Oracle Payments in Release 12 – Take It To the Bank!

Glen T. Ryen

Prisio Technologies

Abstract

The Oracle E-Business Suite Release 12 Payments module provides many options for Funds Disbursement and bank integration, but the new concepts and terminology can be confusing for project managers, functional implementers, technical developers, and end users. This session will present upgraders and new implementers with a blueprint for bank integration and on-site check printing. Key implementation options, security considerations and best practices will be discussed. Developers will get an overview of creating and testing custom BI Publisher eText and RTF templates. Lessons learned from multiple implementations will also be provided – from project management, functional, and technical standpoints.

Introduction

The Payments module introduced in Release 12 of the Oracle E-Business Suite is a powerful engine for handling all of your organization's Funds Disbursement and Funds Capture needs. Oracle Payments centralizes functionality that (prior to Rel. 12) was scattered across different E-Business Suite modules. It processes transactions from several other Rel. 12 modules, such as invoice payments from Oracle Accounts Payables (AP), bank account transfers from Oracle Cash Management (CE), and settlements against credit cards from Oracle Accounts Receivables (AR). Oracle Payments also provides a platform for connecting to third party payment systems and financial institutions.

This paper looks to provide a blueprint for companies implementing or upgrading to the Rel. 12 Payment module and looking to integrate with their bank(s) for automated electronic payments to suppliers and/or securely print on-site AP checks out of Oracle. Each of three areas will be broken down considering the user's role(s) in the implementation or upgrade, with a general process overview and tips/lessons learned section for each role/area:

- **Implementing Payments**
  - Project Manager/Functional Implementer – Overview
  - Project Manager/Functional Implementer – Tips & Lessons Learned
  - Technical Developer – Overview
  - Technical Developer – Tips & Lessons Learned
- **Electronic Payments**
  - Project Manager/Functional Implementer – Overview
  - Project Manager/Functional Implementer – Tips & Lessons Learned
  - Technical Developer – Overview
  - Technical Developer – Tips & Lessons Learned
- **On-site Printed Checks**
  - Project Manager/Functional Implementer – Overview
  - Project Manager/Functional Implementer – Tips & Lessons Learned
  - Technical Developer – Overview
  - Technical Developer – Tips & Lessons Learned

While this paper focuses on bank integration, there are other areas that may be a part of (or should be considered during) your implementation of or upgrade to Rel. 12 Oracle Payments that are outside the scope of this paper. This paper will not address:

- Funds Capture functionality
- Delivery of Separate Remittance Advices to Suppliers
- Positive Pay functionality for check fraud prevention
- Multi-language reporting requirements on checks, remittances, or other reports
- Oracle Cash Management functionality for payment clearing and bank statement loading/reconciliation
- Oracle Accounts Payable functionality for Invoice processing

While this paper does not aim to provide detailed, step-by-step instructions for any one particular area or topic, high level overviews will summarize the important aspects of each topic, with pointers to relevant documentation and resources for the reader to follow up on. Useful references, links, and scripts will be provided in the appendices.

Implementing Payments

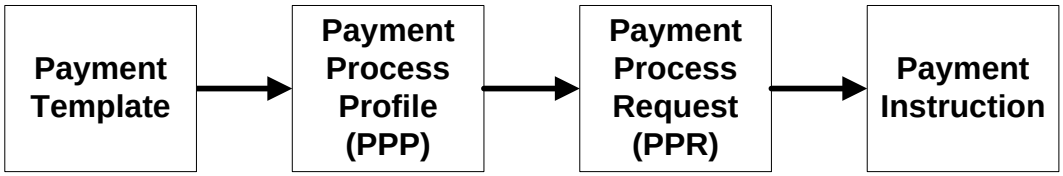
A firm understanding of the module's key functionality will be critical to any Oracle Payments implementation and bank integration effort. At a high level, those key functions are:

- **Defaulting** - Payment Method defaulting on source documents (AP Invoices, in this context)
- **Validation** - Seeded and user-defined validations, both on source documents and within Payments
- **Aggregation** – Collecting and grouping source documents into Payment Process Requests (PPR's) and Payment Instructions
- **Formatting** – Outputting generated payments into the required format, whether that's an electronic file coded to a pre-determined specification (i.e., NACHA) or a PDF image of a physical check
- **Transmission** – Sending generated Payment files securely to their destination(s).

Getting Started
My Oracle Support (MOS) Note #1391460.1 is an excellent place to start. Or go to the "Overview" and "Understanding Funds Disbursement" sections of the Oracle Payments Implementation Guide (pages 1-1 and 3-26, respectively) to gain an understanding of these key concepts.

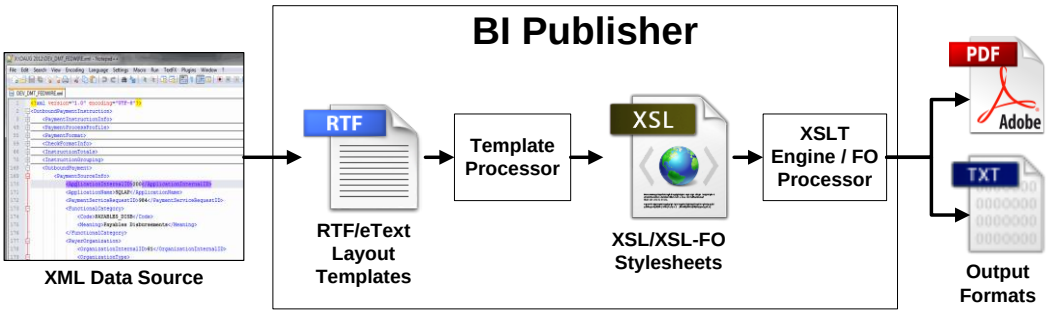
For companies upgrading from Rel. 10.x or 11.x Accounts Payable, MOS Note #733537.1 is a thorough walkthrough of the specific changes you'll be facing. These include new Oracle Payments transactional and setup tables (in the IBY schema), new bank account tables (in the CE schema), and the migration of Supplier data to the new Trading Community Architecture (the TCA tables in the HZ schema).

Once you are familiar and comfortable with the new concepts and terminology, see Section 4 of the Oracle Payments User's Guide – "Funds Disbursement Process Flows". Make sure you understand the flow from:



1. **Payment Templates** facilitate creation of Payment Process Requests. They populate required and optional fields on the PPR, such as the Payment Process Profile.
2. **Payment Process Profiles (or PPP's)** specify how to handle of the documents payable, payments, and payment instructions within a Payment Process Request.
3. **Payment Process Requests (or PPR's)** are analogous to a payment batch in Rel. 10.x or 11.x AP, in terms of the steps they go through – Select, Build, Format, and Confirm. PPR's will contain one or more Payment Instructions, though.
4. **Payment Instructions** are a collection of payments, with aggregate instructions.

Finally, be sure you understand how Oracle's Business Intelligence (BI) Publisher tool fits into the Funds Disbursement picture. BI Publisher (as a general reporting tool) takes data from Data Sources, applies a Layout Template to it to produce a desired output, and optionally deliver it to a destination:



From a Funds Disbursement standpoint, you'll be working with a seeded (but extensible) Data Source for outbound payments. From there you'll apply seeded or custom Layout Templates to produce your electronic payment file or on-site printed checks. After that, you may leverage BI Publisher functionality through Oracle Payments screens to deliver your output to its destination.

Project Manager/Functional Implementer – Overview

The general steps for implementing Oracle Payments are well documented in the Payments Implementation Guide. Begin with Section 2 ("Planning Your Implementation") before moving to Sections 4 and 5 ("Shared Setup Tasks for Funds Capture and Funds Disbursement" and "Setup Tasks for Funds Disbursement"). For bank integration, most of the important questions to answer early on in your implementation/upgrade project are there for you:

- What are your security needs? Consider them as comprehensively as possible, including:
  - Electronic requirements – File encryption/signing method, transmission protocol, etc.
  - Procedural requirements – Separation of duties in AP, Sarbanes-Oxley (SOX) compliance, etc.
  - Data security – Masking of Supplier bank account information, allowing override of Payee Bank Account on proposed Payments, etc.
  - Fraud prevention measures – Positive Pay files, physical check security features, etc.
  - Physical security – Locking or restricting access to the physical printer and check stock, etc.
- What formats do you need to support? The number of different electronic file formats (NACHA, SEPA, bank proprietary, etc.) and on-site printed check layouts will determine the bulk of your technical (and testing) workload and resource requirements.
- What disbursement Payment Methods do you want to support? See page 2-9 in the Payments Implementation Guide for discussion of the level of granularity here.

To that, add:

- What modifications will you need to make to seeded Oracle Payments reports to run your business? Examine in particular the seeded Cash Requirements Report and the Preliminary/Final Payment Registers.
- What are your requirements for printing or delivery of Separate Remittance Advices to your Suppliers? While outside the scope of this paper, this needs to be considered.
- Do you have the right skill sets in house, from a functional, technical, and project management standpoint? How will you address any areas where you feel you're deficient (training, hiring, consultants, etc.)?

Project Manager/Functional Implementer – Tips & Lessons Learned

Key lessons learned from earlier implementations can be broken down into several areas, related here to the different project phases (in a classic "waterfall" methodology) that they generally apply to.

Analysis/Design
From an Analysis perspective, focus on identifying the interfaces and template work required. As mentioned above, this is one of the biggest determinants of your workload and resource requirements. Adding interface or template work to the scope when you're further along in the project (or already live) adds to the overall project cost (and the implementation team's stress level!).

Second, be sure to engage your bank, your IT department, and (if applicable) your Oracle instance hosting provider early on in the process. Many of your security requirements will come out of these discussions, and these items tend to be longer lead time issues – such as having to procure, gain approval for, and install some required security software, adding a file transfer server to a Demilitarized Zone (DMZ), etc. Also, be sure to plan for long testing cycles of your new Funds Disbursements process. If your bank doesn't insist on it for their acceptance testing, you should – for your own comfort with all aspects of the process before you go live.

Third, be sure to concentrate on obtaining buy-in for your envisioned "to-be" state. Your decision makers and end users should all understand Payments' ultimate design, and be well aware of what the resulting build will look like. This should be done prior to testing in order to prevent confusion, conflict, and time-wasting.

Finally, be sure to keep things as simple as possible with the number of Payment Templates you initially define. From an end user perspective, these Payment Templates are what drive the Funds Disbursement process. They pre-populate most of the required fields for submitting Payment Process Requests, so the tendency may be to create a Template for each conceivable payment scenario. But too many Payment Templates will create confusion when users don't know which one to pick. It can also lead to communication issues if you have separate users/groups handling AP Invoice Processing vs. Payments. The former group may be running Cash Requirement reporting off one template while the latter is running the actual payments off a different one, and neither group will know why they are not getting the same results.

Testing
Be sure to test your payments process with as much converted and user-entered data as possible. This will vet out both conversion issues (in terms of record effective dates, correctness of supplier bank account assignments, etc.) and user training issues. Your outbound payments can go through bank integration testing just fine when the developer of the templates controls the test data and scenarios, but the process will be almost certain to fail when the users are in control. So get your users and developers to agree on:

- Which Payment Methods are valid for which disbursement accounts, currencies, and payee countries
- Which values are require per Payment Method
- What data goes where on the Supplier Banks, Bank Branches, and Bank Accounts

Next, expect to run into issues and be sure that you've built enough time and resources into your project plan to regression test any required patches thoroughly. Oracle Payments is a fairly complex module and you're bound to run into the typical issues related to your specific installation. Test for performance as well, if you'll have high transaction volumes. This can reveal bugs in seeded code and inefficiencies in your BI Publisher Layout Templates.

User training
Whether coming from Oracle Accounts Payable Rel. 10.x/11.x or some other ERP or accounting package, the concepts and flows of Oracle Payments will be new and unfamiliar to your end users. So be sure that your users know which Payment Methods to use when (considering payment currency, originating and receiving bank account countries, payment urgency vs. your transaction cost, etc.). Users also need to clearly understand the flow of default values like the Payment Method or payee bank account, from Supplier -> Site -> Invoice -> Payment Schedule, and how those defaults can be overridden. Ultimately, they must also understand that what is on the Invoice Payment Schedule is what gets used to create the payment.

As a result of how default values cascade down to Invoices, Supplier setup is very important. Be sure that your initial Supplier conversion or setup is accurate, especially with respect to the primary account and default Payment Method assigned to each Supplier/Supplier Site. Users handling Supplier maintenance should also know when to use Intermediary and Factor accounts if you have them, and where that information needs to be stored.

To that end, a best practice is to have the users actively involved in the testing with all of the Payment Methods before go-live in a real business setting. Don't just have the users test a few different iterations out, but have them test them all out to gain an understanding of the interdependencies between the information on the Invoices/Supplier records and the Payment Templates.

A final lesson learned for user training is relevant if you separate your payment runs by Pay Group. If you do create a Payment Process Request with a Payment Template that specifies a list of Pay Groups to pay, Oracle Payments will initially select just Invoices with those Pay Groups on them. But be aware that your users can (and often will need to) manually add invoices to the Payment Process Request after initial Invoice selection. If one of those manually-added Invoices does not have a value for Pay Group on it, or has a Pay Group different from the list specified on the Payment Template, the payment Build request will fail without much of an error message in the log file to tell you why. Make your users aware of this pitfall.

Contingency planning
Finally, it's always a good idea to expect the best but prepare for the worst. So be sure you have a fallback plan for processing payments right after go-live. Have some manual paper checks ready and/or maintain access to your bank's website for manually initiating wires and ACH payments. Consider this planning not just for complete failures of being able to disburse funds through Oracle Payments, but (more likely) to address individual payments that get stuck in the payment process due to data quality issues, Oracle bugs, etc.

Technical Developer – Overview

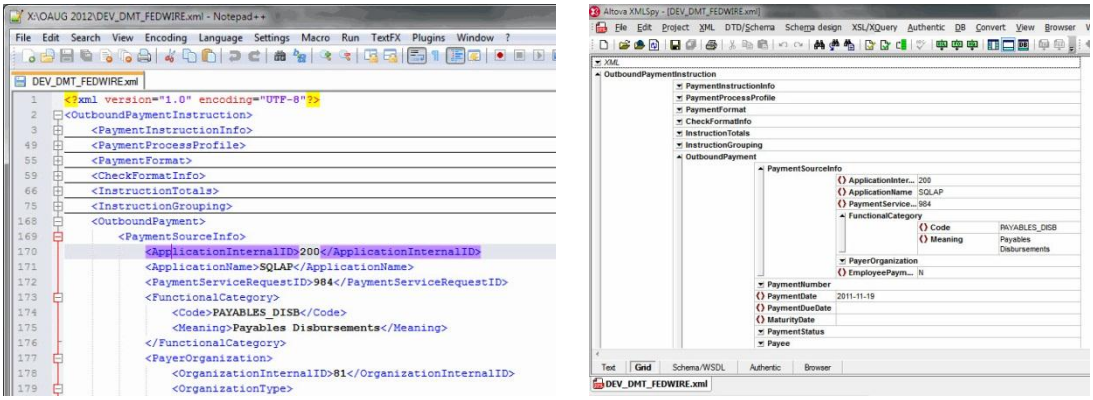
If you're a developer planning to integrate Oracle Payments with a disbursing bank, start by ensuring that you're familiar with the key XML (e**X**tensible **M**arkup **L**anguage) and XSL (e**X**tensible **S**tylesheet **L**anguage) technologies involved in creating custom templates. First you must know the basic structure of an XML document and three key concepts – elements, attributes, and tags. Second, learn the basics of XPath expressions for navigating an XML document tree or hierarchy. After that, an understanding of XSLT (**XSL T**ransformations, used by eText templates for transforming XML documents) and XSL-FO (**XSL F**ormatting **O**bjects, used by RTF (Rich Text Format) templates for formatting XML documents) may come in handy. Oracle's BI Publisher Desktop tool shields you (for the most part) from having to know XSLT and XSL-FO, but knowledge of what's going on under the covers helps in debugging template issues during development, and in satisfying more advanced formatting requirements.

Second, download and install on your development PC the BI Publisher Desktop tool - Patch #12395372 on the My Oracle Support site. This will install a plugin for Microsoft Word that you'll use for developing your templates, as well as a Template Viewer application that allows you to preview your templates locally.

After that, you'll benefit from having some form of XML editor on your machine. These fall under the following categories, generally arranged in terms of increasing features, complexity, and cost:

- **Text Editors** – Since XML files are plain text, the default text editing application for your operating system (like WordPad for MS Windows) will work. But a programmer's text editor like NotePad++ or jEdit (with the XML plugin) will add XML-specific features like syntax highlighting and code folding, for collapsing entire XML nodes into a single line with just a click.
- **Free or Open-source XML Editors** – An XML-specific editor will add other feature, like being able to view your XML in a tree or grid layout, or automatically generating XPath expressions for you. Examples here include XMLFox Editor, Serna XML Editor, and XML Notepad.
- **Java IDE's** – If you also program in Java, your Java IDE (Integrated Development Environment) of choice (like Oracle JDeveloper, Eclipse, or NetBeans) should come with some XML-specific functionality (or offer XML/XSL plugins to add those features).

- **Commercial XML IDE's** – For more involved XML development work, a commercial XML IDE will offer the most features, though generally with more complexity and cost. Examples include Altova XMLSpy, Stylus Studio, Liquid XML Studio, <oXygen/> XML Editor, and XML Marker.



Two views of the same XML in different editors, showing code-folding and syntax highlighting on the left and a grid view on the right.

For the template development related to Funds Disbursement bank integration, any of the free or open-source XML or programming text editors will suffice. If you're doing additional XML, Java, and/or Web Service development, however, it may make sense from a productivity standpoint to invest in a more full-featured tool. Evaluate your options, since even most of the more expensive commercial tools will offer a free trial period. Then use what works best for you.

Beyond those XML and XSL basics, familiarity with other programming languages will help for developing the encryption and transmission logic you'll need. For encryption, you may need to be familiar with PGP or GPG, while Java, Perl, or UNIX shell scripting could all be used for developing your transmission logic.

Technical Developer – Tips & Lessons Learned

Getting the XML

From a development standpoint, it will be quickest if you can get some of the raw XML that you'll be using and start your development with that. Being able to browse that XML file while developing, to mock up additional data conditions within that same file, and to apply your templates against it locally will all save you a lot of time. Save the uploading to the server and running of actual Payment Process Requests for your final rounds of testing.

For Funds Disbursement, you'll be using a seeded Oracle Data Definition called "Oracle Payments Funds Disbursement Payment Instruction Extract 1.0" (short Code "IBY_FD_INSTRUCTION_1_0"). The elements within that XML file are detailed in Appendix K ("Funds Disbursement Extract") of the Oracle Payments Implementation Guide, or in the XML Schema file located on the applications tier at $IBY_TOP/patch/115/publisher/defs/IBY_PPIOUT_1_0.xsd.

You can get that seeded XML output file to your PC in a few ways:

1. From the log file of the "Format Payment Instructions with Text Output" Concurrent Request when you run a Payment Process Request. That log file will have the first 1MB of the unformatted XML, and for smaller payment runs that will be ample for the entire XML file contents.
2. Depending upon your setup, you may also have a "View XML" button available. You can access this from the View -> Request menu, then by clicking on the "Diagnostics" button for the format request above.
3. The XML for a Payment Process Request is also stored in the IBY_TRXN_DOCUMENTS database table, if you have SQL access.

4. If those options don't work, you can assign the "Extract Identity" template to a Payment Format and run a Payment Process Request using that Format. That seeded template (file name of IBY_IDENTITY_en.xsl) passes the entire XML file through unmodified to the format request's concurrent request output file.
5. As a last resort, you can turn on server-level debug using the xdodebug.cfg file in the $AF_JRE_TOP/lib directory. Running the format process for a Payment Process Request should then log the entire XML to the directory you've specified in the "LogDir" entry of that xdodebug.cfg file.

See the References section for more details on some of those methods.

Extending the XML
Once you've successfully managed to retrieve that file for a Payment Process Request run in one of your instances, verify that it contains all the fields that you'll need to put in your electronic payment files and on-site printed checks. If it doesn't, Oracle gives you the ability to extend that XML file by modifying the IBY_FD_EXTRACT_EXT_PUB PL/SQL package. Any field values that you can retrieve from that PL/SQL package can be added as an XML element to the output, at one of five levels:

- Once per Payment Process Request, **or**
- Once per Payment Instruction, **or**
- Once per Payment, **or**
- Once per Document Payable (AP Invoice), **or**
- Once per Document Payable Line (AP Invoice Line).

See the Reference section for a link to an example of how to accomplish that if you need to.

Developing your Layout Template
When you've gotten the XML output and confirmed that it contains all the fields you need (or extended it so that it does), make a point of browsing the available Layout Templates (both seeded in Oracle and on the internet) for one that most resembles the structure of your final output. It's often quicker to model your Template after one of these, rather than beginning with an entirely clean slate.

Once you've started your custom template, it's very helpful to always keep in mind where you are in the XML file hierarchy at any given point in time. Your template (eText or RTF) will be looping through some or all of the Payments, Invoices, and Invoice Lines in that source XML to create its output. So how you tell the template to get a particular field (the XPath expression you use) will depend on which loop you're in, as that will determine how many levels up or down you need to navigate from your current position in the XML tree. Simple errors in XPath syntax account for most issues with your template not transforming or formatting the XML. Capitalization of element names is another big source of those types of errors, since XML element names (and therefore XPath expressions) are case-sensitive.

To debug issues like that, know how to enable debug when you try to preview your template output. Debug information can be viewed in:

- **Microsoft Word** – See the Reference section for a post in Ike Wiggin's blog on how to turn this on with a local copy of xdodebug.cfg, **or**
- **Template Viewer** – The BI Publisher Desktop application, via the "Log (debug) Level" dropdown box under the "Setting (default)" tab, **or**
- **Server Debug Directory** – The debug directory specified in the xdodebug.cfg file, if you've created that file on the application server.

Finally, know what tools are available to you when you need to migrate your completed template from your development instance to the test or production systems. Oracle gives you FNDLOAD to upload and download the layout template metadata, and XDOLoader to upload and download the physical RTF files.

Electronic Payments

Project Manager/Functional Implementer – Overview

For electronic payments specifically, one of your first priorities should be to determine your encryption and transmission security requirements. This is where you'll need technical consensus among several possible groups (your bank, internal IT staff, technical representatives from your hosting provider, internal auditors, and AP staff) as to how you'll secure file transport (e.g. HTTPS/SSL, SFTP or FTPS) and encrypt file contents (e.g., X.509/SSL, PGP, or GPG). Start these discussions early, as you'll likely need time for some of the following:

- Obtaining and exchanging keys and certificates,
- Installing any additional software or key stores on your server,
- Opening ports and modifying access rules on your organization's firewall rules,
- Establishing a Virtual Private Network (VPN) connection between your organization and the bank, and/or
- Procuring and installing new servers (if your transmission server needs to be placed in a DMZ).

Second, be sure your project's instance strategy arranges for development and test instances – both on your end and the bank's.

And even though they're outside the scope of this paper, be sure you also consider:

- What are your requirements for delivering Separate Remittance Advices to your Suppliers?
    - Will you deliver these Advices yourself and/or outsource that function to your bank?
    - What delivery method(s) will be used?
    - What layout and contents will the Remittance Advice have?
- Will you be retrieving any confirmation files from the bank?
- How will you notify users of bank rejects and handle voiding payments in Oracle?

Project Manager/Functional Implementer – Tips & Lessons Learned

Allowing yourself time to define (and test) your own User-defined Validations is something you definitely should plan for. Oracle allows you to set up Validations for each Payment Method you define and then enforce these Validations at two points in the Funds Disbursement flow – at the time of Invoice entry and at the time of Payment build. Invoices can be placed on a Scheduled Payment hold if they don't contain all the fields that your bank will require for that Payment Method, thus ensuring that only complete transactions make it through to be transmitted.

These Validations can check for the presence of field values, the length of those fields (against a minimum, maximum, or exact number of characters), or the contents of those fields (against a list of valid or invalid values). At a minimum consider implementing User-defined Validation on the following fields, as applicable for your Payment Methods:

- **Payee Bank Branch Number** – Should be exactly nine digits for a US ABA routing number, six digits for a UK Sort Code, etc.
- **Payee Bank BIC (Bank Identifier Code)** – Eight to eleven characters, used for most international electronic payments like CHIPS (Clearing House Inter-Payments System) wires and SEPA (Single European Payment Area) Credit Transfers.
- **Payee Bank Account Number or IBAN (International Bank Account Number)** – Check for a value in one of these fields, and the number of characters present. Note that the field format and length varies by country.
- **Payee Bank Country** – Check this against the list of country codes that your bank will accept for that Payment Method (US for ACH, one of the 37 ISO country codes in the Single European Payment Area for SEPA, etc.)
- **Payer Internal Bank Account** – Since your user/company ID in the bank's system will be associated with your Internal bank account, be sure this field has been pulled in your Payment Instructions

- **Payment Currency** – Check this against a list that your bank will accept for that Payment Method (US NACHA payments must be in USD, SEPA payments must be in Euro, etc.)
- **Payment Document Number** – These will be your check numbers, when sending a printed check file to your bank or a third party for outsourced printing. Generally not required for ACH's and wires.

While these Validations can confirm the presence of certain fields, they're limited in how they can validate the content. You can't create a User-defined Validation to validate logical "OR" conditions, to say that a certain type of wire payment requires either an Account Number or an IBAN. You also can't call your own PL/SQL, say to validate that a routing number check digit is correct.

For that last type of validation, you should be using the seeded functionality of the Supplier maintenance pages to validate country-specific rules for Bank Number, Account Number, etc. See Appendix page B-3 of the Payments Implementation Guide, "Country-Specific Validations". If you find, though, that Oracle is incorrectly rejecting valid banking information for your Supplier, those validations can be disabled via a profile option ("CE: Disable Bank Validations", available if you have patch #7582842). If you find yourself in that situation, you should log an SR with Oracle Support to get the validations corrected for the countries in which you'll be making payments rather than disabling the validations entirely.

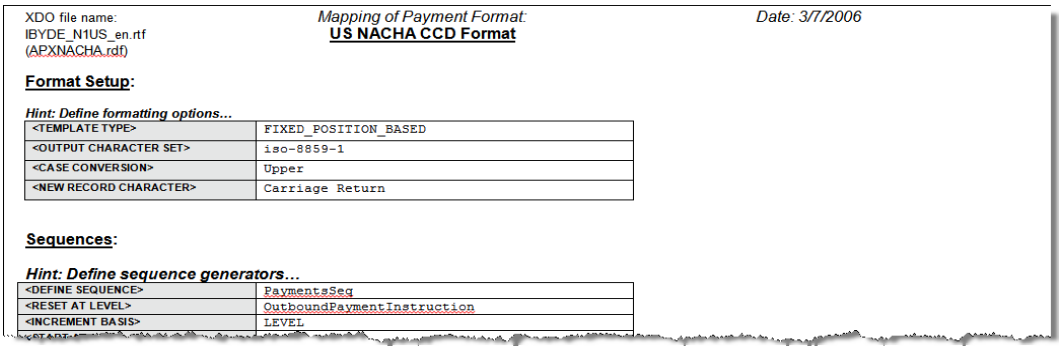
And finally (to re-iterate), be sure you allow enough time to thoroughly test your process from end-to-end. You and your bank should expect to test all the various combinations of payments you may have. At a minimum that's:

(Number of disbursement accounts)
x (Number of distinct payment types)
x (Number of currencies you pay in)
Number of payment scenarios to test

For payment types, don't just consider the main categories like NACHA and wire. Count instead the distinct sub-types, like CCD, PPD, or CTX within NACHA or Book, FEDWIRE, and CHIPS within wires. For each scenario, be sure to test both positive and negative, good transactions and transactions with missing or invalid key data elements. You're testing not just your process, but programs on the bank's end as well. Your goal should be to have the final round of tests run with production-like volumes of converted data, to find out both how good your data is and if you have any performance issues to address.

Technical Developer – Overview

BI Publisher's eText templates make creating fixed field width or delimited text files from E-Business Suite data much easier than before. The code you'll be producing is an RTF document that looks very much like a technical specification:



XDO file name:
IBYDE_N1US_en.rtf
(APXNACHA.rdf)

Mapping of Payment Format:
US NACHA CCD Format

Date: 3/7/2006

Format Setup:

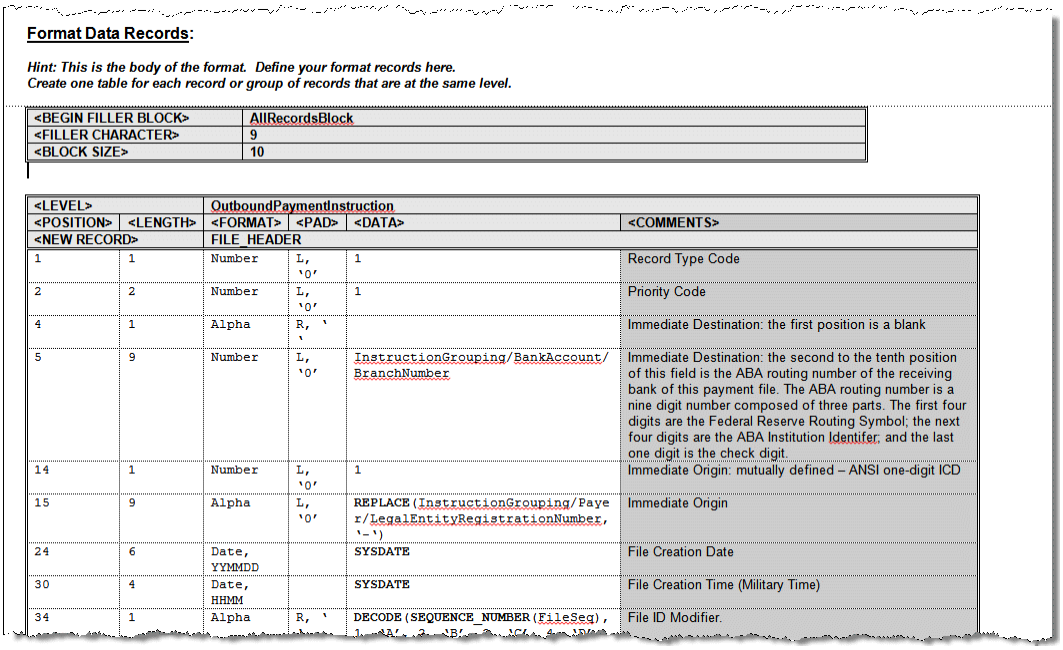
Hint: Define formatting options...

<TEMPLATE TYPE>	FIXED_POSITION_BASED
<OUTPUT CHARACTER SET>	iso-8859-1
<CASE CONVERSION>	Upper
<NEW RECORD CHARACTER>	Carriage Return

Sequences:

Hint: Define sequence generators...

<DEFINE SEQUENCE>	PaymentsSeq
<RESET AT LEVEL>	OutboundPaymentInstruction
<INCREMENT BASIS>	LEVEL



Format Data Records:

Hint: This is the body of the format. Define your format records here.
Create one table for each record or group of records that are at the same level.

<BEGIN FILLER BLOCK>	AllRecordsBlock
<FILLER CHARACTER>	9
<BLOCK SIZE>	10

<LEVEL>		OutboundPaymentInstruction			
<POSITION>	<LENGTH>	<FORMAT>	<PAD>	<DATA>	<COMMENTS>
<NEW RECORD>		FILE_HEADER			
1	1	Number	L, '0'	1	Record Type Code
2	2	Number	L, '0'	1	Priority Code
4	1	Alpha	R, ' '		Immediate Destination: the first position is a blank
5	9	Number	L, '0'	InstructionGrouping/BankAccount/BranchNumber	Immediate Destination: the second to the tenth position of this field is the ABA routing number of the receiving bank of this payment file. The ABA routing number is a nine digit number composed of three parts. The first four digits are the Federal Reserve Routing Symbol; the next four digits are the ABA Institution Identifer; and the last one digit is the check digit.
14	1	Number	L, '0'	1	Immediate Origin: mutually defined – ANSI one-digit ICD
15	9	Alpha	L, '0'	REPLACE(InstructionGrouping/Payer/LegalEntityRegistrationNumber, '-')	Immediate Origin
24	6	Date, YYMMDD		SYSDATE	File Creation Date
30	4	Date, HHMM		SYSDATE	File Creation Time (Military Time)
34	1	Alpha	R, '	DECODE(SEQUENCE_NUMBER(FileSeq),	File ID Modifier.

You specify the record formats, fields to include, whether data will be left padded with zeroes, right padded with spaces, etc. BI Publisher will take care of transforming that RTF document into an XSL stylesheet, which it then applies to the XML data to produce the desired text output.

The key steps are as outlined above:

- Install the Template Viewer application if you haven't already (patch #12395372).
- Get a copy of the raw (unformatted) XML from a Payment Process Request in your instance.
- Find the seeded template that most closely matches your file structure, in terms of whether it includes file/batch headers/footers, just payment level records or invoice details as well. Use that as your model.
- Modify the commands (primarily the Data Rows) within the RTF to meet your field layout needs.
- Unit test locally in the Template viewer application before uploading to the server.

Once your eText template is complete, you can design and build your transmission logic. This could leverage one of the seeded Transmission Protocols that Oracle Payments delivers, such as SFTP. The limitation with the seeded Protocols is that they likely will not support all the requirements that your bank will stipulate, such as encrypting, compressing, and/or digitally signing the payment file before transmission.

For those familiar with servlet programming, you can code your own Transmission Servlet to transmit your files to your bank. This servlet would need to answer an HTTP or HTTPS request from Oracle Payments and forward it along synchronously to the bank. See Chapter 8 ("Using Oracle Payments with External Payment Systems") of the Implementation Guide. Put in the terms of that guide, this would be a "processor" type Payment System running in batch mode.

A simple and often quicker alternative to developing a Transmission Servlet is a server-side script (e.g., a UNIX shell script). This can be run as a scheduled concurrent request to transmit any outbound payment files queued up in your outbound directory (specified in your Payment Process Profiles). It can be manually submitted by the users as the final step of the payment flow, or you could set it to fire automatically after the file is generated. A good option there would be to have the completion of the "Format Payment Instructions with Text Output" Concurrent Request raise a Business Event, which could trigger another Concurrent Request or BPEL (Business Process Execution Language) process. Note that this does require configuration and testing to ensure your Business Event system is handling the publishing and subscribing correctly for those events.

Technical Developer – Tips & Lessons Learned

Some tips for eText template development:

- Start with a seeded template as a model. Don't start your RTF from square one. Seeded layout templates like "US NACHA Generic Format" (IBYDE_N_US_en.rtf) or "US CCDP Consolidated Format" (IBYDE_C1_en.rtf) contain working examples of many of the eText features you may need to leverage.
- Develop, debug, and preview locally in the Template Viewer, manually modifying your source XML at first to create your test data conditions. Again, this is much quicker than having to enter Suppliers and their bank information; create, validate and approve Invoices; select, build, and format a Payment Process Requests; and then void payments and start the Payment Process Request over for each change you make to the template.
- Know the features available to you. The basic Commands:

    - <TEMPLATE TYPE> – For whether you would like a fixed-width or delimiter-based output.
    - <DEFINE LEVEL> – To re-group or sort the source XML data.
    - <DEFINE SEQUENCE> – For file, batch, or record numbers.
    - <DISPLAY CONDITION> – To only display a record if a given data condition is satisfied.
    - <BEGIN FILLER BLOCK> – For blocking records (a NACHA requirement for some banks)
    - <DEFINE CONCATENATION> – To referenced concatenated child records at the parent level

    as well as available Expressions and Operators such as:

    - IF-THEN-ELSIF-END IF
    - COUNT
    - SUM
    - DECODE
    - SUBSTR

You'll be hard-pressed to find a requirement for the contents of an electronic payment file that you can't meet with the features of eText templates. For developers, it's usually just a matter of being aware of those available features and knowing how to code them in your template. A good reference here is Chapter 4 ("Creating an eText Template") in the Oracle XML Publisher Report Designer's Guide.

On-site Printed Checks

Project Management/Functional – Overview

The process of implementing on-site printed checks in Oracle Payments should begin like that for electronic payments – with determining your security requirements. Get the appropriate stakeholders and decision-makers from your Accounts Payable or Treasury department together with your internal auditors. Consider your security holistically, including:

- Will you use blank check stock or stock with pre-printed MICR (Magnetic Ink Character Recognition) information?
- Will you print signature images? Where will your signature files be stored and how will they be secured?
- Are the seeded responsibilities for check printing acceptable to you? Some clients find the seeded "Payables Manager" responsibility not to be SOX compliant, because a user with that responsibility can take a Payment Process Request from initiation to completion without another user being involved.
- How will you physically secure the check printer and stock?

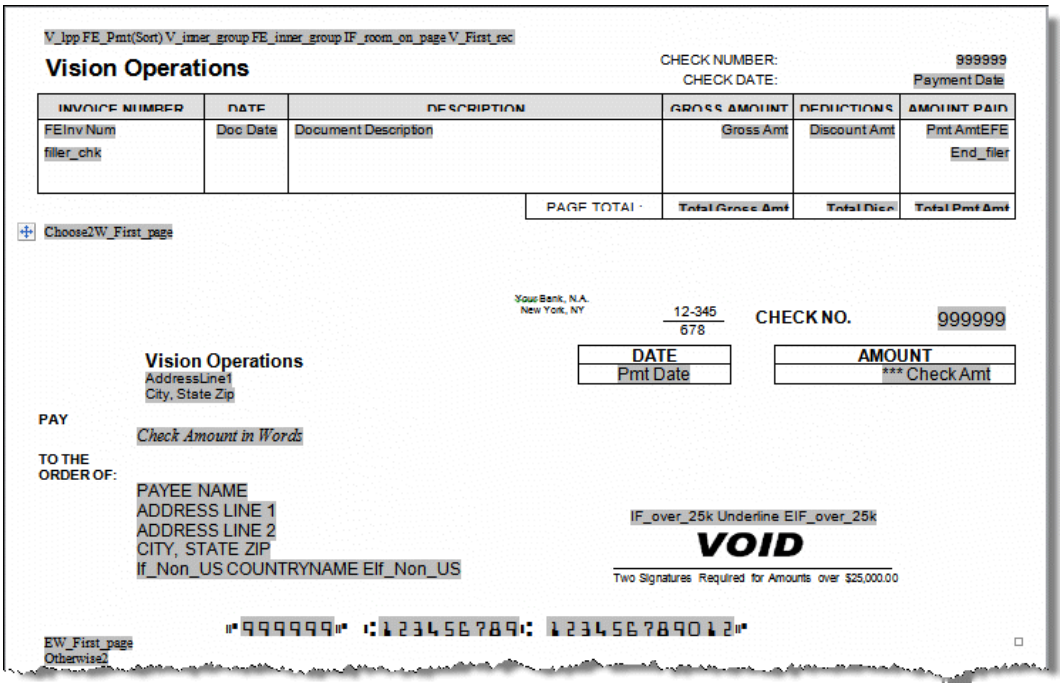
Project Management/Functional – Tips & Lessons Learned

For printed checks, be sure to allow enough time for printer procurement and setup if you don't already have an Oracle-compatible check printer in use. Choosing a model goes by your expected volume, but also gets to your security requirements – whether you need MICR or plain toner, if you'd like a physical key lock on the printer or the stock trays, etc.

And if you'll be printing the MICR character strings from Oracle Payments onto blank check stock, plan for multiple rounds of MICR acceptance testing with the bank. Printing MICR from Oracle is more flexible and secure than going with pre-MICR'ed stock, but the placement requirements have very tight tolerances.

Technical – Overview

For developers, printing on-site checks from Oracle Payments will mean creating an RTF Layout Template in the BI Publisher Desktop plug-in for Microsoft Word. Your Template here will be a formatted version of your check, complete with tables, images, boilerplate text items, etc. Microsoft Word form fields will be used to specify conditional and looping logic, and as placeholders for the field values that will be pulled from your source XML. It may look like the following:



The basic steps are the same as creating an eText template, but here you'll be doing your local previewing and debugging within Microsoft Word. You can preview RTF templates from the separate Template Viewer application, but there is no requirement to leave Word to do so.

Here again, take some time to get to know the features available to you in RTF templates. Chapter 2 of the Oracle XML Publisher Report Designer's Guide ("Creating an RTF Template") will give you all the details, but several that you're likely to need for your on-site printed checks:

- Custom Fonts
- Fixed Row Enumeration
- Conditional Formatting

Custom Fonts

Custom fonts used on printed checks include E-13B (used in the MICR string of US and Canadian checks), OCR-B1 (required for some elements on the face of UK checks), and various barcodes (for postal sorting or other scanning requirements). The steps for adding custom fonts to your RTF template:

- **Obtain the font** – For E-13B, the freely available GnuMICR font will be acceptable to many banks and can be your first choice. Some banks won't accept that font, however, and you'll then need to purchase an E-13B font package from a font supplier (e.g., IDAutomation, Advantage Laser, etc.)
- **Make it accessible locally** – Copy the font to both your operating system and BI Publisher font directories and reference it in an xdo.cfg file to allow you to see the font when previewing your template within Microsoft Word.
- **Make it accessible to Oracle Payments** – When your local development is complete, upload your font to the server and create your font mappings to allow printing directly from Oracle Payments.

See Section 4 ("Administration") and Appendix B ("XML Publisher Configuration File") in the Oracle XML Publisher Administration and Developer's Guide for additional details. Several good entries in Ike Wiggin's and Tim Dexter's blogs also detail the setup in a more step-by-step fashion. See the Reference section for those links.

Fixed Row Enumeration
Fixed row enumeration is a technical workaround for the fact that BI Publisher RTF templates (and the XSL-FO technology they're based on) do not have a way of creating a fixed height table containing a variable number of rows. This is a small step backwards from Oracle Reports, which offered that capability out of the box. For BI Publisher, the logic for the workaround is complicated and involves creating XSL variables and parameterized sub-templates. So it's not the first template type that you'd look to develop as one of your first BI Publisher reports.

But if you're looking to ensure the constant vertical placement of MICR string on a check, no matter how many Invoices are listed on the remittance section above it, there's currently no way of avoid it – you'll need to employ this technique. Rather than detail the steps here, see the "Anatomy of a Template I - Fixed Row Enumeration" entry in Tim Dexter's blog or the "Developing reports printed on Pre-Printed Stationary" entry on Anil Passi's website for the technique.

Conditional Formatting
BI Publisher RTF templates give you the ability to quickly and easily make the appearance of fields, images, and formatting elements conditional upon values within your XML data. This comes in very handy for several security features that you can build into your template.

One common requirement is for displaying a second signature line with the words "Amounts over $XX,XXX require two signatures" if the check amount is over a certain threshold. That way, low-value checks print with a single scanned signature image and can be mailed directly from the printer, while larger checks would require an additional signature from a manager. That manager's signature can be automatic, but many clients prefer it to be manual as an additional control. BI Publisher easily supports making the printing of the second signature line, the additional boilerplate text, and an additional signature image conditional upon the payment amount. The threshold amount can be hard-coded into your template, or stored in a Descriptive Flex Field and passed into the XML via the IBY_FD_EXTRACT_EXT_PUB package.

For digital signatures, the scanned image can be placed directly in your RTF template or pulled from any URL accessible to the format concurrent request program (running on your concurrent manager tier). Placing the signature in the $OA_HTML directory on the applications server allows you to secure the image with standard OS file permission rules. Additional security can be obtained by restricting access to the file via IP address, so that only the application server can access it. The file name to use can also be dynamically constructed by your RTF template, if different checks require different signatures.

As an additional physical control, you can even set it up so that the signature file is on a secured USB drive controlled by an AP manager. That USB drive can be encrypted (with TrueCrypt, for example) and only accessible when the volume is mounted with a password and/or key file. But if the USB drive is inserted into a designated PC (and mounted to a consistent drive letter on that PC), the signature image on that drive can then be mapped to a URL accessible to the format process. If the image URL is not available when payments are formatted, then the RTF template can be set to print a void stamp image instead. Note that this approach will require work (and consent) from your internal PC/networking group and whoever hosts your Oracle instance. So be sure to check with them before attempting to go with this approach.

Finally, if you truly require that your signature images and/or check stock layouts be contained on a SIMM or SD card that must be physically inserted into the printer itself (or a box attached to the printer) before they can be printed, be aware that there's no seeded way in BI Publisher to accomplish that. Controlling those printers or printer boxes to pull images from a SIMM or SD card requires sending Printer Control Language (PCL) codes, while BI Publisher only produces PDF output at this time. So you would either need to:

1. Parse the PDF generated from BI Publisher and convert it to PCL so that you can insert the necessary additional commands to pull the image from the card, or
2. Purchase that functionality from a third-party provider (e.g., Evergreen's DocuGuard®)

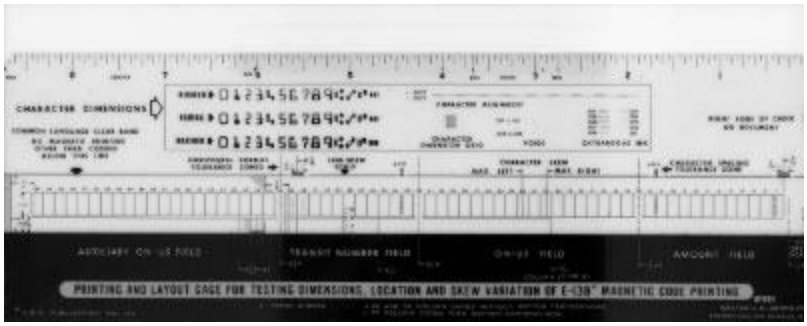
Technical – Tips & Lessons Learned

For printed checks, starting with a model RTF template is a good idea. See the "Standard Form Feed Check Format" (IBYDC_FF1.rtf) or one of the many examples available on the internet. Even if you don't base your template on an existing one, seeing the examples in real code is one of the best ways to learn the more advanced layout options – like conditional formatting, overlaying images, fixed row enumeration, etc.

Printed MICR

If you're going to have your RTF template print the MICR characters on your on-site checks, plan for more development and testing time than if you're using pre-MICR'ed stock. You gain flexibility and security, but aligning the MICR will be tricky. Some helpful tips:

- Get a Glardon gauge – This is a clear plastic guide that you overlay on your printed checks to verify the horizontal and vertical alignment of your MICR string within the clear band area, as well as the spacing between individual MICR characters. It will look something like this:



  Your bank may provide you with one, or they're available from many check-printing supply sites. Either way, do not attempt this process without a MICR gauge on-hand.
- Use Microsoft Word tables for placement – This applies not just to not just the MICR, but all the items on the check face. You get much more precise placement control with adjusting table row height, column widths, and individual cell margins vs. trying to use tabs, spaces, and adjusting the font sizing and line spacing of blank filler rows.
- For your tables in Word, get in the habit of setting your Row Height to "Exactly". The default value of "At Least" can make it very difficult for you to control the vertical placement of all your items, as you won't know which row just increased in size due to some other adjustment you just made.
- Set your default Unit of Measure to Millimeters instead Inches, for more exact MICR placement. This option is available in Word 2010 under the File ribbon by choosing "Options" and then "Advanced". Under the "Display" section is the "Show measurements in units of:" field. In Word 2007, access this field through the "Word Options" button under the large, round Office button in the top left of your screen.

When you're ready to print, either locally or on the server, be sure to check your printer driver settings. You want to be sure that your printer driver PDF printing options specify scaling at 100%, and no automatic shrink to fit. Your PDF image should start consistently in the top left corner of the printable area, with no vertical or horizontal centering. This can easily cause large differences in your MICR placement as you migrate your printing from your development PC to your test instance(s) and production, so be aware of these settings.

Finally, don't waste too much time on getting your MICR placement exact until you're ready (and able) to print to the production check printer. You may have to attach the production printer temporarily to your Oracle test instance to be able to print your MICR acceptance testing checks for the bank. But you want to make sure that not only are your final test checks printed with the same printer drivers and through the same printer servers as you'll be using in production, but that they come out of the same physical printer that you'll be using after go-live.

Experience has shown that the vertical alignment of a printed PDF image does vary sometimes between printers, even between two printers of the same exact model. That variation can be significant, when measured against the

tight tolerances of the MICR placement requirements. Be aware, too, that the vertical alignment in printers may drift over time, as the internal gears and rollers wear through use. So center your MICR vertically in the middle of the gauge boxes, to allow for some drift up or down over time without going out of spec.

Pre-MICR'ed stock
Buying and using check stock with the check number and MICR string pre-printed may be less secure and flexible than using completely blank stock, but it certainly makes the RTF Template development process easier. Many clients feel it is secure enough for their requirements, and/or they have existing check stock on hand that they'd like to continue to use when they switch over or upgrade to Oracle Payments.

From a development stand point, you can code your pre-printed Layout Template using nested tables. You can have a single row, fixed-height table for your entire remittance section (i.e., the stub above or below the check), and then a multi-row table within that single row for all your Invoice details. Size your Invoice font and spacing and set your fields so that no data wraps within these cells. That determines the row height in your multi-row table. Then just determine how many rows of Invoices will fit within your fixed-height remittance section.

From there, set up your Payment Document with a "Paper Stock Type" value of "Prenumbered Stock" and the "Number of Lines per Remittance Stub" field set to the number of Invoice rows that you've determined will fit. Seeded Oracle Payments functionality will then take care of generating non-negotiable overflow checks for you, any time a given payment pays more than that number of Invoices.

Testing
For checks where you'll be printing the MICR, you bank will specify how many samples that they'll need you to mail them for their MICR acceptance testing. You should consider that your minimum. For either those or pre-MICR'ed checks, you'll want to make sure you test all your data conditions – Suppliers with varying numbers of address lines, very long address lines, and foreign addresses as applicable. Be sure everything is visible within the window of the mailing envelopes you'll be using.

For amounts, test checks that are under, over and exactly at any thresholds you define for conditional formatting. Also test very large amounts, to ensure the digits fit in the pay amount space allotted. This also often serves to test checks where the payment amount in words is very long. For most checks, this field will need to be able to wrap around within its table cell without affecting the placement of the payee address and MICR string below it.

Finally, be sure to test payments of varying numbers of Invoices per check, including a large enough number of Invoices to force one or more overflow documents. This is especially important on checks where you're printing the MICR, since you'll also need to code the logic for those overflow documents yourself. It also tests that your fixed row enumeration is correct, and the vertical alignment of the MICR doesn't vary with the number of Invoices paid.

Summary

While the new concepts and terminology may be confusing at first, the effort you put into learning Oracle Payments Rel. 12 can pay big dividends in terms of business benefits for your organization. It is a powerful engine for handling all your organization's Funds Disbursement (and Funds Capture) needs. Hopefully this paper has helped provide you with the overview you need, to allow you to implement or upgrade to the Rel. 12 Payment module and integrate with your bank(s) for automated electronic payments and/or securely print on-site AP checks.

About the Author

Glen T. Ryen is the Vice President of Technology at Prisio Technologies. He has over 15 years of experience in implementing, integrating, and customizing the Oracle E-Business Suite, and four years of experience specifically with Rel. 12 Oracle Payments. He has helped numerous Fortune 500 customers in various industries with their Oracle Applications needs, in technical, functional, and management capacities. He can be reached at GlenR@prisiotechnologies.com.

References & Useful Links

R12 Payments

1. MOS Note #1391460.1 – "R12: Understanding What the Oracle EBS "Payments" (IBY) module Is, and What It Does"
2. Oracle® Payments Implementation Guide Release 12.1 – Part No. E13416-04
3. Oracle® Payments User's Guide Release 12.1 – Part No. E13415-04
4. http://www.norcaloaug.com/seminar_archive/2008_training_day_pres/5_01_anderson%20.ppt – "Payment Processing with Oracle Payments: What's New in Oracle E-Business Suite Release 12" by Victoria Anderson (Manager, Payment Product Management)
5. "How to Setup and Extend R12 Payments" by Chris Thom (Applications Analyst, Johnson County Government), OAUG Collaborate 2011.
6. MOS Notes:
    a. #733537.1 – "R12 Upgrade: Functional Upgrade Impacts Document for Oracle Payments (FINANCIALS)"
    b. #579132.1 – "R12 Oracle Payments Processing 'How To' documents"
    c. #821133.1 – "R12 Payment Process Request - Functional and Technical Information"

BI Publisher General Links (Oracle Sites)

1. Patch 12395372 - Update For BI Publisher Desktop 10.1.3.2.1 (5.6.3)
2. Oracle XML Publisher Report Designer's Guide – Part No. B31410-01
3. Oracle XML Publisher Administration and Developer's Guide – Part No. B31412-01
4. Overview of Available Patches for Oracle XML Publisher embedded in the Oracle E-Business Suite
5. MOS Notes:
    a. #364547.1 – "Troubleshooting Oracle XML Publisher For The Oracle E-Business Suite"
    b. #404928.1 – "Oracle XML Publisher Release 12 Known Issues"
6. http://www.oracle.com/technetwork/middleware/bi-publisher/overview/index.htm – Oracle Technology Network BI Publisher Overview, with links to documentation index, downloads, and community
7. http://forums.oracle.com/forums/forum.jspa?forumID=245 – Oracle Technology Network forum
8. http://blogs.oracle.com/xmlpublisher – Tim Dexter's blog
9. http://blogs.oracle.com/BIDeveloper – A BI Publisher developer's diary, by Ashish Shrivastava
10. http://bipconsulting.blogspot.com – Oracle BI Publisher Consulting blog, by Kan Nishida

BI Publisher General Links (Non-Oracle Sites)

1. http://bipublisher.blogspot.com - Ike Wiggin's BI Publisher Blog
2. http://garethroberts.blogspot.com - In Depth Apps blog by Gareth Roberts
3. http://www.strsoftware.com/bi-publisher-university/ - STR Software's BI Publisher University
4. "XML Publisher In EBS For Newbies" by Srini Chavali (Cummins Inc.), OAUG Collaborate 2011.

BI Publisher for Check Printing

1. MOS Note #312353.1 – "Check printing using XML Publisher"
2. http://oracle.anilpassi.com/xml-publisher-developing-reports-printed-on-pre-printed-stationary-2.html – A step-by-step example of fixed row enumeration in an RTF template.
3. https://blogs.oracle.com/xmlpublisher/entry/hard_check_print_security_alte – USB drive signature storage
4. https://blogs.oracle.com/xmlpublisher/entry/conditional_check_signature_im – Conditional images
5. https://blogs.oracle.com/xmlpublisher/entry/hard_check_printing – An overview of Evergreen's DocuGuard solution for PDF to PCL conversion and signature image storage on a SIMM/SD card.
6. http://www.idautomation.com/fonts/micr/ – MICR fonts for purchase.
7. http://www.idautomation.com/oracle/xml_publisher_barcode.html – Creating Barcodes in XML Publisher.

8. http://www.advlaser.com/MICR-Fonts-s/8.htm – Printers, blank stock, MICR toner, MICR font, and Glardon gauges for purchase.

XML, XSL, and XPath

1. http://www.ibm.com/developerworks/opensource/library/x-xmltools/index.html?ca=drs-, http://www.xml.com/pub/pt/3 – XML tool options and reviews
2. http://www.ibm.com/developerworks/xml/newto/ – Resources for learning XML
3. http://www.w3.org/TR/xpath/ – Resources for learning XPath expressions
4. http://www.w3.org/Style/XSL/ – The Extensible Stylesheet Language Family

Useful Scripts

Payment Templates – Details of Payment Templates defined:

```
SELECT apt.template_id, apt.template_name,
    ipmv.payment_method_name pmt_method_name,
    cba.bank_account_name bank_acct_name,
    ipp.payment_profile_name pmt_profile_name,
    cpd.payment_document_name pmt_doc_name, apt.pay_group_option pay_grp_optn,
    apt.ou_group_option ou_grp_optn, apt.currency_group_option curr_grp_optn,
    apt.description ppp_description, apt.inactive_date, apt.addl_pay_thru_days,
    apt.addl_pay_from_days, apt.low_payment_priority, apt.hi_payment_priority,
    apt.vendor_id, apt.pay_only_when_due_flag,
    apt.vendor_type_lookup_code vdr_type_lcode, apt.bank_account_id,
    apt.payment_profile_id, apt.zero_inv_allowed_flag, apt.payment_method_code,
    apt.inv_exchange_rate_type, apt.payment_date_option, apt.addl_payment_days,
    apt.payment_exchange_rate_type, apt.zero_amounts_allowed,
    apt.payables_review_settings, apt.calc_awt_int_flag, apt.payments_review_settings,
    apt.document_rejection_level_code doc_reject_lvl, apt.create_instrs_flag,
    apt.payment_rejection_level_code pmt_reject_lvl, apt.payment_document_id,
    plc.displayed_field supplier_type, pv.vendor_name payee,
    alc1.displayed_field template_type_name, gdct.user_conversion_type user_rate_type,
    fu.user_name
  FROM ap_payment_templates apt, po_lookup_codes plc, iby_payment_methods_vl ipmv,
    iby_payment_profiles ipp, ce_bank_accounts cba, ap_lookup_codes alc1,
    gl_daily_conversion_types gdct, po_vendors pv, fnd_user fu, ce_payment_documents cpd
 WHERE     fu.user_id = apt.last_updated_by
    AND plc.lookup_code(+) = apt.vendor_type_lookup_code
    AND plc.lookup_type(+) = 'VENDOR TYPE'
    AND cba.bank_account_id(+) = apt.bank_Account_id
    AND ipmv.payment_method_code(+) = apt.payment_method_code
    AND alc1.lookup_type(+) = 'PAYMENT_TEMPLATE_TYPE'
    AND alc1.lookup_code(+) = apt.template_type
    AND gdct.conversion_type(+) = apt.payment_exchange_rate_type
    AND ipp.payment_profile_id(+) = apt.payment_profile_id
    AND pv.party_id(+) = apt.party_id
    AND apt.payment_document_id = cpd.payment_document_id(+)
    -- Uncomment to limit to specific Payment Templates
    -- AND apt.TEMPLATE_NAME LIKE 'J%'
ORDER BY apt.template_name, ipmv.payment_method_name;
```

Payment Process Profiles – Verification of your PPP setup:

```
SELECT app.payment_profile_name, ppp.system_profile_code, ppp.payment_format_code,
       pm.payment_method_name, ppp.periodic_sequence_name_1 seq_name,
       app.payment_profile_id pmt_prof, app.reset_value_1 reset_val,
       app.last_used_number_1 last_used, ppp.processing_type,
       ppp.mark_complete_event mrk_cmplt, ppp.outbound_pmt_file_directory,
       ppp.outbound_pmt_file_prefix, ppp.outbound_pmt_file_extension ext,
       ppp.positive_pay_file_directory, ppp.positive_pay_file_prefix,
       ppp.positive_pay_file_extension pp_ext, ppp.default_payment_document_id def_pmt_docid,
       ppp.last_update_date, sra.sra_override_payee_flag sra_ovrd_payee,
       fmt.format_name sra_fmt, sra.automatic_sra_submit_flag sra_autosub
  FROM iby_sys_pmt_profiles_b ppp, iby_acct_pmt_profiles_vl app,
       iby_applicable_pmt_profs appl_prof, iby_payment_methods_vl pm,
       iby_remit_advice_setup sra, iby_formats_vl fmt
 WHERE     1 = 1
       AND ppp.system_profile_code = app.system_profile_code
       AND ppp.system_profile_code = appl_prof.system_profile_code(+)
       AND appl_prof.applicable_value_to = pm.payment_method_code(+)
       AND NVL (appl_prof.applicable_type_code, 'PAYMENT_METHOD') = 'PAYMENT_METHOD'
       AND ppp.system_profile_code = sra.system_profile_code(+)
       AND sra.remittance_advice_format_code = fmt.format_code(+)
       -- Only custom PPP's, if this is your naming convention
       AND ppp.system_profile_code LIKE 'XX%';
```

Payment Documents – Setup data, by internal disbursing Bank Account:

```
 SELECT acct.bank_account_name, acct.bank_account_num, acct.currency_code,
        acct.multi_currency_allowed_flag multi_curr, acct.payment_multi_currency_flag pmt_mc,
        pmtdoc.payment_document_name pmt_doc_name, fmt.format_name,
        pmtdoc.first_available_document_num first_avail_docno,
        pmtdoc.last_available_document_number last_avail_docno,
        pmtdoc.last_issued_document_number last_issued_docno,
        pmtdoc.payment_document_id pmt_doc_id, pmtdoc.internal_bank_account_id,
        pmtdoc.paper_stock_type, pmtdoc.attached_remittance_stub_flag attchd_stub,
        pmtdoc.number_of_lines_per_remit_stub lines_per_stub, pmtdoc.format_code,
        pmtdoc.inactive_date, fu1.user_name created_by, pmtdoc.creation_date,
        fu2.user_name last_updated_by, pmtdoc.last_update_date,
        DECODE (inactive_date, NULL, 'Y', 'N') status
   FROM ce_payment_documents pmtdoc, iby_formats_vl fmt, ce_bank_accounts acct,
        fnd_user fu1, fnd_user fu2
  WHERE     pmtdoc.format_code = fmt.format_code
        AND pmtdoc.internal_bank_account_id = acct.bank_account_id
        AND pmtdoc.created_by = fu1.user_id
        AND pmtdoc.last_updated_by = fu2.user_id
        -- Uncomment to limit to a certain document name
        -- AND payment_document_name LIKE '%'
ORDER BY acct.bank_account_name, pmtdoc.payment_document_name;
```


Layout Templates - Metadata and RTF/eText files for all Outbound Payment Instruction Layout Templates:

```
SELECT xtv.application_short_name tmpl_app, xtv.template_code,
       xtv.ds_app_short_name ds_app, xtv.data_source_code, xtv.template_type_code tmpl_type,
       xtv.default_language def_lang, xtv.default_territory def_terr, xtv.template_status,
       xtv.start_date, xtv.end_date, xtv.template_name,  xtv.description, xtv.created_by,
       xtv.creation_date, xtv.last_updated_by, xtv.last_update_date, xtv.last_update_login,
       (SELECT application_name
          FROM fnd_application_vl
         WHERE application_short_name = xtv.application_short_name) application_name,
       (SELECT meaning
          FROM fnd_lookups
         WHERE lookup_type = 'XDO_TEMPLATE_TYPE'
               AND lookup_code = xtv.template_type_code) template_type,
       (SELECT data_source_name
          FROM xdo_ds_definitions_vl
         WHERE data_source_code = xtv.data_source_code
               AND application_short_name = xtv.ds_app_short_name) data_source_name,
       (SELECT file_name
          FROM xdo_lobs
         WHERE ( (    lob_type = 'TEMPLATE'
                  AND xdo_file_type != 'RTF'
                  AND xdo_file_type = xtv.template_type_code
                  AND xdo_lobs.LANGUAGE = xtv.default_language
                  AND xdo_lobs.territory = xtv.default_territory)
                OR (    lob_type = 'TEMPLATE_SOURCE'
                    AND xdo_file_type IN ('RTF', 'RTF-ETEXT')
                    AND xdo_lobs.LANGUAGE = xtv.default_language
                    AND xdo_lobs.territory = xtv.default_territory)
                OR (    xdo_file_type = 'RTF'
                    AND lob_type = 'TEMPLATE_SOURCE'
                    AND LANGUAGE = xtv.mls_language
                    AND territory = xtv.mls_territory
                    AND EXISTS
                           (SELECT mls.lob_code
                              FROM xdo_lobs mls
                             WHERE mls.lob_type = 'MLS_TEMPLATE'
                                   AND mls.lob_code = xtv.template_code
                                   AND mls.application_short_name = xtv.application_short_name
                                   AND mls.LANGUAGE = xtv.default_language
                                   AND mls.territory = xtv.default_territory)
                    AND NOT EXISTS
                              (SELECT LOCAL.lob_code
                                 FROM xdo_lobs LOCAL
                                WHERE LOCAL.lob_type = 'TEMPLATE_SOURCE'
                                      AND LOCAL.lob_code = xtv.template_code
                                      AND LOCAL.application_short_name = xtv.application_short_name
                                      AND LOCAL.LANGUAGE = xtv.default_language
                                      AND LOCAL.territory = xtv.default_territory)))
               AND lob_code = xtv.template_code
               AND xdo_lobs.application_short_name = xtv.application_short_name)
         AS default_template_file,
       (SELECT file_name
```

```
          FROM xdo_lobs
         WHERE lob_type = 'TEMPLATE_SOURCE' AND lob_code = xtv.template_code
               AND xdo_lobs.application_short_name =xtv.application_short_name
               AND xdo_lobs.LANGUAGE = xtv.mls_language
               AND xdo_lobs.territory = xtv.mls_territory) AS mls_template_file,
       (SELECT NAME
          FROM fnd_iso_languages_vl
         WHERE iso_language_2 = xtv.default_language) AS default_file_lang,
       DECODE (xtv.default_territory, '00', '', ftv.territory_short_name) AS default_file_terr,
       xtv.mls_language, xtv.mls_territory, xtv.default_output_type
  FROM xdo_templates_vl xtv, fnd_application_vl fav, fnd_territories_vl ftv
 WHERE     1 = 1
       AND fav.application_short_name = xtv.application_short_name
       AND ftv.territory_code(+) = xtv.default_territory
       -- Only Outbound Payment Instructions
       AND xtv.application_short_name = 'IBY'
       AND data_source_code = 'IBY_FD_INSTRUCTION_1_0';
```

PPR and Source XML – Payment Process Requests formatted today, with the source XML:

```
SELECT trxnmid,
       EXTRACTVALUE (
         XMLType (document),
         '/OutboundPaymentInstruction/PaymentInstructionInfo/UserAssignedRefCode') ppr_name,
       EXTRACTVALUE (
         XMLType (document),
         '/OutboundPaymentInstruction/PaymentInstructionInfo/FormatProgramRequestID')
         fmt_request_id,
       EXTRACTVALUE (
         XMLType (document),
         '/OutboundPaymentInstruction/PaymentProcessProfile/PaymentProcessProfileName')
         ppp_name,
       EXTRACTVALUE (
         XMLType (document),
         '/OutboundPaymentInstruction/CheckFormatInfo/PaymentDocumentName')
         pmt_doc_name,
       EXTRACTVALUE (
         XMLType (document),
         '/OutboundPaymentInstruction/InstructionTotals/PaymentCount') pmt_cnt,
       document raw_xml
  FROM iby_trxn_documents itd
 WHERE 1 = 1
   AND itd.creation_date > TRUNC (SYSDATE);
```

Abbreviations Used

ABA	American Bankers Association
ACH	Automated Clearing House
AP	Accounts Payables
AR	Accounts Receivables
BIC	Bank Identifier Code
BIP	Business Intelligence Publisher
BPEL	Business Process Execution Language
CCD	Corporate Credit or Debit
CHIPS	Clearing House Inter-Payments System
CTX	Corporate Trade Exchange
DMZ	Demilitarized Zone
ERP	Enterprise Resource Planning
FTPS	File Transfer Protocol Secure
GPG	GNU Privacy Guard
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IBAN	International Bank Account Number
ISO	International Organization for Standardization
MICR	Magnetic Ink Character Recognition
MOS	My Oracle Support
NACHA	National Automated Clearing House Association
PCL	Printer Control Language
PDF	Portable Document Format
PGP	Pretty Good Privacy
PPD	Prearranged Payment and Deposits
PPR	Payment Process Request
PPP	Payment Process Profiles
RTF	Rich Text Format
SD	Secure Digital
SEPA	Single European Payment Area
SIMM	Single Inline Memory Module
SOX	Sarbanes-Oxley
SSL	Secure Sockets Layer
SWIFT	Society for Worldwide Interbank Financial Telecommunication
TCA	Trading Community Architecture
URL	Uniform Resource Locator
USB	Universal Serial Bus
VPN	Virtual Private Network
XML	Extensible Markup Language
XMLP	XML Publisher, former name of BI Publisher
XSL	Extensible Stylesheet Language
XSL-FO	XSL Formatting Objects
XSLT	XSL Transformations