bstrVal == CComBSTR(L"JobPageOrder") )

{

CComPtr<MSXML2::IXMLDOMNode> pFirst(NULL);

CComQIPtr<MSXML2::IXMLDOMElement> pOption;

hrStatus = pElement->get\_firstChild(&pFirst);

if( (SUCCEEDED(hrStatus)) && ((pOption = pFirst) != NULL) )

{

CComVariant vOption;

pOption->raw\_getAttribute(CComBSTR(L"name"), &vName);

if( vName.bstrVal == CComBSTR(L"Reverse") )

bReverseOrder = TRUE;

}

}

}

}

}

return bReverseOrder;

}

**Figure 4. Non-PSS parsing code**

1. Test for the feature of interest by name. When it is found, drill down to the child Option element and test the value of its “name” attribute to determine if the feature is selected. This program logic depends on the knowledge that the Print Schema specification requires exactly one child Option element of a JobPageOrder Feature element. With PSS, this knowledge is encapsulated in the PSS metadata and need never be reflected in program logic.

# Accessing Features with PSS

The following code sample accesses the same feature (reverse page order) as the previous native XML code sample, but using the PSS API instead. It illustrates these basic steps for using PSS:

1. Instantiate a PSS Consumer object.
2. Call the Request method to create data structures for features of interest.
3. Call ReadPrintTicket to populate the data structures.
4. Remove features as necessary.
5. Call the WritePrintTicket method to generate the changed print ticket.

HRESULT CPageOrderFilter::HandleReverseOrder(IStream\* pStream,

IStream\* pOutStream)

{

HRESULT hrStatus = S\_OK;

1 PTJobPageOrder\* pPTJobPageOrder = NULL;

2 ICorePrintSchemaSystemConsumer\*

pConsumer;

3 hrStatus = IPSDDKCoreCreateInstance(m\_cbPrinterName,

&pConsumer);

if( SUCCEEDED(hrStatus) )

{

4 pConsumer->Request(PSSKeyword::PrintTicket::Enum::PSK\_JOBPAGEORDER,

(PVOID\*)&pPTJobPageOrder,

TRUE);

5 pConsumer->ReadPrintTicket(pStream,

PSFScope::Enum::PRINTTICKET\_JOB);

6 if( pPTJobPageOrder->enumOption ==

PSSJobPageOrder::Option::Enum::PSK\_REVERSE )

{

hrStatus = ReversePages();

if( SUCCEEDED(hrStatus) )

7 pConsumer->Remove(

PSSKeyword::PrintTicket::Enum::PSK\_JOBPAGEORDER,

PSFScope::Enum::PRINTTICKET\_JOB);

}

8 hrStatus = pPrintTicketConsumer->WritePrintTicket(pOutStream,

PSFScope::Enum::PRINTTICKET\_JOB);

}

return hrStatus;

}

**Figure 5. PSS code without parsing**

1. Declare a pointer of type PTJobPageOrder to reference the PSS-generated data structure for the feature.
2. Declare an interface pointer to the PSS Consumer object.
3. Instantiate the PSS Consumer object.
4. Call Request to allocate the data structure for the feature. The PSK\_JOBPAGEORDER enum value is generated from the PSS metadata.
5. Read the PrintTicket to populate the data structure.
6. Test the content of the data structure and handle the feature if requested.
7. If the feature was handled by the driver, call the Remove method to remove the feature from the PrintTcket.
8. Call WritePrintTicket to generate a new, possibly changed, PrintTicket.

## Application Usage Scenario

This sample illustrates access to the print capabilities of a device from application code. In this scenario, a hypothetical application needs to produce collated copies of a document. The application uses PSS to determine whether the device supports collation; if so, it creates a print ticket including the collate feature. If not, the application does the collation in its own code.

VOID CollateCopies(IStream\* pOutStream)

{

HRESULT hrStatus = S\_OK;

PCDocumentCollate\* pPCCollate = NULL;

PTDocumentCollate\* pPTCollate = NULL;

ICorePrintSchemaSystemConsumer\* pConsumer;

1 hrStatus = IPSDDKCoreCreateInstance(m\_cbPrinterName,&pConsumer);

if( SUCCEEDED(hrStatus) )

{

2 pConsumer->Request(

PSSKeyword::PrintCapabilities::Enum::PSK\_DOCUMENTCOLLATE,

(PVOID\*)&pPCCollate);

3 pConsumer->Request(

PSSKeyword::PrintTicket::Enum::PSK\_DOCUMENTCOLLATE,

(PVOID\*)&pPTCollate,

TRUE);

4 if( pPCCollate->displayName.Length() > 0 )

{

5 pPTCollate->enumOption =

PSSDocumentCollate::Option::Enum::PSK\_COLLATED;

6 pConsumer->WritePrintTicket(pOutStream,

PSFScope::Enum::PRINTTICKET\_DOCUMENT);

}

else

{

7 // Collate in application code

}

}

}

**Figure 6. Using PSS from an application**

1. Instantiate the Consumer object.
2. Call Request to allocate data structure for the required PrintCapabilities feature. For PrintCapabilities documents, the Request method itself populates the data structures. No explicit read call is needed.
3. Call Request to allocate the data structure for the corresponding PrintTicket feature.
4. Test for the presence of the requested feature in the Print Capabilities.
5. If the feature is present (that is the device supports the feature, assign values to the data structure for the corresponding PrintTicket feature).
6. Call WritePrintTicket to generate PrintTicket XML, which includes the requested feature.
7. If the requested feature was not present in the PrintCapabilities, implement the feature in application code (not shown).

## Printer Driver Usage Scenario

The following code sample illustrates setting a feature in the print ticket. The SetBooklet function in this sample is part of a hypothetical printer driver user interface module. When the user selects booklet printing through the user interface, it calls SetBooklet.

VOID SetBooklet(IStream\* pOutStream)

{

HRESULT hrStatus = S\_OK;

PTJobBindAllDocuments\* pPTBind = NULL;

ICorePrintSchemaSystemConsumer\* pConsumer;

1 hrStatus = IPSDDKCoreCreateInstance(m\_cbPrinterName,

&pConsumer);

if( SUCCEEDED(hrStatus) )

{

2 pConsumer->Request(

PSSKeyword::PrintTicket::Enum::PSK\_JOBBINDALLDOCUMENTS,

(PVOID\*)&pPTBind,

FALSE);

3 pPTBind->enumOption =

PSSJobBindAllDocuments::Option::Enum::PSK\_BOOKLET;

pPTBind->option.bindingGutter = 300;

4 hrStatus = pConsumer->WritePrintTicket(pOutStream,

PSFScope::Enum::PRINTTICKET\_JOB);

}

}

**Figure 7. Using PSS from the driver user interface**

1. Instantiate the Consumer object.
2. Request a data structure for the desired feature. In this case the feature does not previously exist in the print ticket; the Request method adds a new feature to the internal feature list of the Consumer object.
3. Assign appropriate values to the data structure to reflect the choice of the user.
4. Call WritePrintTicket to generate the XML, including the new feature requested by the user.

# Conclusion

Zoran created PSS to enable developers to accurately process Print Schema documents with ease. PSS is a component of IPS DDK 3.0, Zoran’s XPS print driver framework; it is also available as a stand-alone component to other print driver and application developers. In IPS DDK 3.0, PSS is used throughout whenever PrintTicket or PrintCapabilities documents are created, accessed, or changed in print drivers. PSS enables developers to take full advantage of the precision, comprehensiveness, and extensibility provided by the Print Schema platform, without the need for complex, error-prone, and brittle code.

PSS eases the transition from binary, DEVMODE-based printer configuration to the extensible Print Schema platform by:

* Encapsulating the details of Print Schema XML.
* Exposing configuration data as native C++ structures.
* Providing a familiar COM API for accessing configuration data.
* Enabling developers to access configuration data with code that is simpler, less error prone, and better performing than ad-hoc XML parsing.

# Call to Action

* It is time to move away from DEVMODE.

DEVMODE is the device configuration mechanism for the GDI driver model. As a binary standard, its extensibility and adaptability are limited. With the advent of the Windows Vista driver model, the comprehensive, open, extensible, XML-based Print Schema is the new device configuration mechanism. Although DEVMODE is still supported through conversion APIs, the Print Schema is the way to manage configuration data going forward.

* Implement Print Schema support.

The Print Schema is the native print configuration mechanism for Windows Vista and future versions of Windows. All print drivers should fully support the Print Schema, and applications should be Print Schema-aware to take full advantage of the Windows Vista printing model.

* Contact Zoran for more information about using the Print Schema System for Print Schema support.

Zoran stands ready to help print driver and application developers move to full Print Schema support quickly, with elegant and well-performing code. Contact Zoran at [pssinfo@zoran.com](mailto:pssinfo@zoran.com?subject=Print%20Schema%20System%20(PSS)%20white%20paper) for more information on the Print Schema System.

# Resources

Print Schema Specification

<http://go.microsoft.com/fwlink/?LinkId=86086>

Print Schema on MSDN

<http://msdn.microsoft.com/en-us/library/ms716462(VS.85).aspx>

XML Paper Specification

<http://go.microsoft.com/fwlink/?LinkId=86085>

Ecma Office Open XML File Formats Standard

<http://www.ecma-international.org/news/TC45_current_work/TC45_available_docs.htm>

Ben Kuhn's Blog

<http://blogs.msdn.com/benkuhn/>

[Adrian Ford on XPS et cetera](http://blogs.msdn.com/adrianford/default.aspx)

<http://blogs.msdn.com/adrianford/>

XPSDrv Filter Pipeline

<http://www.microsoft.com/whdc/device/print/XPSDrv_FilterPipe.mspx>

Microsoft XML Paper Specification Essentials Pack

<http://go.microsoft.com/fwlink/?LinkId=86088>

Windows Driver Kit (WDK)

<http://go.microsoft.com/fwlink/?LinkId=86090>

XPS Home Page

<http://go.microsoft.com/fwlink/?LinkId=86091>

Windows Hardware and Device Central Home Page

<http://go.microsoft.com/fwlink/?LinkId=86093>



**Zoran Corporation**  
Imaging Division  
  
Tel: 781-791-6000  
Fax: 781-791-6111  
  
One Wall Street   
Burlington, MA 01803  
USA

**For questions or inquiries please email:** [pssinfo@zoran.com](mailto:pssinfo@zoran.com?subject=PrintSchema%20System%20Whitepaper)

1. This code excerpt is a simplified version of the actual generated code in the interest of brevity and clarity. [↑](#footnote-ref-2)