

TIME TO COLLABORATE

APRIL 19-23, 2020
LAS VEGAS, NV
#C20LV

[ATTENDCOLLABORATE.COM](https://attendcollaborate.com)



COLLABORATE 20
TECHNOLOGY & APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

OATUG
ORACLE APPLICATIONS & TECHNOLOGY USERS GROUP



Quest



COLLABORATE 20

TECHNOLOGY & APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

BI Publisher Unleashed

Practical and Innovative
Techniques for the Everyday

Session ID:

11607

Prepared by:

Joe Tseng

O2Works, LLC

Monday, April 20, 2020

Remember to complete your evaluation for this session within the app!

[COLLABORATE.OATUG.ORG](https://collaborate.oatug.org)

#C20LV

Agenda

- Introduction
- Technique #1 – Process Wrapping
- Technique #2 – Notifications and Flat File Generation
- Technique #3 – Dynamic Boiler-plating
- Technique #4 – OAF Extension Reporting
- Technique #5 – Bursting to XML

Introductions

- Joe Tseng
 - Technical EBS Consultant, O2Works, LLC
 - Over 25 years technical implementation experience in Oracle EBS
 - Contact Information: jtseng@o2works.com

About O2Works

- **O2Works** is one of the leading E-Business Suite services providers offering the most experienced teams of functional and technical consultants in the industry. Our hands-on **resources average 19+ years of experience** focused exclusively on implementing, upgrading, integrating, and extending Oracle's E-Business Suite.



Technique #1 – Process Wrapping



COLLABORATE 20

TECHNOLOGY & APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

OATUG

ORACLE APPLICATIONS & TECHNOLOGY USERS GROUP

Technique #1 – Process Wrapping

- One of the most common misunderstandings of BI Publisher is that it is just a reporting tool. BI Publisher can be used for so much more.
- BI Publisher can be used to “wrap” entire processes, encapsulating functionality in one tidy “package” of code.
 - Use of before and after report triggers tied to PLSQL packages can handle almost all processing requirements
 - Process Exception reporting can be easily supported through use of a simple data templates and formatting templates
 - Notifications can all be handled by BI Publisher bursting

Technique #1 – Process Wrapping

- Using BI Publisher to “wrap” processes can lead to a smaller code maintenance “signature”
 - No longer a need for separate concurrent processes and executables
 - Concurrent processes are instead setup as programs that use the XDODTEXE executable

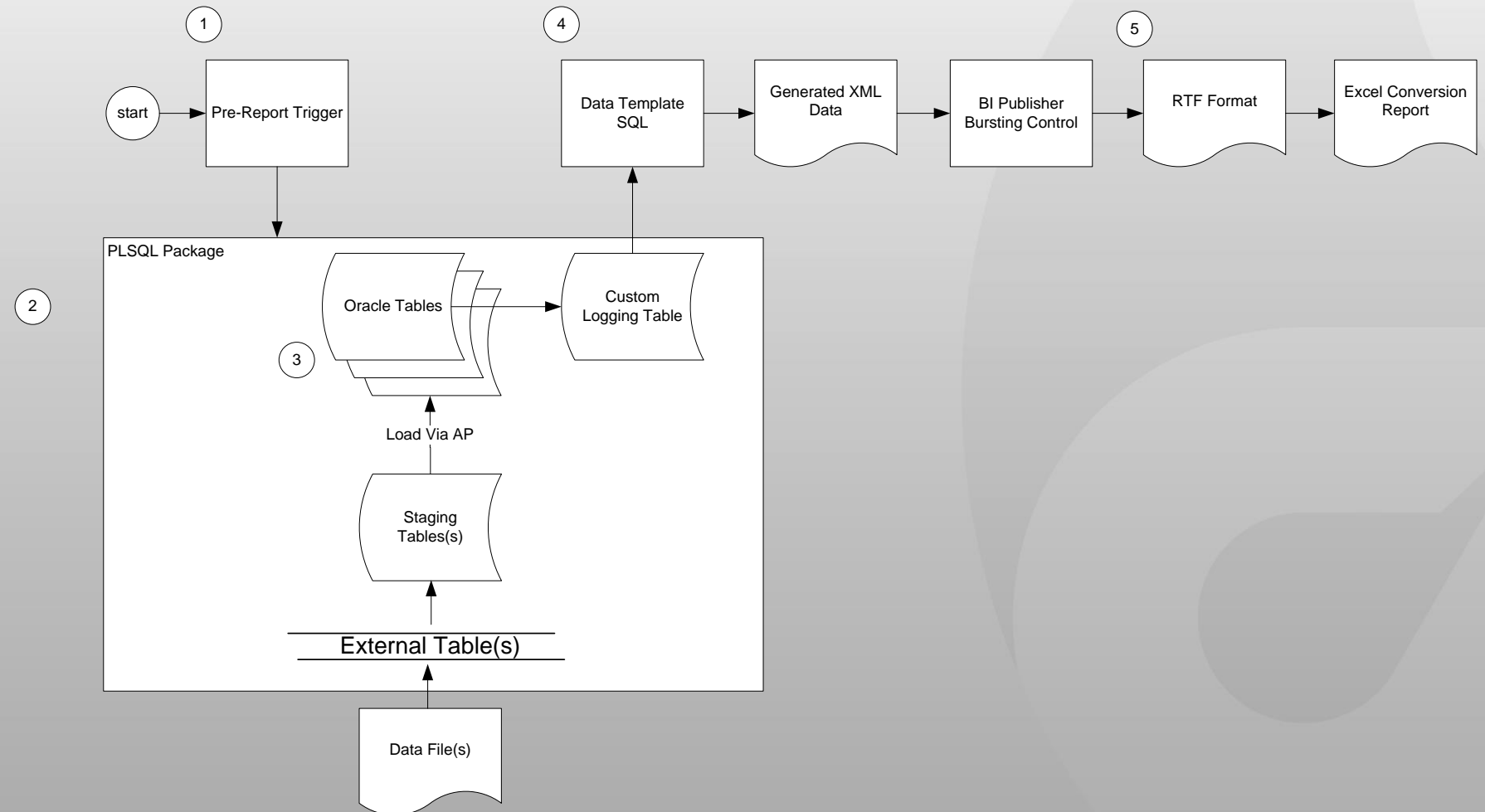
The screenshot shows the 'Concurrent Programs' configuration window. The 'Program' field is set to 'Italian Electronic Invoice' and is checked as 'Enabled'. The 'Short Name' is '_AR_IT_EINV', the 'Application' is 'Custom Application', and the 'Description' is 'Italian Electronic Invoices Generation'. Under the 'Executable' section, the 'Name' is 'XDODTEXE' and the 'Method' is 'Java Concurrent Program'. The 'Request' section includes fields for 'Type', 'Incrementor', 'MLS Function', and 'Operating Unit Mode' (set to 'M'). There are several checkboxes: 'Use in SRS' (checked), 'Run Alone' (unchecked), 'Enable Trace' (unchecked), 'Recalculate Default Parameters' (unchecked), 'Allow Disabled Values' (unchecked), 'Restart on System Failure' (checked), and 'NLS Compliant' (checked). The 'Output' section has a 'Format' dropdown set to 'XML', with 'Save (S)' and 'Print' checked. There are also fields for 'Columns', 'Rows', 'Style', and 'Printer'. The 'Business Events' section at the bottom has checkboxes for 'Request Submitted (Y)', 'Request On Hold', 'Request Resumed', 'Request Running', 'Program Completed', 'Post Processing Started', 'Post Processing Ended', and 'Request Completed (Z)'. At the bottom of the window are buttons for 'Copy to...', 'Session Control', 'Incompatibilities', and 'Parameters (G)'.

Technique #1 – Process Wrapping

- Data Conversion Processes
 - A before report trigger can be used to call PLSQL that loads data via Oracle API
 - Autonomous writes to staging or logging tables and be populated and used for reporting conversion results back to the user
 - Multi-threaded processing can be easily supported by using the before report function as a control procedure

Technique #1 – Process Wrapping

- Data Conversion

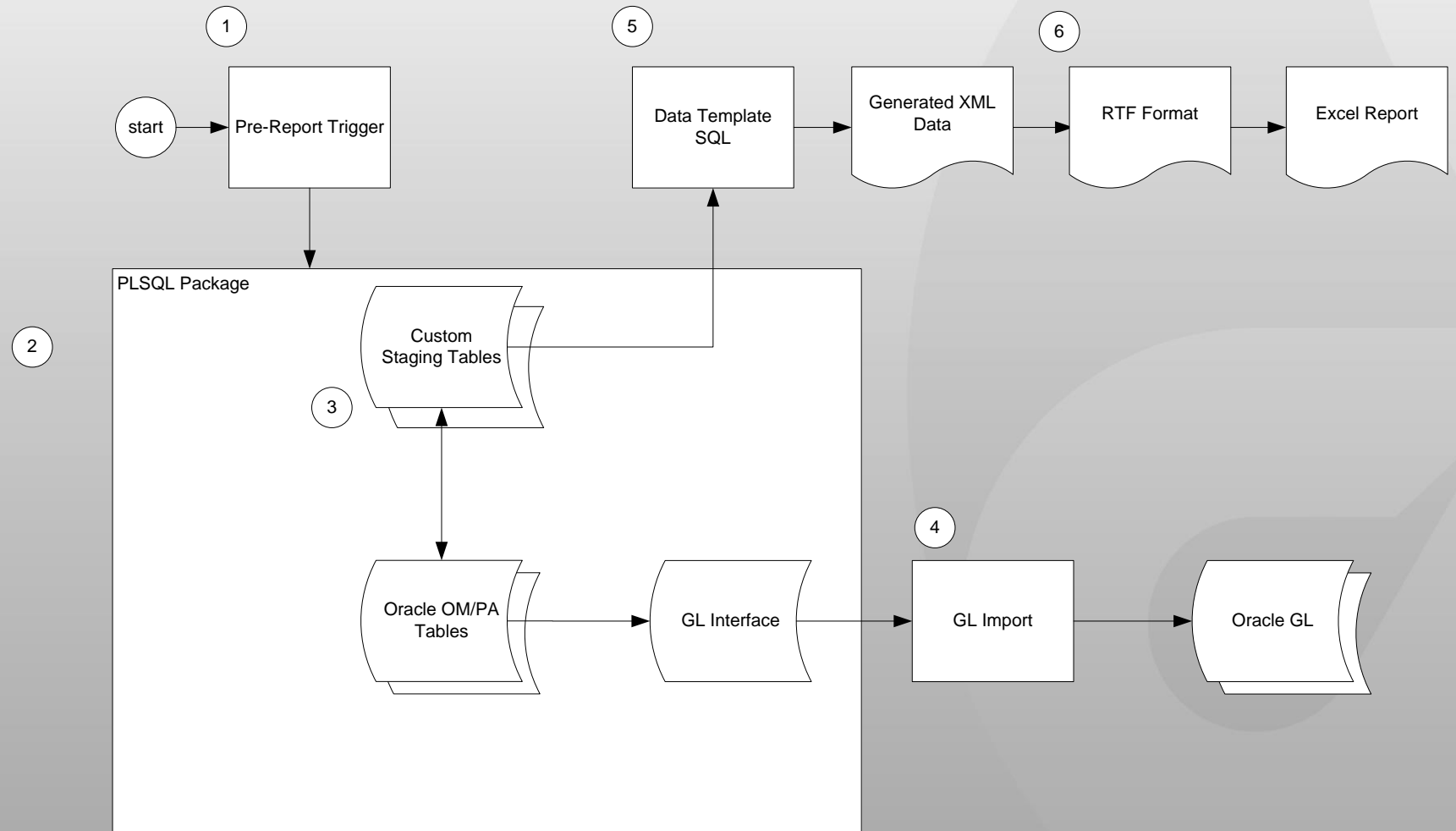


Technique #1 – Process Wrapping

- Transaction Load processes
 - Before report trigger can be used to call PLSQL that loads external data into Oracle interface tables or into Oracle via APIs.
 - Autonomous writes to staging or logging tables and be populated and used for reporting conversion results back to the user
 - Post-process response files can easily be accommodated through BI Publisher bursting
 - Exception notifications can also be accommodated through BI Publisher bursting

Technique #1 – Process Wrapping

- Transaction Processing Example



Technique #1 – Process Wrapping

- DON'T
 - Always create new concurrent program executables for everything
 - Re-invent the wheel and write your own notification engine for exceptions
- DO
 - Use BI Publisher to “wrap” your processes
 - Use BI Publisher to generate well formed error reports instead of relying on concurrent manager log files
 - Use BI Publisher to expand exception notification processing in critical processes

Technique #2 – Notifications and Flat File Generation



COLLABORATE 20

TECHNOLOGY & APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

OATUG
ORACLE APPLICATIONS & TECHNOLOGY USERS GROUP

Technique #2 – Notifications and Flat File Generation

- A common requirement found almost everywhere is the need to send email notifications or generate flat files. Too often, developers actually code the functionality for these requirements into PLSQL code, rather than using the inherent capabilities of BI Publisher
 - Developers will use UTL_FILE or FND_FILE to generate flat files specific to the business purpose.
 - To send email, developers will use UTL_SMTP to send out emails
- Coding these functions into database code can lead to excessive and unnecessary code maintenance issues.

Technique #2 – Notifications and Flat File Generation

- Flat File Example – Typical Code

```
--added as part of V1.5 (end)
v_extract_file_line_c :=
    c_asbank_ext_rec.check_number
    || '|'
    || c_asbank_ext_rec.ul_bank_code
    || '|'
    || '8101'
    || '|'
    || -- UL Bank Brach Code for CCIC
    || c_asbank_ext_rec.ul_bank_account_num
    || '|'
    || c_asbank_ext_rec.benf_bank_account_num
    || '|'
    || --c_asbank_ext_rec.BENF_BANK_CODE || '|' || --Commented as part of V1.5
    || c_asbank_ext_rec.benf_bank_code_cn
    || '|'
    || -- Added as part of V1.5
    || -- c_asbank_ext_rec.BENF_BANK_BRANCH_CODE || '|' || --Commented as part of V1.5
    || c_asbank_ext_rec.benf_bank_branch_code_cn
    || '|'
    || -- Added as part of V1.5
    || c_asbank_ext_rec.benf_bank_name
    || '|'
    || c_asbank_ext_rec.benf_name
    || '|'
    || c_asbank_ext_rec.city
    || '|'
    || c_asbank_ext_rec.payment_currency
    || '|'
    || c_asbank_ext_rec.payment_amount
    || '|'
    || c_asbank_ext_rec.ul_bank_account_num
    || '|'
    || c_asbank_ext_rec.pay_date
    || '|'
    || --ltrim(V_INVOICE_NUM) || '|' || --Commented as part of V1.5
    || LTRIM (v_invoice_num)
    || '|'
    || '0'
    || '|'
    || -- Priority
    || v_ct_email_account
    || '|'
    || --added as part of V1.1
```

Technique #2 – Notifications and Flat File Generation

- Email Example – Typical Code

```
/*To send the notification if the credit hold is released manually or automatically*/
IF ( p_release_mode = 'AUTOMATIC'
    OR p_release_mode = 'Credit Check Failure'
) AND (v_recipient IS NOT NULL)
THEN

    fnd_file.put_line
        (fnd_file.OUTPUT,
         '||RPAD(p_order_number,50)||RPAD(v_recipient,50)
         ');

    v_mail_conn := utl_smtp.open_connection (v_mail_host, 25);
    utl_smtp.helo (v_mail_conn, v_mail_host);
    utl_smtp.mail (v_mail_conn, v_from);
    utl_smtp.rcpt (v_mail_conn, v_recipient);
    v_subject := 'Order Hold Release Notification for Order# '||p_order_number;

    SELECT 'Hi,'
           || CHR(10)
           || CHR(10)
           || 'Following '
           || DECODE(p_no_of_holds,1,'Hold ','Holds ')
           || 'from Order # '
           || p_order_number
           || ' for customer - '
           || l_party_name
           || ' ( Account# '
           || l_account_number
           || ' )'
           || DECODE(p_no_of_holds,1,' has been ',' have been ')
           || 'released.'
           || CHR(10)
           || p_hold_name
           || CHR(10)
           || CHR(10)
           || 'This is an auto-generated email, please do not reply to this email.'
    INTO l_body
    FROM DUAL;

    UTL_SMTP.DATA (v_mail_conn,
                  'Date: '
                  || TO_CHAR (SYSDATE, 'Dy, DD Mon YYYY hh24:mi:ss')
                  || crlf
                  || 'From: '

```

Technique #2 – Notifications and Flat File Generation

- Use BI Publisher to simplify flat file generation and eliminate the need for unmanageable code
 - Put your SQL in a simple Data Template
 - Create an eText RTF formatting template
 - Burst the output to a designated location using a bursting control file

Technique #2 – Notifications and Flat File Generation

- Flat File Generation – Data Template

```
<sqlStatement name="Q_data">
<![CDATA[
SELECT db.name                               instance,
       a.period_name                         period_name,
       c.name                               ledger_name,
       b.segment1
       || '.' || b.segment2
       || '.' || b.segment3
       || '.' || b.segment4
       || '.' || b.segment5
       || '.' || b.segment6
       || '.' || b.segment7
       || '.' || b.segment8
       || '.' || b.segment9
       a.currency_code                       acct,
       a.currency_code                       currency_code,
       a.actual_flag                         actual_flag,
       a.translated_flag                     translated_flag,
       c.currency_code                       ledger_currency,
       NVL(a.begin_balance_dr,0) + nvl(a.period_net_dr,0) ending_dr,
       NVL(a.begin_balance_cr,0) + nvl(a.period_net_cr,0) ending_cr,
       ( NVL(a.begin_balance_dr,0) + nvl(a.period_net_dr,0) )
       -
       ( NVL(a.begin_balance_cr,0) + nvl(a.period_net_cr,0) ) ending_bal
FROM gl_balances a,
     gl_code_combinations b,
     gl_ledgers c,
     v$database db
WHERE a.code_combination_id = b.code_combination_id
AND a.ledger_id = c.ledger_id
AND a.period_name = :P_PERIOD_NAME
AND a.translated_flag IS NULL
AND c.name != 'UL Consolidated'
AND a.currency_code != 'STAT'
AND a.template_id IS NULL
--AND b.segment1 = '140'                -- FOR TESTING
--AND b.segment2 = '62040'             -- FOR TESTING
]]>
</sqlStatement>
</dataQuery>
<dataStructure>
<group name="G_file" source="Q_data">
  <element name="instance" value="instance"/>
  <group name="G_data" source="Q_data">
    <element name="period_name" value="period_name"/>
    <element name="ledger_name" value="ledger_name"/>
    <element name="acct" value="acct"/>
    <element name="currency_code" value="currency_code"/>
    <element name="actual_flag" value="actual_flag"/>
    <element name="translated_flag" value="translated_flag"/>
    <element name="ledger_currency" value="ledger_currency"/>
    <element name="ending_dr" value="ending_dr"/>
    <element name="ending_cr" value="ending_cr"/>
    <element name="ending_bal" value="ending_bal"/>
  </group>
</group>
</dataStructure>
```

Technique #2 – Notifications and Flat File Generation

- Flat File Generation – eText Format

ViewsImmersiveShowZoomWindow

12345678910

XDO file name:AR_REPRINT_FILE_GEN.rtfMapping of Payment Format:AR CF REPRINT FORMATDate: 11/12/2019

Format Setup:

Hint: Define formatting options...

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

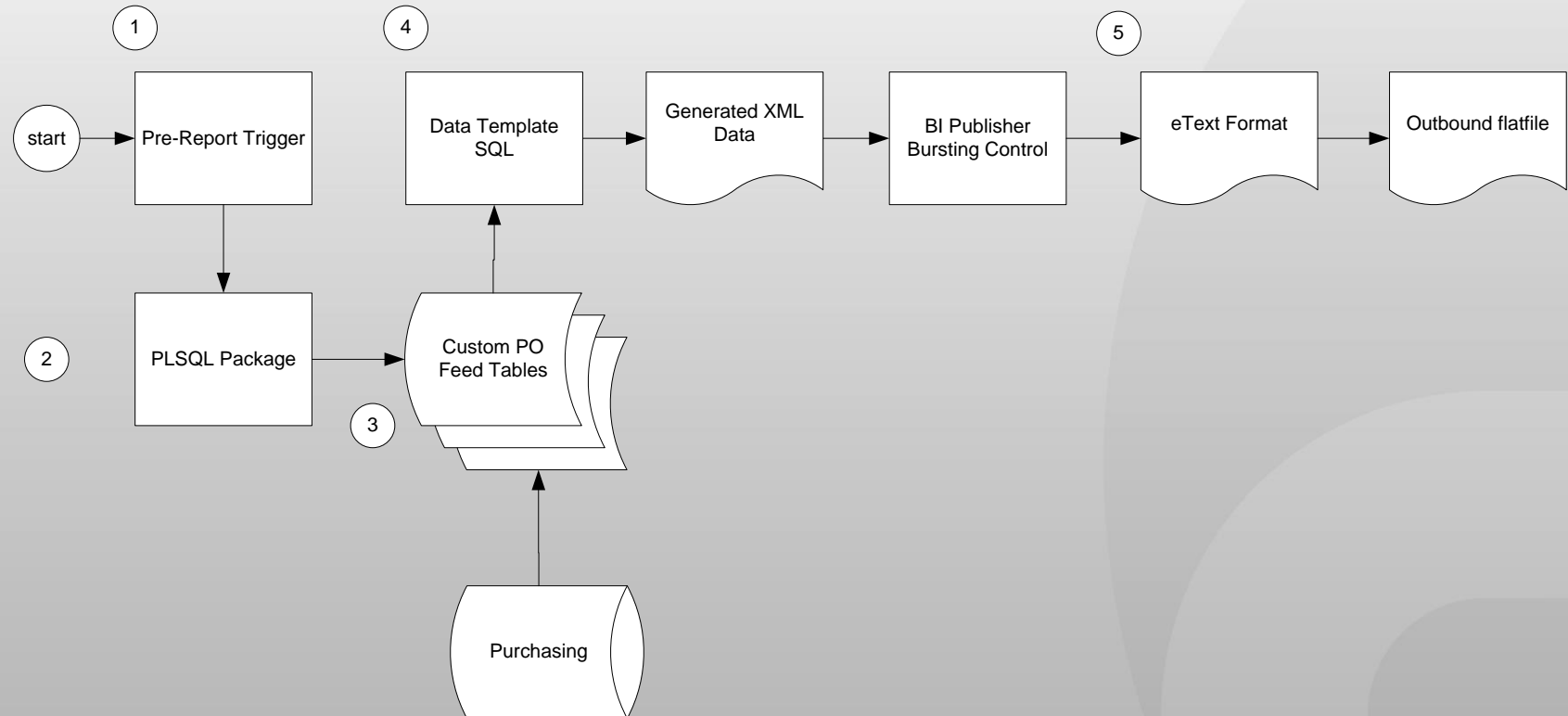
Format Data Records:

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.

<LEVEL>	G INVOICES				
<POSITION>	<LENGTH>	<FORMAT>	<PAD>	<DATA>	<COMMENTS>
<NEW RECORD>	FILE HEADER				
	4	Number		ORG_ID	Business Unit. If defined in ERP
	5	Alpha		\\ ''	Delimiter
	25	Number		PARTY_SITE_NUMBER	Company Code. If used in ERP. Preferred: use BU first. Company only if needed
	5	Alpha		\\ ''	Delimiter
	20	Character		ACCOUNT_NUMBER	Customer Number
	5	Alpha		\\ ''	Delimiter
	30	Alpha		CLASS	Invoice Document Type. Key field
	5	Alpha		\\ ''	Delimiter
	40	Alpha		TRX_NUMBER	Invoice Number. Key field
	5	Alpha		\\ ''	Delimiter
	15	Number		PAYMENT_SCHEDULE_ID	Invoice suffix. Part of the invoice key - varies in use by ERP system. Key field; required if used in implementation. May be required to ensure unique invoice records
	5	Alpha		\\ ''	Delimiter
	300	Character		FILE_NAME_LONG	PDF Filename. ** Filename will be renamed in Cforia to "Company + BU + CustNum + Prefix + Invoice + Suffix.pdf"
	5	Alpha		\\ ''	Delimiter
	5	Alpha		\\ ''	Delimiter
	5	Alpha		\\ ''	Delimiter
	5	Alpha		\\ ''	Delimiter
	5	Alpha		\\ ''	Delimiter
<END LEVEL>	G INVOICES				

Technique #2 – Notifications and Flat File Generation

- Flat File Generation



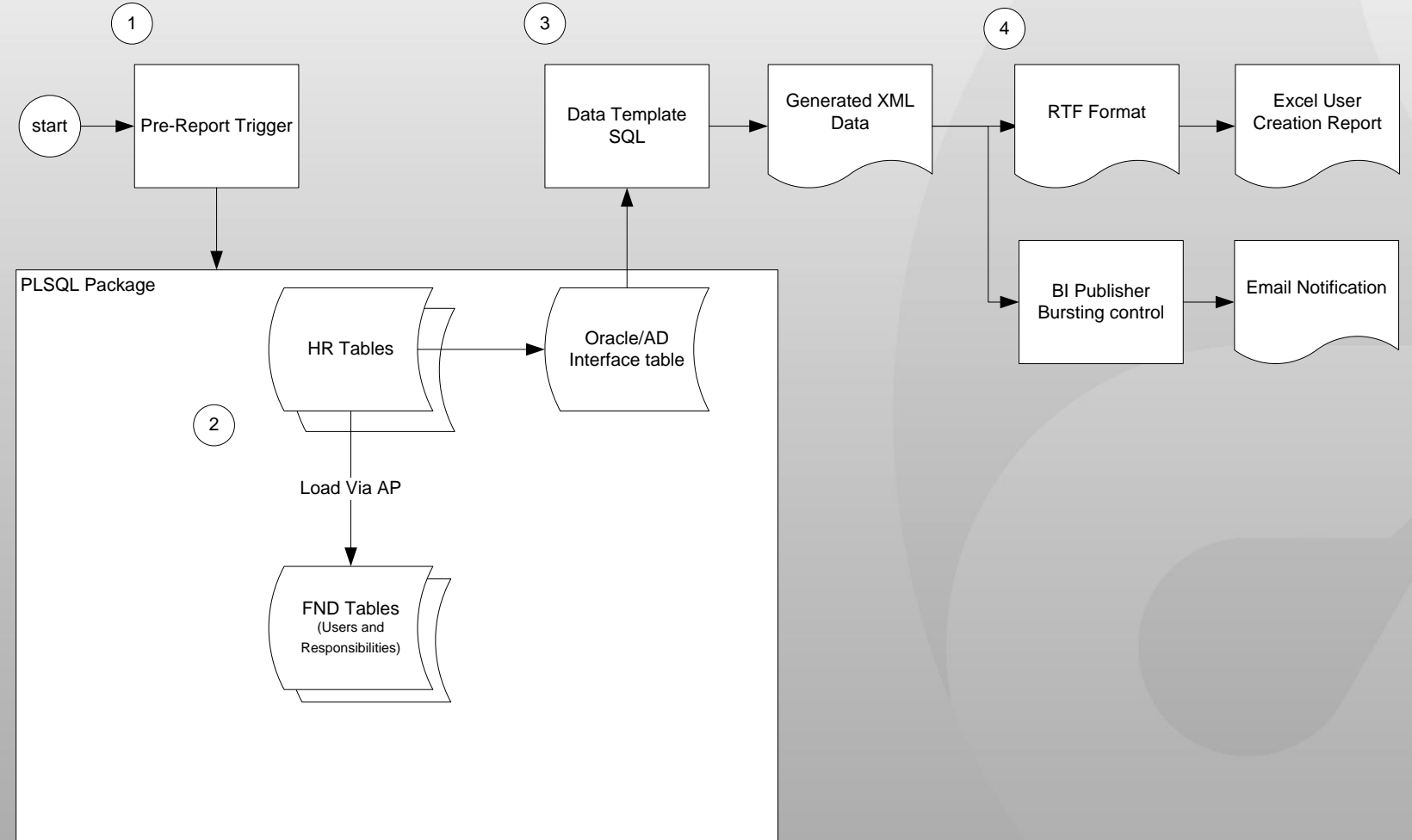
Technique #2 – Notifications and Flat File Generation

- Flat File Generation – Bursting Control File

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- $Id: █████_AR_CF_RPRINT_FILE_GEN_BURST.xml 1234 2019-12-02 10:15:00Z 123456 $ -->
<!-- Revision History:
<!--      Date      By      Description
<!--      =====
<!--      14-JAN-2020    jtseng    Initial version, taken from RFC # CHG0081514, but for
<!--      =====      RFC# CHG0082187
<!--      =====
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi" type="Bursting">
  <xapi:request select="█████_AR_CF_RPRINT_FILE_GEN/LIST_G_INVOICES">
    <xapi:delivery>
      <xapi:filesystem id="FILE_DELIVERY_RPRNT" output="{DIRECTORY_PATH}/{FILE_NAME}" />
    </xapi:delivery>
    <xapi:document utput-type="etext" delivery="FILE_DELIVERY_RPRNT">
      <xapi:template type="etext" location="xdo://█████.█████_AR_CF_RPRINT_FILE_GEN.en.00/?getSource=true">
        </xapi:template>
      </xapi:document>
    </xapi:request>
  </xapi:requestset>
```

Technique #2 – Notifications and Flat File Generation

- Transaction Processing and Notification



Technique #2 – Notifications and Flat File Generation

- Email Notification Generation – Bursting Control File

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- $Header: BURSTING_FILE_AR_ARXSGP.xml 115.1 2015/10/05 03:54:01 xdouser noship $ -->
<!-- dbdrv: none -->

<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi" type="bursting">
<xapi:request select="/ARXSGPO_CPG/LIST_G_SETUP/G_SETUP/LIST_G_STATEMENT/G_STATEMENT">
<xapi:delivery>
<xapi:email id="{CUSTOMER_ID}" server="{SMTP_SERVER_NAME}" port="25" from="{EMAIL_FROM}" reply-to="">
<xapi:message id="{CUSTOMER_ID}" to="{EMAIL_ADDRESS}" attachment="true" subject="Statement {SEND_TO_CUSTOMER_NAME}">
Dear Customer :

Attached you will find your current statement. Please remit payment at your earliest convenience. If you do not have a copy of the invoice

Bank of
ACH ABA #
Account #
Swift Code
Wire ABA#

Lockbox:

We sincerely appreciate your business.

Sincerely,

The Accounting Team

</xapi:message>
</xapi:email>
</xapi:delivery>
<xapi:document key="{SEND_TO_CUSTOMER_NAME}" output="Statement" output-type="pdf" delivery="{CUSTOMER_ID}">
<xapi:template type="rtf" location="xdo://AR.ARXSGPO.en.US/?getSource=true"
filter="//G_STATEMENT[TOTAL_AMOUNT_DUE!='0']"/>
</xapi:document>
</xapi:request>
</xapi:requestset>
```

Technique #2 – Notifications and Flat File Generation

- DON'T
 - Propagate unmanageable code by always using UTL_FILE or FND_FILE to generate flat files
 - Write unnecessary code using UTL_SMTP (or other means) to send out emails
- DO
 - Simplify your code with a BI Publisher Data Template and eText formats to generate Flat Files
 - Use BI Publisher Bursting to send email

Technique #3 – Dynamic Boiler- plating



COLLABORATE 20

TECHNOLOGY & APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

OATUG

ORACLE APPLICATIONS & TECHNOLOGY USERS GROUP

Technique #3 – Dynamic Boiler-Plating

- Typical usage of BI Publisher formatting templates uses different templates for languages and territories
- Multiple templates, however, can lead to a maintenance headache

Technique #3 – Dynamic Boiler-Plating

- Multiple formatting templates supporting multiple languages

The screenshots show the 'Personalize Search' page in Oracle BI Enterprise Edition. The page has a blue header with the Oracle logo and navigation tabs: Templates, Data Definitions, and Administration. The 'Search' section includes fields for Name, Application, and Type, along with a 'Go' button. A 'Personalize Table Layout: (ResultsRn)' section is also present. The main content area displays a list of templates, each with a 'Name' column and a 'Duplicate' button. The templates are organized into a table with columns: Name, Application, Data Definition, Type, Start Date, End Date, and Duplicate. The table shows various templates for different languages and formats, such as 'BNP Argentina Invoice Spanish Report - Original' and 'BNP Japan Invoice Report - Duplicate'. The 'Duplicate' button is used to create new templates based on existing ones, demonstrating the dynamic boiler-plate technique.

Name	Application	Data Definition	Type	Start Date	End Date	Duplicate
BNP Argentina Invoice Spanish Report - Original	XXUL Custom Application	UL BNP Argentina Invoice Report - Original	RTF	23-Sep-2015		
BNP Japan Invoice Report - Original	XXUL Custom Application	UL BNP Japan Invoice Report - Original	RTF	27-Oct-2012		
BNP Japan Invoice Report - Duplicate	XXUL Custom Application	UL BNP Japan Invoice Report - Duplicate	RTF	27-Oct-2012		
BNP German Invoice Report-Original	XXUL Custom Application	UL BNP Invoice Report - Original	RTF	12-Jul-2013		
BNP German Invoice Report-Duplicate	XXUL Custom Application	UL BNP Invoice Report - Duplicate	RTF	12-Jul-2013		

Technique #3 – Dynamic Boiler-Plating

- An alternate approach to having multiple templates is to use a single template
 - Assemble all data elements and boiler-plate elements into a staging table.
 - Have boiler-plate fields be programmatically determined
 - Boiler-plate fields can be set based upon different criteria – such as language or operating unit
 - Expand the data template to query both boiler-plate fields and data elements
 - Place place-holder fields for boiler-plate fields into the template

Technique #3 – Dynamic Boiler-Plating

- Programmatic build of boiler-plate text

```
log_message('set_boilerplate_fields()+ 10');
p_header_rec.hdr_billing_to      := xxul_ar_invoice_utils_pkg.get_text('HDR_BILLING_TO', p_header_rec.language_code);
p_header_rec.hdr_return_addr_note := xxul_ar_invoice_utils_pkg.get_text('HDR_RETURN_ADDR_NOTE', p_header_rec.language_code, p_header_rec.org_id);

-- #CHG0073099
IF p_header_rec.print_tax_invoice_flag = 'Y' THEN
    p_header_rec.hdr_dtl_invoice_number      := xxul_ar_invoice_utils_pkg.get_text('HDR_DTL_THAI_ORACLE_NUMBER', p_header_rec.language_code); -- This returns "Oracle Inv#" for Thai transactions
    p_header_rec.hdr_dtl_thai_trxn_number    := xxul_ar_invoice_utils_pkg.get_text('HDR_DTL_THAI_TRXN_NUMBER', p_header_rec.language_code); -- This returns "Number" for custom Thai Invoice Number
    p_header_rec.hdr_document_type          := xxul_ar_invoice_utils_pkg.get_text('HDR_DOCUMENT_TYPE_TAX_INV', p_header_rec.language_code, p_header_rec.org_id, NULL, p_header_rec.cust_trx_type);
    p_header_rec.hdr_dtl_invoice_date       := xxul_ar_invoice_utils_pkg.get_text('HDR_DTL_TAX_INVOICE_DATE', p_header_rec.language_code);
ELSE
    p_header_rec.hdr_dtl_invoice_number      := xxul_ar_invoice_utils_pkg.get_text('HDR_DTL_INVOICE_NUMBER', p_header_rec.language_code); -- This returns Invoice (standard trx label)
    p_header_rec.hdr_dtl_invoice_date       := xxul_ar_invoice_utils_pkg.get_text('HDR_DTL_INVOICE_DATE', p_header_rec.language_code);
END IF;

-- CHG0073691 -- Cost limit increase/decrease (Credit Note) specific to Thailand
IF (p_header_rec.credit_note_flag = 'Y' AND p_header_rec.cost_limit_increase_flag = 'N') THEN
    p_header_rec.hdr_document_type := 'CREDIT NOTE';
    p_header_rec.line_tot_sub_total_cn := xxul_ar_invoice_utils_pkg.get_text('LINE_TOT_SUB_TOTAL_CN', p_header_rec.language_code);
ELSEIF (p_header_rec.credit_note_flag = 'Y' AND p_header_rec.cost_limit_increase_flag = 'Y') THEN
    p_header_rec.line_tot_sub_total_cn := xxul_ar_invoice_utils_pkg.get_text('LINE_TOT_SUB_TOTAL_CN', p_header_rec.language_code);
END IF;
```

Technique #3 – Dynamic Boiler-Plating

- The BI Publisher formatting template:

<pre><?./ancestor-or-self::*/LINE_DTL_DESCRIPTION_COL?></pre>	<pre><?./ancestor-or-self::*/LINE_DTL_QUANTITY_COL?></pre>	<pre><?./ancestor-or-self::*/LINE_DTL_UNIT_MEASURE_DSP?></pre>	<pre><?./ancestor-or-self::*/LINE_DTL_UNIT_SELLING_PRICE_DSP?></pre>	<pre><?./ancestor-or-self::*/LINE_DTL_EXTENDED_AMOUNT_DSP?><end for-each?><end for-each?></pre>
<pre><?for-each:LIST_G_INV_LINE?><?for-each:G_INV_LINE?><?if@row:number(ROWX mod2)=0?><?attribute@incontext:background-color;#D3D3D3?><?add-page-total:qtytot;'QUANTITY_DSP'?><?add-page-total:pagetot;'EXTENDED_AMT'?><?add-page-total:contd_footer;1?><?DESCRIPTION_DSP?><?./../INVOICE_PRINT_TYPE?></pre>	<pre><?QUANTITY_DSP?></pre>	<pre><?UNIT_OF_MEASURE_DSP?></pre>	<pre><?UNIT_SELLING_PRICE_DSP?></pre>	<pre><?EXTENDED_AMOUNT_DSP?><end for-each?><end for-each?></pre>

Technique #3 – Dynamic Boiler-Plating

- Depending upon the requirements, build a custom front-end so a smart user or analyst can dynamically specify boiler-plate text
 - A simple front-end can be built using Oracle Forms or APEX
 - The front-end can support rules for which boiler-plate to show based upon different criteria – such as language or operating unit (or both)

Technique #3 – Dynamic Boiler-Plating

- In this example, invoice “boilerplate” text for different languages, operating units, and bill to countries is maintained in a basic custom table with a supporting custom Oracle Form:

Operating Unit, French language
Specific text for a Bill to in France

UL Invoice Text and Lookup Maintenance

Code: ADDITIONAL_PAYMENT_MESSAGE
Description: Additional payment message at bottom of invoice
Default Value:

Values

Value	Language Code	Operating Unit	BT Country	Key
Note: All payments are to be made directly from an i	US	DEWI Espana		
Commentaire: Tous les paiements doivent être faits	F	DEWI France	France	
Note: All payments are to be made directly from an i	US	DEWI France		
Hinweis: Alle Zahlungen müssen direkt von einer Bar	D	DEWI OCC	Germany	
Note: All payments are to be made directly from an i	US	DEWI OCC		

Editor

Commentaire: Tous les paiements doivent être faits directement depuis un organisme et un compte non sujet à des sanctions de la part des Nations Unis ou des autorités locales ou US. Pour plus d'informations sur nos services, merci de consulter notre page internet www.ul.com ou www.dewi.de. Pour toute question de facturation et de changement d'adresse, merci de contacter notre Département Comptable au +33 (0) 4 27 46 22 70 ou par fax +33 (0) 4 27 46 22 79. DEWI France, 90 rue Paul Bert, 69003 Lyon, France Directeur général: Jens Peter Molly DEWI France est inscrite au Registre du Commerce de Lyon sous le numéro 483 622 130 RCS LYON. DEWI France est une succursale de la société à responsabilité limitée de droit allemand UL International GmbH sise Eberstrasse 96, 26382 Wilhelmshaven, Germany, immatriculée auprès du Tribunal d'instance d'Oldenburg sous le numéro HBR 130241

OK Cancel Search

Technique #3 – Dynamic Boiler-Plating

- Dynamic outputs, one template

AG
CIO UL AG Taiwan Branch
112台北市北投区大业路260号 (财务组)

中山市, 广东省 528415
中国

发票

发票号码: 34720126744
客户编号: 494875
发票日期: 26-JUL-2016
订单号码: 11153611
PO:
支付期限:
VAT Registration:
客户联系人:

应付总额: 998.93 USD

收费明细	数量	单位	单价	金额
Lighting Services, OOLV(Listed Tube), 4 models, UL+CUL (E480762)				
UL6UL (New Certification)				
Transportation				
Living Expenses				
Lodging				
合计				5,758.93
Less Deposit Invoice(34780027975)				-4,760.00
应付总金额				998.93 USD

以上UL提供服务的中国增值税及其他附加税按0.83%计算, 为USD 393.33。
请在付款给UL前直接把相关税额支付给中国税局, 该税额 (含增值税及其他附加税) 是基于0.83%计算所得。
UL发票金额并不包含任何适用的中国可能征收地方税务局税。您不得扣除任何形式或缴纳增值税或营业税、银行手续费、关税或捐税等。
具体税费可能因不同地区规定有所区别, 以当地税局规定为准。

第 1 页 / 共 3 页

The address below is for regional mailing purposes only and is not associated with the local entity displayed.

Quality Assurance Private Limited
CIO UL India Pvt Ltd.
Credit & Collections Department
Kalyani Platina, 3rd Floor, Block I, No. 24, EPIP Zone,
Phase II
Whitefield, Bangalore - 560 066, INDIA

INVOICE

Invoice: 10920084635
Bill To Account: 1169888
Invoice Date: 19-JUL-2016
Sales Order: 11362039
PO:
Payment Terms: NET 30
Permanent Account #:
Service Tax Number:
CIN:
Customer Contact:

Amount Due:

Description	Quantity	UOM	Unit Price	Amount
Softlines testing				
Total Amount Due				2,121.76 INR

Services Provided For:

Additional Information

Project Name IA16-18379

Page 1 of 2

Technique #3 – Dynamic Boiler-Plating

- DON'T
 - Always propagate a maintenance nightmare by using different templates for different languages or purposes
- DO
 - Use dynamic boiler-plating to ease your maintenance issues
 - Create a SINGLE template that can support different languages (or other criteria), rather than creating a template for every language
 - Create a front-end from which a smart user can maintain document contents.

Technique #4 – OAF Extension Reporting



COLLABORATE 20

TECHNOLOGY & APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

OATUG

ORACLE APPLICATIONS & TECHNOLOGY USERS GROUP

Technique #4 – OAF Extension Reporting

- Typically, BI Publisher reports and processes are setup as concurrent programs within Oracle EBS. An relatively unknown extension option, however is to create an OAF page that enables a BI Publisher Report to be executed directly from an EBS form or page.
 - Create an OAF page that uses a custom controller to launch a BI Publisher report
 - Create a function for the custom OAF page
 - Enable personalizations on the EBS OAF page or form to call the OAF page.

Technique #4 – OAF Extension Reporting

- Using JDeveloper, a custom OAF page and Controller can instantiate the objects necessary to call a BI Publisher report

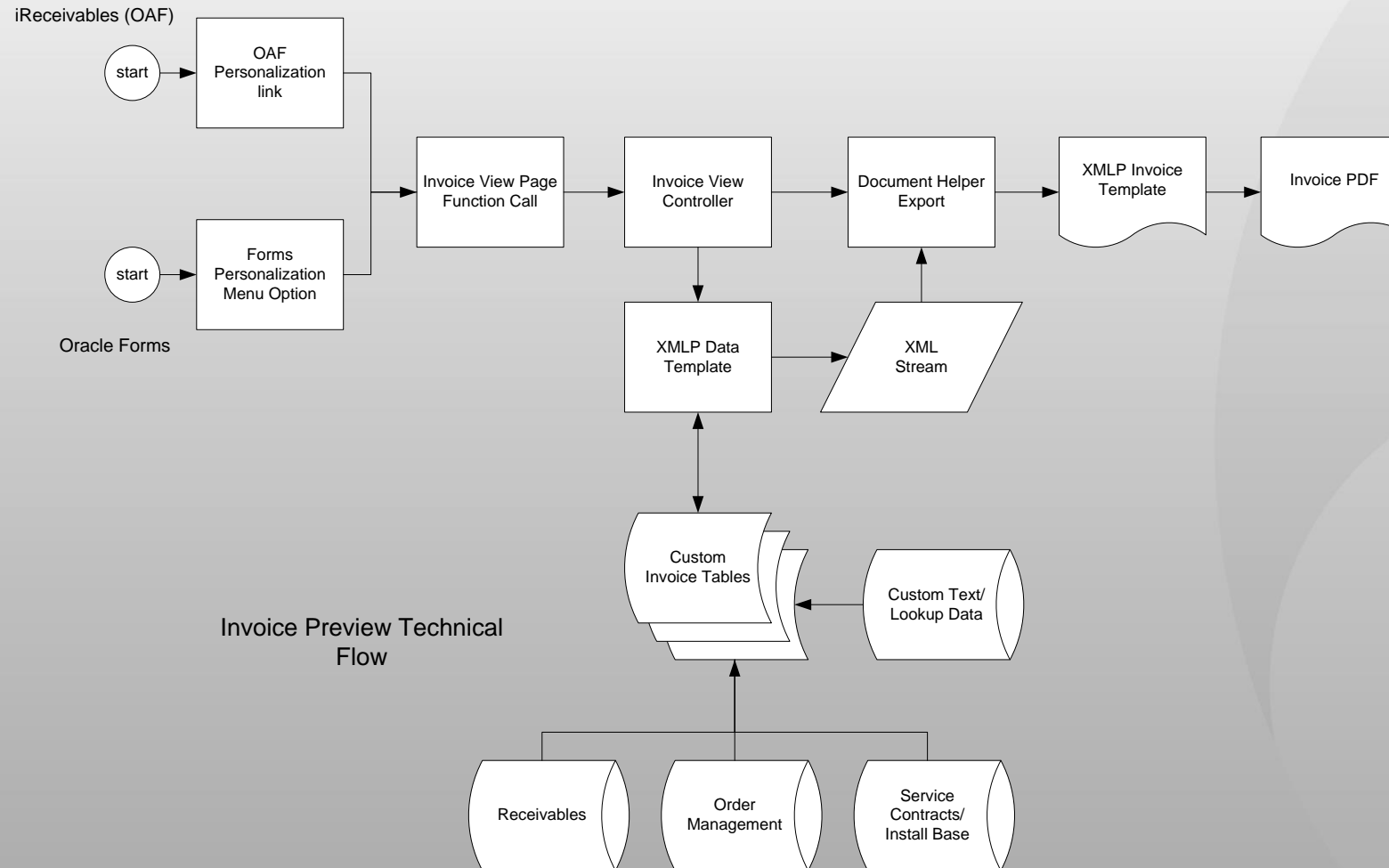
```
}  
  
public BlobDomain getInvoiceData(String pFromCustomerTrxId,  
                                String pToCustomerTrxId,  
                                String pFromConsInvId,  
                                String pToConsInvId,  
                                String pRebuildFlag,  
                                String pForceEnglishFlag) {  
  
    /* inbound parameters  
       0 - customer trx id  
       1 - build flag  
       2 - force english flag  
    */  
  
    String applicationShortName = "XXUL";  
    String dataSourceCode = "XXUL_AR_INVOICE";  
  
    BlobDomain blobDomain = new BlobDomain();  
  
    try {  
        DataTemplate dataTemplate = new DataTemplate(((OADETransactionImpl) getOADETransaction()).getAppsContext(),  
                                                    applicationShortName,  
                                                    dataSourceCode);  
  
        // Get parameters and set them  
  
        ArrayList parameters = dataTemplate.getParameters();  
        Iterator it = parameters.iterator();  
    }  
}
```

Technique #4 – OAF Extension Reporting

- Controller Code

```
    }  
  
    } else {  
  
        BlobDomain result = (BlobDomain)oaAM.invokeMethod("getInvoiceData", parameters);  
        System.out.println("*****");  
        System.out.println("Iteration: 201604080656");  
  
        System.out.println("*****XML Data *****");  
  
        System.out.println(result.toString());  
  
        try {  
            DocumentHelper.exportDocument(pageContext,  
                                        "XXUL",  
                                        "XXUL_AR_INVOICE",  
                                        "en",  
                                        "00",  
                                        result.getInputStream(),  
                                        "PDF",  
                                        null);  
        } catch (Exception e) {  
            System.out.println("Exception occurred. " + e.getMessage());  
        }  
    }  
}
```


Technique #4 – OAF Extension Reporting



Technique #4 – OAF Extension Reporting

- Example: Invoice Preview via an OAF Extension page

Oracle Applications - EBSDEV Cloned From PRERP on February-20-17

File Edit View Folder Tools Reports Actions Window Help

Exchange Rate
Create Accounting
View Accounting
Distributions
Balances
Freight
Show Closed Invoices
Line Items
Overview
Sales Credits
Tax
About AR
PrintPreview (OLD)
Print Preview

Transaction (UL Jap)

Transaction

Source
Number
Class
Type
Reference
Legal Entity

Main

Name HOS
Number 84917
Location 1305739
Address
YAO-SHI, OSAKA 581-0071 Japan
Contact

Ship

Commitment
Payment Term NET 60
Invoicing Rule In Advance
Due Date 23-AUG-2016

Date 24-JUN-2016
GL Date 24-JUN-2016
Currency JPY
Document Num
Transaction [BL]
Complete

Balance Due

Line 0
Tax 0
Freight 0
Charges 0
Total 0

Details Refresh

Bill To

Name
Number 84917
Location 1305738
YAO-SHI, OSAKA 581-0071 Japan
UMEDA, TOMOKI

Sold To

Name HOSIDEN CORP
Number 84917

Paying Customer

Name
Number
Location

Payment Details

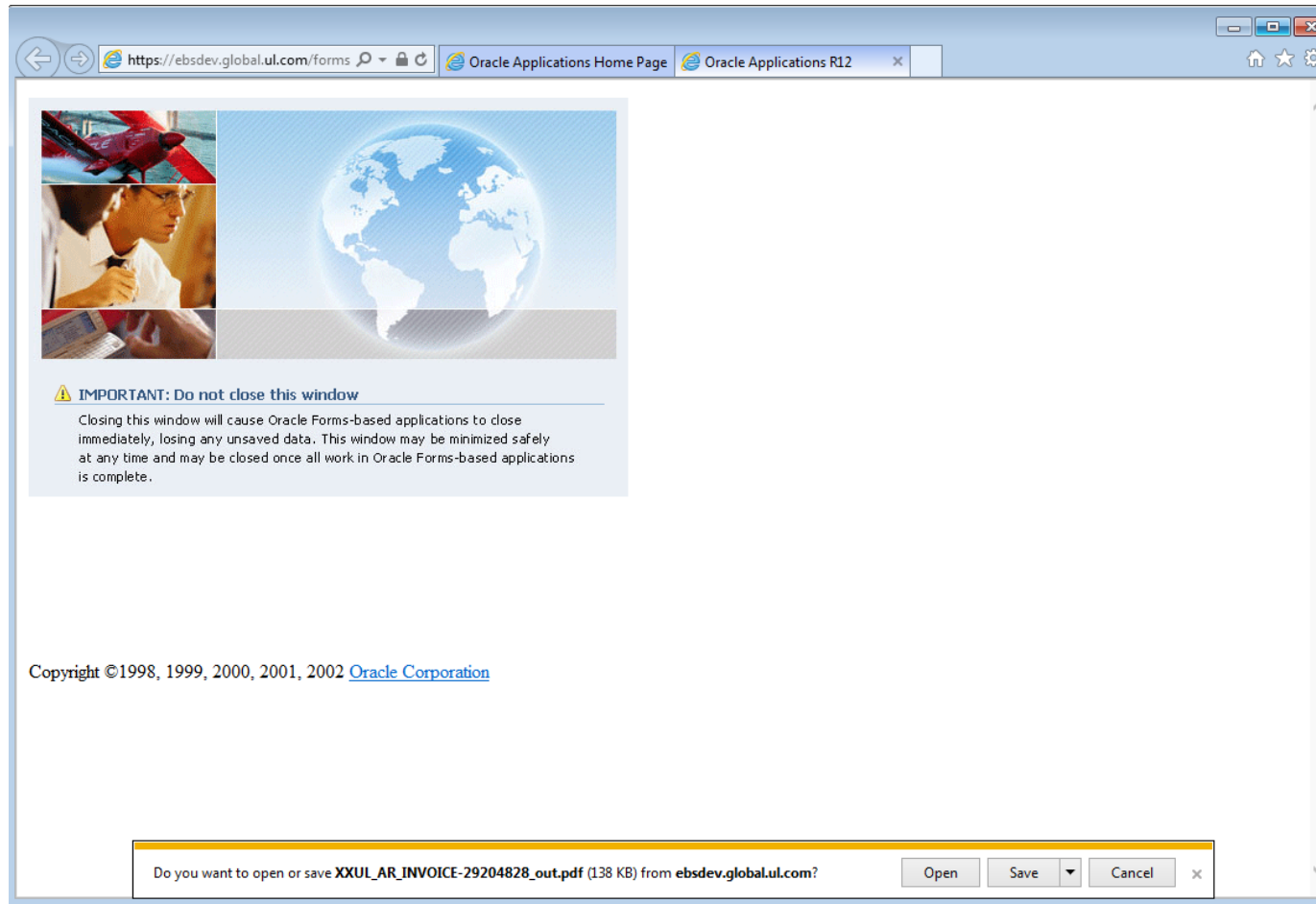
Receipt Method
Payment Method
Instrument Number
Select Instrument

Line Items Tax Freight Distributions Sales Credits Incomplete

FRM-40200: Field is protected against update.
Record: 1/1 <OSC>

Technique #4 – OAF Extension Reporting

- Example: Invoice Preview via an OAF Extension page



Technique #4 – OAF Extension Reporting


- Example: Invoice Preview via an OAF Extension page


XXUL_AR_INVOICE-29204828_out.pdf - Adobe Reader


File Edit View Window Help

1 / 2 100%

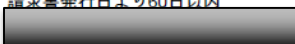
Comment

 **Japan, Inc.**
〒516-0021 三重県伊勢市朝熊町4383番326
株式会社 UL Japan





請求書

請求書番号: 14020049985
請求先お客様番号: 84917
発行日: 24-JUN-2016
オーダー番号: 11147901
P.O.番号:
お支払期日: 請求書発行日より60日以内
UL TAX ID: 
貴社担当者: 梅田 朋季様

請求金額合計: 506,520 円

業務内容	数量	単位	単価	金額
各国電波法申請代行				

Technique #4 – OAF Extension Reporting

- DON'T
 - Be limited to thinking that submitting a concurrent process through the standard submission screen is your only option
- DO
 - Build an OAF extension page to help a user maintain “focus” on the transaction or document at hand

Technique #5 – Bursting to XML



COLLABORATE 20

TECHNOLOGY & APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

OATUG

ORACLE APPLICATIONS & TECHNOLOGY USERS GROUP

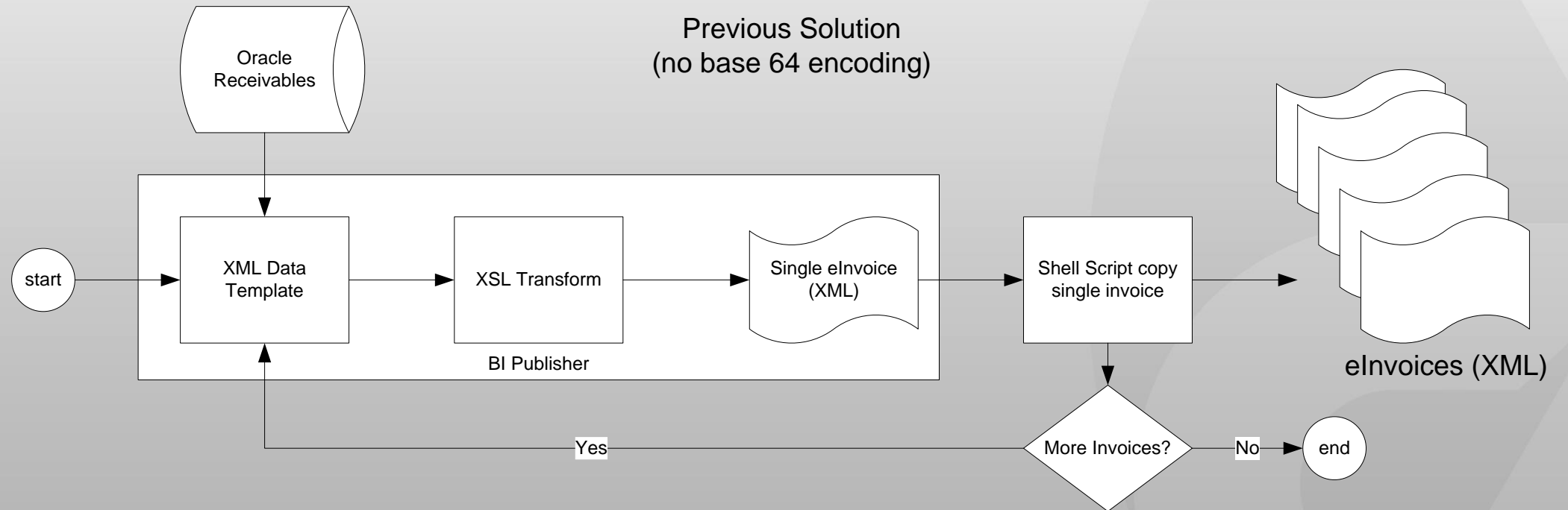
Technique #5 – Bursting to XML

- One of the most surprising limitations of BI Publisher is the seemingly limited ability to use bursting to generate XML files.
 - Cannot burst through XSL to generate XML files
 - Can use an eText RTF to “construct” an XML file
- An alternative approach is to use the native XML functions inside the Oracle database to build XML and burst the XML to an eText RTF.

Technique #5 – Bursting to XML

- At our client, the Italian government required the generation of eInvoice files. These were XML data files.
 - Oracle provided a localization “patch” that generated these files
 - Functionality was very limited, and the files still needed to be processed through a 3rd party vendor. Customizations to the localization provided were still required.
 - The main limitation in the provided patch was that it could only generate one “eInvoice” file at a time.
- The initial solution built was severely limited by the one-at-a-time limitation. In order to generate each eInvoice file, a concurrent process was required for each file. Generating 5000 invoices required 5000 concurrent processes

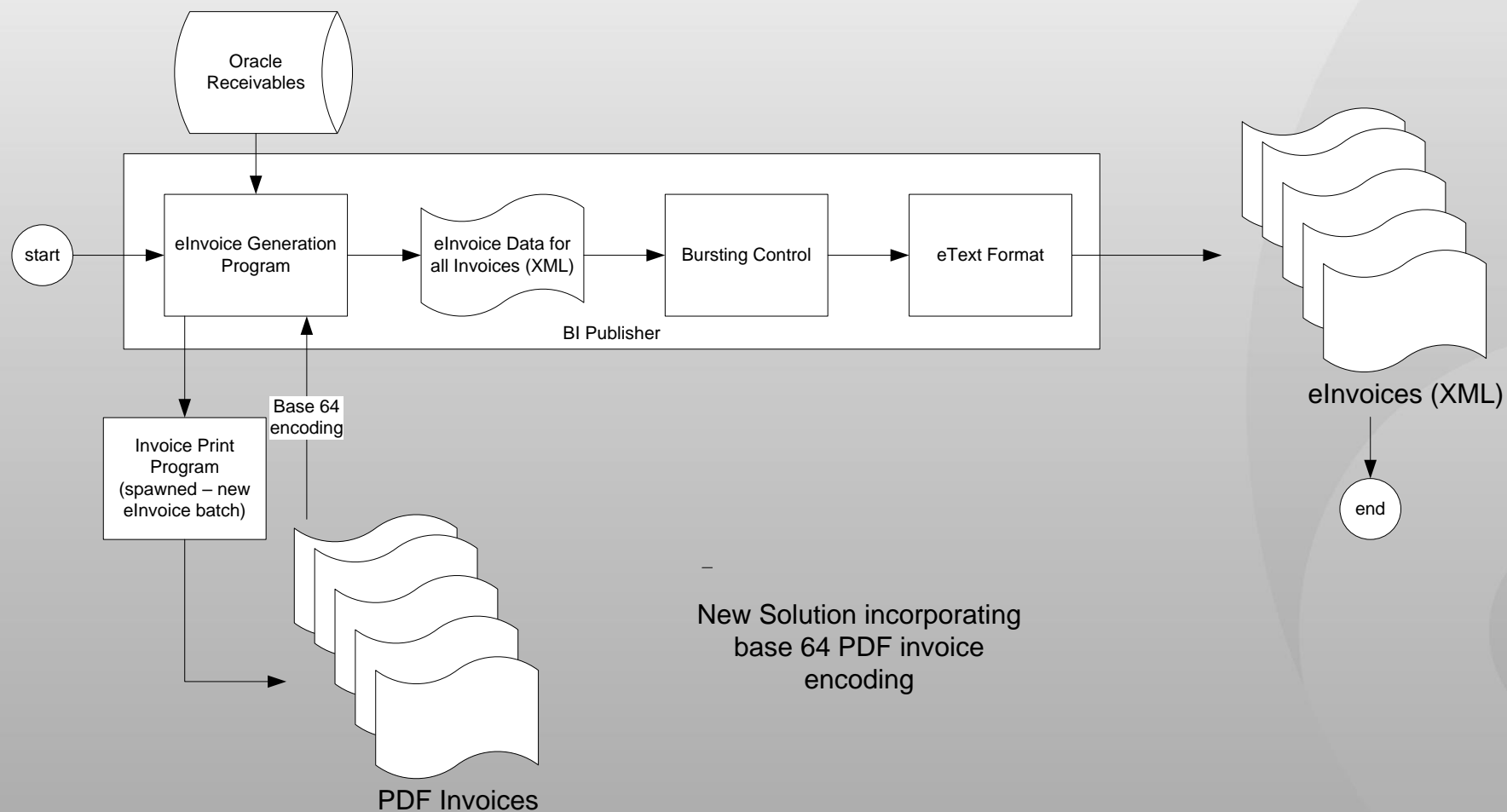
Technique #5 – Bursting to XML



Technique #5 – Bursting to XML

- After implementation, a new requirement was added. The PDF of each invoice was required to be added to the generated XML “eInvoice” so that the customer could print the invoice.
- We re-architected the solution to add the PDF invoice as a base 64 encoded string in the XML and, more importantly, moved the generation of the XML from BI Publisher to within the database
 - A Java utility was written to base 64 encode a PDF file
 - The SQL in the Oracle provided data template was moved into PLSQL code and “encased” in XML generating functions
 - The XSL transform used to generate the XML was not used in BI Publisher, but was instead fired from within the database code
 - A simple eText RTF formatting template was used in the bursting control file to generate the XML.

Technique #5 – Bursting to XML



Technique #5 – Bursting to XML

- XML generation moved to the PLSQL code

```
CURSOR Trx_Footer_Details_cur (P_TRX_ID                ra_customer_trx_all.customer_trx_id%TYPE,
                             P_INTERFACE_HDR_ATTRIBUTE1 ra_customer_trx_all.interface_header_attribute1%TYPE,
                             P_TAX_CODE                VARCHAR2,
                             P_TRX_DOC_TYPE            VARCHAR2
                             ) IS

SELECT XMLELEMENT("LIST_G_TRX_FOOTER_DETAILS", XMLAGG(
    XMLELEMENT("G_TRX_FOOTER_DETAILS",XMLFOREST(tax_rate, tax_rate_status , taxable_func_amt_per_rate, tax_func_amt_per_rate,
    taxable_entered_amt_per_rate, tax_entered_amt_per_rate, nature_of_vat_f, law_reference_f, trx_line_attribute1_f, trx_line_attribute2_f,
    trx_line_attribute3_f, trx_line_attribute4_f, trx_line_attribute5_f, trx_line_attribute6_f, trx_line_attribute7_f, trx_line_attribute8_f,
    trx_line_attribute9_f, trx_line_attribute10_f, trx_line_attribute11_f, trx_line_attribute12_f, trx_line_attribute13_f, trx_line_attribute14_f,
    trx_line_attribute15_f, tax_ent_amt_min_dep, tax_ent_amt_per_min_dep, dep_total, dep_amount)))) output
FROM
(SELECT trim(to_char(rates.percentage_rate,'990D00','NLS_NUMERIC_CHARACTERS = ','.')) tax_rate,
decode(P_TAX_CODE,'IT_22D','S','IT_22S','S',decode(rates.def_rec_settlement_option_code,'IMMEDIATE','I','DEFERRED','D','S')) tax_rate_status ,-- DH changed from null to S
CASE WHEN P_TRX_DOC_TYPE = 'TD04' THEN trim(to_char(abs(sum(ROUND(item_dist.amount* NVL(trx.exchange_rate, 1),2)))+ nvl(b.dep_amount,0),'9999999999999999D00','NLS_NUMERIC_CHARACTERS = ','.')) ELSE
trim(to_char(abs(sum(ROUND(item_dist.amount* NVL(trx.exchange_rate, 1),2))),'9999999999999999D00','NLS_NUMERIC_CHARACTERS = ','.')) END taxable_func_amt_per_rate,
CASE WHEN P_TRX_DOC_TYPE = 'TD04' THEN trim(to_char(abs(sum(ROUND(tax_dist.amount * NVL(item_dist.percent/100,1) * NVL(trx.exchange_rate, 1),2))),'9999999999999999D00','NLS_NUMERIC_CHARACTERS = ','.')) ELSE
trim(to_char(abs(sum(ROUND(tax_dist.amount * NVL(item_dist.percent/100,1) * NVL(trx.exchange_rate, 1),2))),'9999999999999999D00','NLS_NUMERIC_CHARACTERS = ','.')) END tax_func_amt_per_rate,
CASE WHEN P_TRX_DOC_TYPE = 'TD04' THEN trim(to_char(abs(sum(ROUND(item_dist.amount,2)))+ nvl(a.dep_amount,nvl(b.dep_amount,0)),'9999999999999999D00','NLS_NUMERIC_CHARACTERS = ','.')) ELSE
trim(to_char(sum(ROUND(item_dist.amount,2))+ nvl(a.dep_amount,nvl(b.dep_amount,0)),'9999999999999999D00','NLS_NUMERIC_CHARACTERS = ','.')) END taxable_entered_amt_per_rate, ----
CASE WHEN P_TRX_DOC_TYPE = 'TD04' THEN trim(to_char(ROUND(abs(SUM(tax_dist.amount * NVL(item_dist.percent/100,1)))+ nvl(a.dep_tax,nvl(b.dep_tax,0)),2),'9999999999999999D00','NLS_NUMERIC_CHARACTERS = ','.')) ELSE
trim(to_char(ROUND(abs(SUM(tax_dist.amount * NVL(item_dist.percent/100,1)))+ nvl(a.dep_tax,nvl(b.dep_tax,0)),2),'9999999999999999D00','NLS_NUMERIC_CHARACTERS = ','.')) END tax_entered_amt_per_rate,
max(JE_IT_ELECTRONIC_INV_EXTRACT.get_reporting_code(tax_lines.tax_line_id,'NATURE_OF_VAT')) nature_of_vat_f,
max(JE_IT_ELECTRONIC_INV_EXTRACT.get_reporting_code(tax_lines.tax_line_id,'LAW_REFERENCE')) law_reference_f,
max(item_lines.attribute1)   trx_line_attribute1_f,
max(item_lines.attribute2)   trx_line_attribute2_f,
max(item_lines.attribute3)   trx_line_attribute3_f,
max(item_lines.attribute4)   trx_line_attribute4_f,
max(item_lines.attribute5)   trx_line_attribute5_f,
max(item_lines.attribute6)   trx_line_attribute6_f,
max(item_lines.attribute7)   trx_line_attribute7_f,
```

Technique #5 – Bursting to XML

- All XML components assembled into one raw XML field

```
SELECT XMLELEMENT("JEITEIFO", XMLCONCAT(XMLELEMENT("P_LEGAL_ENTITY_ID", P_LEGAL_ENTITY_ID),
      XMLELEMENT("P_CUST_ACCOUNT_ID", P_CUST_ACCOUNT_ID),
      XMLELEMENT("P_BILL_TO_SITE_USE_ID", P_BILL_TO_SITE_USE_ID),
      XMLELEMENT("P_GEN_OPTION", P_GEN_OPTION),
      XMLELEMENT("P_TRX_DATE_FROM", P_TRX_DATE_FROM),
      XMLELEMENT("P_TRX_DATE_TO", P_TRX_DATE_TO),
      XMLELEMENT("P_TRX_ID", P_TRX_ID),
      XMLELEMENT("P_OLD_TRANSMISSION_NUM", P_OLD_TRANSMISSION_NUM),
      XMLELEMENT("P_NEW_TRANSMISSION_NUM", P_NEW_TRANSMISSION_NUM),
      XMLELEMENT("P_TRANSMISSION_FILE_VER", P_TRANSMISSION_FILE_VER), |
      XMLELEMENT("P_PROFILE_CLASS_ID", P_PROFILE_CLASS_ID),
      XMLELEMENT("P_TRANSACTION_TYPE_ID", P_TRANSACTION_TYPE_ID),
      XMLELEMENT("P_TRANSACTION_CLASS", P_TRANSACTION_CLASS),
      l_xml_rec.col1, l_xml_rec.col2, l_xml_rec.col3, l_xml_rec.col4, l_xml_rec.col5, l_xml_rec.col6, l_hdr_loop)) FINAL_RAW_XML
INTO l_final_raw_xml
FROM DUAL;
```

Technique #5 – Bursting to XML

- Assembled XML stored in a temporary table

sqlplus.exe - Shortcut

```
-----
NUM_CHAR          VARCHAR2          IN
SQL> descr █████_ar_it_einv_xml
Name              Null?      Type
-----
REQUEST_ID                NUMBER
REC_CREATION_DATE         DATE
CUST_TRX_ID               NUMBER
TRX_NUMBER                NUMBER
SEQUENCE_NUMBER           NUMBER
TRX_DATE                  DATE
ORG_ID                    NUMBER
COL1                      SYS.XMLTYPE STORAGE BINARY
COL2                      SYS.XMLTYPE STORAGE BINARY
COL3                      SYS.XMLTYPE STORAGE BINARY
COL4                      SYS.XMLTYPE STORAGE BINARY
COL5                      SYS.XMLTYPE STORAGE BINARY
COL6                      SYS.XMLTYPE STORAGE BINARY
COL7                      SYS.XMLTYPE STORAGE BINARY
COL8                      SYS.XMLTYPE STORAGE BINARY
COL9                      SYS.XMLTYPE STORAGE BINARY
COL10                    SYS.XMLTYPE STORAGE BINARY
COL11                    SYS.XMLTYPE STORAGE BINARY
COL12                    SYS.XMLTYPE STORAGE BINARY
FINAL_RAW_XML             SYS.XMLTYPE STORAGE BINARY
STATUS                    VARCHAR2(20)
MESSAGE                   VARCHAR2(200)
SQL>
```

Technique #5 – Bursting to XML

- XML transformed via SQL in the Data Template

```
</properties>
<parameters>
  <parameter name="P_LEGAL_ENTITY_ID" dataType="NUMBER"/>
  <parameter name="P_CUST_ACCOUNT_ID" dataType="NUMBER"/>
  <parameter name="P_BILL_TO_SITE_USE_ID" dataType="NUMBER"/>
  <parameter name="P_PROFILE_CLASS_ID" dataType="NUMBER"/>
  <parameter name="P_TRANSACTION_CLASS" dataType="VARCHAR2"/>
  <parameter name="P_TRANSACTION_TYPE_ID" dataType="NUMBER"/>
  <parameter name="P_RPT_GEN_OPTION" dataType="VARCHAR2"/>
  <parameter name="P_RPT_GEN_OPTION_DUMMY" dataType="VARCHAR2"/>
  <parameter name="P_RPT_GEN_OPTION_DUMMY1" dataType="VARCHAR2"/>
  <parameter name="P_TRX_DATE_FROM" dataType="VARCHAR2"/>
  <parameter name="P_TRX_DATE_TO" dataType="VARCHAR2"/>
  <parameter name="P_TRX_ID" dataType="NUMBER"/>
  <parameter name="P_TRANS_PROG_NUM" dataType="NUMBER"/>
  <parameter name="P_TRANS_FILE_VER" dataType="VARCHAR2"/>
  <parameter name="P_NO_INV_PER_FILE" dataType="VARCHAR2"/>
</parameters>
<lexicals>
</lexicals>
<dataQuery>
  <sqlStatement name="Q_INVOICE">
    <![CDATA[ SELECT z.trx_number TRX_NUMBER,
      decode(:P_LEGAL_ENTITY_ID, 59290, 'Nuovo Istituto', 59289, 'ICQ SRL',
        187547, 'UL_GmbH Italy', 46281, 'UL_Italia', 59287, 'ICQ_Holding') FOLDER,
      EXTRACTVALUE(z.coll,'LIST_G_LE_DETAILS/G_LE_DETAILS/FILENAME') FILENAME,
      db.name INSTANCE,
      XMLTRANSFORM(z.final_raw_xml, xsl_x.xsl_transform).getClobVal() xmloutput
    FROM (SELECT XMLTYPE(c.file_data,l) xsl_transform
    FROM xdo_lobs c
    WHERE lob_code = 'XXUL JEITEIFOB2B'
    AND c.lob_type = 'TEMPLATE' ) xsl_x,
    _ar_it_einv_xml z,
    v$database db
    WHERE z.request_id = FND_GLOBAL.CONC_REQUEST_ID
    --AND z.cust_trx_id = :P_TRX_ID
    ]]>
  </sqlStatement>
</dataQuery>
<dataTrigger name="beforeReport" source="XXUL_AR_IT_EINV_PKG.beforeReport(:P_LEGAL_ENTITY_ID, :P_CUST_ACCOUNT_ID, :P_BIL
:P_RPT_GEN_OPTION, :P_RPT_GEN_OPTION_DUMMY, :P_R
<dataStructure>
  <group name="G_INVOICE" source="Q_INVOICE">
    <element name="TRX_NUMBER" value="TRX_NUMBER"/>
```


Technique #5 – Bursting to XML

- eText Format to produce XML

XDO file name:
Date:04/19/16
[REDACTED]_XML_GEN.rtf

XML Generate eText/XML
[REDACTED] XML GEN eText/XML Export

Format Setup:

Hint Define formatting options...

<TEMPLATE TYPE>		DELIMITER_BASED
<OUTPUT CHARACTER SET>		UTF-8
<NEW RECORD CHARACTER>		

Format Data Records :

<LEVEL>		G_INVOICE
<MAXIMUM LENGTH>	<FORMAT>	<DATA>
<NEW RECORD>		G_DATA
	Alpha	XMLOUTPUT
<END LEVEL>		G_INVOICE

Technique #5 – Bursting to XML

- Bursting Control to generate XML through eText format

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- *****
*****
Program File Name      : [REDACTED] AR IT EINV BURST.xml
Created By            : [REDACTED]
Creation Date         : 15-APR-2019
Object Type          : Bursting Control File
Object Name           : [REDACTED]_AR_IT_EINV_BURST (Data Template)
Description            : [REDACTED] Italian B2B Electronic Invoice.
*****
Modification History
*****
Date                  Changed By          Description
*****
15-APR-2019           [REDACTED]          RFC CHG0078081 [REDACTED] Italian Electronic Invoice used to burst output in to correct folders
-->
<xapi:requestset xmlns:xapi="http://xmlns.oracle.com/oxp/xapi" type="Bursting">
  <xapi:request select="/[REDACTED]_AR_IT_EINV/LIST_G_INVOICE/G_INVOICE">
    <xapi:delivery>
      <xapi:filesystem id="FILE_DELIVERY_XML" output="/oracle_ebs/${INSTANCE}/AR/ItalyElecInv/${FOLDER}/${FILENAME}"/>
    </xapi:delivery>
    <xapi:document output-type="etext" delivery="FILE_DELIVERY_XML">
      <xapi:template type="etext" location="xdo://[REDACTED]_AR_IT_EINV.en.00/?getSource=true">
        </xapi:template>
      </xapi:document>
    </xapi:request>
  </xapi:requestset>
```

Technique #5 – Bursting to XML

- The new solution built was greatly improved over the previous
 - Only 1 process needed to generate all XML files, instead of one per invoice document
 - Run-time was reduced from three hours for a typical batch to 5 minutes
- The Design was scalable for future enhancements. Recently, the business required that a second attachment be added to the generated XML file for each invoice.

Technique #5 – Bursting to XML

- DON'T
 - Be limited by BI Publisher's inability to burst to XML
- DO
 - Utilize native Oracle database XML functions to aggregate and build XML
 - Use BI Publisher bursting to dynamically generate XML



COLLABORATE 20

TECHNOLOGY & APPLICATIONS FORUM
FOR THE ORACLE COMMUNITY

Joe Tseng

jtseng@o2works.com

Session ID:

11607

OATUG
ORACLE APPLICATIONS & TECHNOLOGY USERS GROUP

Remember to complete your evaluation for this session within the app!