Report Builder Basics

This is a quick beginner’s guide to report builder. The example has been tested in both 10.1.x and 11.2.x.

It has example output for HTML, Excel and Text.

If you are trying to create custom coded reports or TcRA reports this is the wrong document!

This document covers Item Reports and Summary Reports created using Report Builder. The example creates a very simple Summary Report, however, there are notes if you are creating an Item Report and the steps for formatting the output are the same so it should still be of use to you.

It is recommended that regardless of you requirement, if you are new to Report Builder you follow this example to create the summary report described, following the sections in the order they are written. Hopefully this will help you understand the principals, steps and provide some useful approaches to building reports.

For more detail see the online documentation for Report Builder, Query Builder and PLM XML/TC XML Export Import Administration.

There are additional Newsletters for report builder, search for Report Builder Workflow Report With Signoff Audit Information, in the solution center, there are two documents, they are the same basic report for TC10.1 and TC11.2. The need for 2 documents for this report is because of the data model of Workflow related objects, not because of differences in the tools.

XSLT processing. It should be noted that the XSLT processor that is used OOTB does not support latest versions of all of XSLT. So you will not be able to use version 2 functions for example.

It is important to note that the XSLT used in the stylesheets is third party therefore you will not find Teamcenter documentation or direct support for this.

Contents:

[Introduction 3](#_Toc476227908)

[Example Report Requirement 4](#_Toc476227909)

[Before You Create the Report - Decisions 4](#_Toc476227910)

[Report Type 4](#_Toc476227911)

[TCXML Vs PLMXML 4](#_Toc476227912)

[Prerequisites - Query or Transfer Mode 5](#_Toc476227913)

[Create Query In Query Builder – Summary Report 5](#_Toc476227914)

[Create Transfer Mode In PLM XML/TC XML Export Import Admin – Item Report ONLY 6](#_Toc476227915)

[Create Report 7](#_Toc476227916)

[Test the Report to Verify Minimum Exported 8](#_Toc476227917)

[Create A Property Set 10](#_Toc476227918)

[Debugging Stylesheet Issues 11](#_Toc476227919)

[Create Style Sheet 12](#_Toc476227920)

[Add Stylesheet Definition to Report 12](#_Toc476227921)

[Basic Entries 14](#_Toc476227922)

[Retrieving Information 15](#_Toc476227923)

[Format Date 18](#_Toc476227924)

[Text Output 19](#_Toc476227925)

[HTML Output 20](#_Toc476227926)

[Excel Output 23](#_Toc476227927)

[Full Style Sheet – Example for Text Output 27](#_Toc476227928)

[Full Style Sheet – Example for HTML Output 29](#_Toc476227929)

[Full Style Sheet – Example for Excel Output 32](#_Toc476227930)

Introduction

Configured Report Builder reports need three main elements:

Input objects

These can be single objects that you select and run a report against (Item Report) or input objects retrieved using a query (Summary Report). Note that an Item Report may be defined for any of the main workspace objects.

Export Definition

Details of the input object(s), any related objects and properties are exported by the report generation in xml format. This maybe a PLMXML or TCXML formatted file. The process uses an optional closure rule and optional property set. For an Item Report the closure rule must be attached to a transfer mode.

NOTE the exact content of a file exported using Report Builder may NOT be the same as the format if you used the same transfer mode, closure rule and property set with a PLMXML export or TCXML export. There may be differences. This is important when you are writing an XSLT definition for the output. You need to look at the report builder generated xml.

An Output Definition (optional)

This is optional if all you want is a tcxml or plmxml file. If you want an alternative then you need to create an xsl stylesheet file that contains XSLT (Extensible Stylesheet Language Transformations) to format the xml. The stylesheet file, named .xsl is imported into report builder as a stylesheet. Example importing is given in this document.

The definition outputs supported are HTML, MSExcel, MSWord, Text and xml.

The XSLT must be version 1 compliant, this is what is supported. This makes the possible functions very limited. You will get format errors if you use unsupported XSLT so make sure what you do use is compliant with version 1.

Text and xml output are often used where the information exported will be consumed in another system and the standard PLMXML or TCXML are not supported formats in the remote system.

If you are producing MSExcel or MSWord there are restrictions on what you can do because the format output will be office 2003 xml. For Excel there is an example in this newsletter. There are also examples shipped with the OOTB product.

Example Report Requirement

In this example the requirement is that the report needs to output information on revisions that have a specification relation to a dataset that has been modified before or after a date that will be supplied when the report is run.

The output required is Item ID, Revision, Date Modified, Date Created, and Status. The output format will be 3 different types:

* HTML
* Delimited text file
* Excel

Before You Create the Report - Decisions

Report Type

As described in the introduction, you need to decide what your input will be. Do you want to select an object then run a report with the selected object as input (Item Report) or do you want to be able to run a query, so have one or more objects as input to the report. The principals are the same for both types but you do need to know which you need! ☺

In this worked example a Summary Report is created, this is used because the requirement we have in this case is to use search criteria to determine input.

NOTE while the example creates a Summary Report, you will find notes that apply if you are creating an Item Report and the steps for formatting the output are the same.

TCXML Vs PLMXML

The formats have very different xml output so you cannot easily change your mind once you have started creating your xsl stylesheet.

The choice between the two formats is mainly a choice of what you are going to do with the output. If you are going to create an XSLT stylesheet to post process then the fact that PLMXML is a more widely used format will not be so relevant. The advantage of TCXML is that it supports the export of more properties.

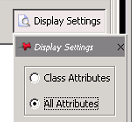
If you want to use PLMXML then make sure you check the xml output before you start to create your xsl file so you are sure that you can access the information you need with this format. This example is PLMXML. The principals are the same for both types. If you are beginner follow this example and then look at the Workflow report newsletter if you want a TCXML example.

Prerequisites - Query or Transfer Mode

You need to create one of these because they are required properties for the report. You need a Transfer Mode if you are going to create an Item Report. Your Transfer mode does not need a closure rule at this stage.

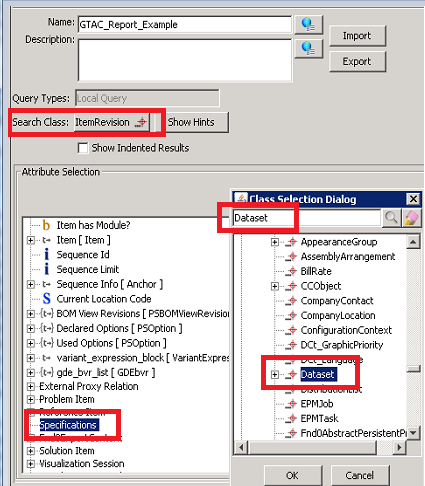
Create Query In Query Builder – Summary Report

When a summary report is run the user will be presented with the input criteria you define in the Query. The Search class you select will be the objects that are submitted to the export.

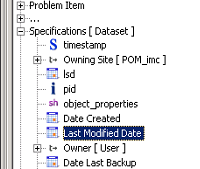
In this case Item Revision objects are the required input to the report so this is selected as the search class. Before you select criteria, select All Attributes from the Display Settings: 

For the search criteria we need to select a dataset that is related using a specification relation:

* Double click Specifications
* Type in dataset in the search box for the class selection and hit enter
* Select Dataset from the list

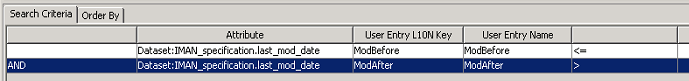


Now double click Last Modified Date from the Dataset properties.



Do this twice to add lines for before and after the date.

* You should update the User Entry L10N Key so that there are two different labels for the input, the default will be the property name, so both would be the same.
* You should also update the operator so that you are search for dates less than for one and greater than for the other. At least one of these should also be equal to.



NOTE in the image above there is nothing in the first column of the first row, this is just automatic when you save. When you first add the line there will be an AND in this cell, you can ignore it.

Create Transfer Mode In PLM XML/TC XML Export Import Admin – Item Report ONLY

If you are creating an Item report then at this stage all you need to do is create a transfer mode of the correct type. Make sure your Transfer Mode is of the same type as your report, TCXML or PLMXML. You do not need a closure rule at this stage.

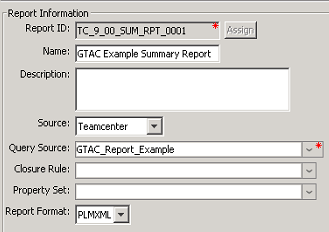
Create Report

Create the report now because it allows us to see what is exported for no further effort – it may be enough without a closure rule or property set! ☺

From Report Builder use the menu on the top bar and select File -> Create Report

Select the type – in this case Summary Report.

* On the next screen you can type in a unique ID or you can assign (and this is a TC10.1 screen shot, do not be confused by the report ID)
* Select the Query you just created. Note you can use an OOTB query or any existing query if it suits your requirement
* Enter a display Name
* This example uses the default format of PLMXML



If you are creating an Item report you will need to pick the class, this is the object type the user is going to select to run the report against. The Transfer Mode is the only other requirement. This can be a Transfer mode with no closure rule.

Just select Finish at this point, you do not need a stylesheet at this stage, in fact it is better not to have one until you have finished developing the export, including the closure rule if you need to create one. With no style sheet you can easily capture the xml output, see below.

Test the Report to Verify Minimum Exported

Testing before applying a closure rule or property set will allow you to check what is the minimum that is exported for you basic report. For a summary report:

From RAC, go to **My Teamcenter** and select **Tools** -> **Reports** -> **Report Builder Reports**

Select your summary report from the list

Enter search criteria that will return representative objects, without giving you a huge output file.

For an Item report, select a valid object, right key menu to select Generate Report.

When you run the report with no defined stylesheets you simply generate the xml file.

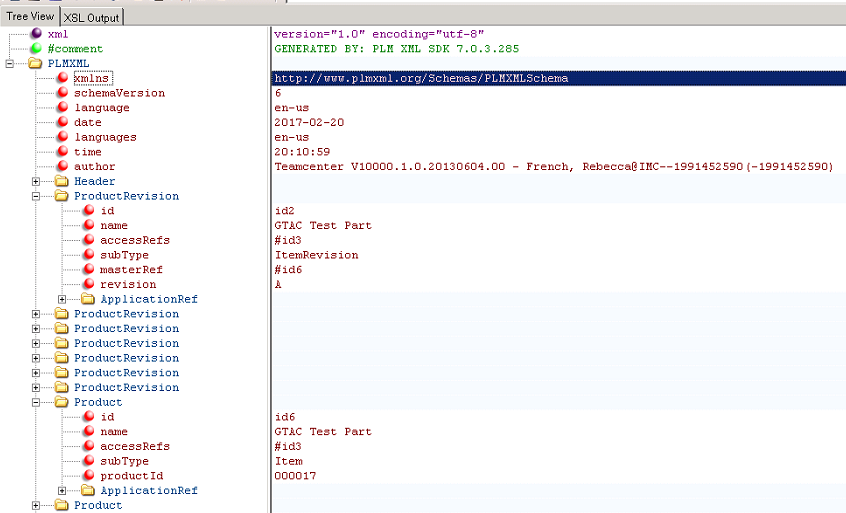
NOTE that the output for a report may NOT be the same as you get if you use command line or interface Export actions, so it is important you run the report with no stylesheet to verify the xml file.

Depending on your settings the xml may open automatically in a browser. This is not necessarily the best tool. It may be helpful to install Microsoft’s XML Notepad (https://www.microsoft.com/en-gb/download/details.aspx?id=7973) and view the output. So save the file to a convenient location and name and open from there in XML Notepad. Once complete keep a copy to help when you are creating the stylesheet

Look to see what objects have been exported, in this example the objects are OOTB Item Revisions as the input. The input object will clearly affect the output xml. So for the example, you should see the following types of top level elements:

|  |  |
| --- | --- |
|  | So the objects that are likely to be of interest in the output are obvious but just in case ☺:  ProductRevision – Represents ItemRevision  Product – Represents Item  DataSet – Dataset  While generally you can guess the mapping, the location for the schema files is normally documented in the Basic concepts for using PLMXML section of the PLM XML/TC XML Export Import Administration manual. |

To see what properties are exported open up the appropriate elements.



There are not enough properties. The requirement was:

Item ID, Revision, Date Modified, Date Created, and Status.

For the dates these should be accessible from the ItemRevision, the Status is actually on a referenced object, Release Status. This might normally imply that a closure rule is needed and where you have to navigate to another object this often the case. For typed references you will often find that you should be able get the related objects exported just using a property set.

NOTE there may be some properties you will struggle to export because of your data model knowledge, but also note that not every property you see on an object is exportable, many properties are run time and these cannot be directly exported. The code derives the values of a runtime property so you would need to be able to create a closure rule or property set that will navigate to the related object which “owns” the property. However, some runtime properties are not just simple navigations to a related object, so cannot be accessed at all.

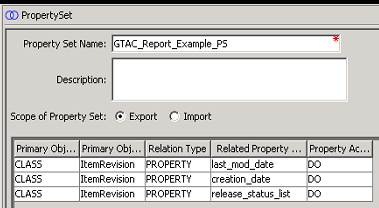
Create A Property Set – (PLM XML / TC XML Export Import Administration)

For each of the following properties create an entry for class ItemRevision,

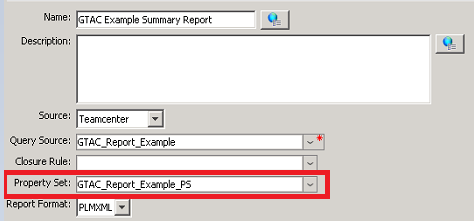
last\_mod\_date

creation\_date

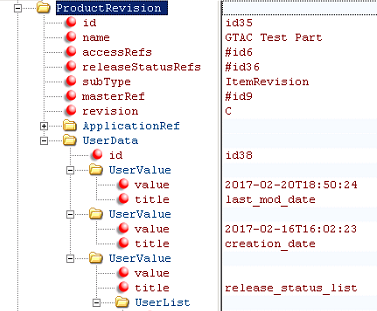
release\_status\_list



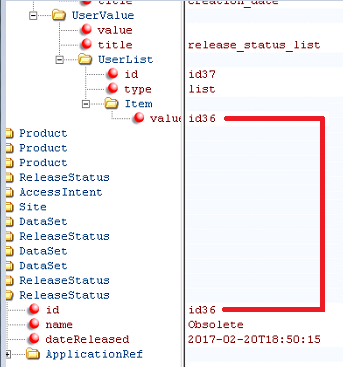
Update the report to use the property set:



Run the report and look at the output:



You can see that these additional properties appear under the UserData element. There is a UserValue for each of them. For the dates the value appears as an attribute. For the release status there is an additional element, UserList. This does not contain a value, instead it contains a reference in an Item element. This reference uses an identifier of another object within the file. Note that these identifiers are generated at the time of the export so they apply to only this file.



This is important when you come to create the xsl.

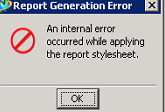
Debugging Stylesheet Issues

The main issue you get is with the XSLT. The main reasons are either incorrect format, or unsupported functions (it cannot be said often enough, version 1).

The best advice you can have is to make sure you do very small updates to your stylesheet in between updating the named reference on the dataset and doing a test.

The general rule is use only XSLT version 1 in addition if you are supporting the office output formats use only what is supported for 2003.

If you have unsupported XSLT (including typos) you will get the following error in RAC:



If you are using a 2-tier client you might see something similar to the following in the TAO window:

XSLT The property ' transform Transforming... ' is not a valid QName. (02032017\_1316\_2868\_GTAC\_Rev\_Report\_Example\_Text.xsl, Error: 29)

In either client in the syslog you will see:

REPORT ERROR:Error returned by Xalan transformer in REP\_\_transform\_xml ITK. SAXParseException: (02032017\_1316\_2868\_GTAC\_Rev\_Report\_Example\_Text.xsl, line 29, column 10)

In this case this error is a missing “ on line 28. And the number after Error does normally indicate that the issue is on that line or the one above. BUT if it’s a failure to close something, it’s possible the error is picked up on the open line for the element.

Create Style Sheet

This may be obvious but **before** you create the stylesheet make sure you have done the following:

* Identified a known set of test data that contains representative data with known values so you can validate your output
* Confirmed the output xml from the report contains all of the required information
* Where information is on a related object check that the output contains sufficient links so you can navigate to it

Below you will find examples for text, HTML, and Excel.

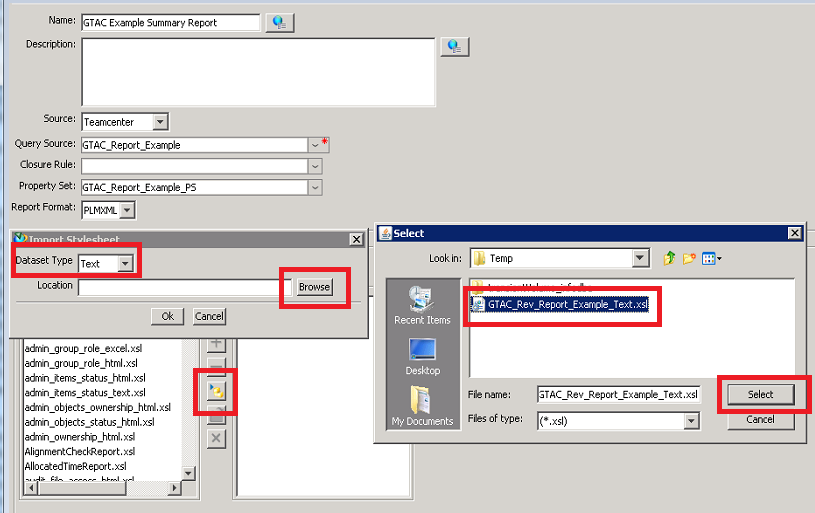
The method to retrieve the information is the same in all cases, it is only the formatting that differs.

Add Stylesheet Definition to Report

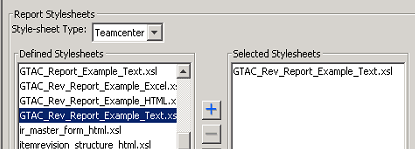
Do not try to create the whole stylesheet in on go, it will make it much harder to find errors. Do it iteratively. For initial report for example just stick to first few lines. See section Basic Entries on page 14, and use just the entries in that section to start you off. Put these lines in a file with descriptive name and extension xsl. This will give you an empty page when you run the report but it will give you something you can start with.

From the Report Data tab.

* Select the import stylesheet icon from the stylesheet section.
* Select the Report Format (Text, HTML etc)
* Browse to select your basic xsl file.



Find the imported template on the list and Add it to the report, select Modify to update



Once you have added it to the report and created the dataset, you will need to update it by updating the named reference from the Dataset. The name of the dataset will be the original file name, as you see in the list above, the Dataset type will be of the form Crf<Format>Stylesheet, e.g. CrfTextStylesheet or CrfHtmlStylesheet or CrfExcelStylesheet

Basic Entries

There are number of possible entries you may see before the first template statement. Shown here are the basic ones. For more examples see the OOTB stylesheets which are contained in TC\_DATA in the crf\resources directory.

|  |  |
| --- | --- |
| <?xml version="1.0" encoding="utf-8" ?> | XML prolog contains the version and the encoding. |
| <xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  xmlns:plm="http://www.plmxml.org/Schemas/PLMXMLSchema"> | The xsl:stylesheet element gives the definition.  NOTE that the version is 1this is the maximum you can use and limits the functions you can use, you cannot make it process 2.0 by updating this, the restriction is the processor.  The xmlns entries are name space entries that define prefixes for each of the extension elements. This enables the processor to validate the input. For HTML and Text these will just be basic xsl and the plm schema. There is a tcxml schema if that is what you are using. Again you cannot make the process understand something just by adding it here, there are restrictions. See the examples below and the OOTB stylesheets. |
| <xsl:output method="text" version="1.0" indent="yes"/> | This defines the output. NOTE it is not directly related to what you are exporting, for example for Excel it is NOT excel, it is xml. See each example. Note this element is closed directly using the / at the end. |
| </xsl:stylesheet> | This appears at the end. All elements must be closed. If they are nested, the close statement will appear at the end of the section they enclose.  All elements are normally nested within the stylesheet element. So this is normally the last line of the file. |

Retrieving Information

The following applies for all output formats. Note that the navigation principles are the same for TCXML but because the structure of the xml file is different the implementation is different, see the OOTB examples or see the Report Builder Workflow Report With Signoff Audit Information newsletter which covers TCXML output.

|  |  |
| --- | --- |
| <xsl:template match="/"> | The template element contains the main definition that will process the input. The match attribute associates this template to part of the input document.  In this case match=”/” associates it to all of the input.  This element must be closed at the end of the template code. |
| <xsl:for-each select="/plm:PLMXML/plm:ProductRevision"> | The file may contain one or many ItemRevisions. So there needs to be a loop.  The plm: is to indicate which schema this applies to, as defined in the xmlns entries at the start of the stylesheet.  The structure must match the input xml. |
| ………..  ………..  ………..  ………..  ……….. | Inside the for each element you need to write the XSLT to get the values for each revision, see next few pages for detail of what goes here |
| </xsl:for-each>  </xsl:template> | Close the for each and template elements |

Start by looking at the xml so you know what you need to get and how you might navigate to it.

|  |  |
| --- | --- |
|  | Looking at the input xml you see that relative to the ProductRevision element the revision property is easy retrieve, there is a revision attribute with the value.  For last modified date it a little more complicated, relative to the ProductRevision it is 2 levels down.  UserData -> UserValue, you will also need to make some distinction between the two different UserValue elements.  It is likely the timestamp string will need to be formatted into something that will be better as a display value.  For released status and item ID the values are not contained in the element at all, there is an id number reference. |
|  | To get the item id there will need to be a search for the value of the masterRef attribute, without the # at the start  The id should be in an element PLMXML -> Product as attribute id.  Then the value that is actually wanted is on attribute productId for that Product element.  It is a similar navigation for the release status but for this the element is PLMXM -> ReleaseStatus, the attribute on ReleaseStatus is dateReleased |

Then inside the for each loop you can fill in the XSLT to get the values you need.

|  |  |
| --- | --- |
| <xsl:variable name="revision" select="./@revision"/> | Create a variable named revision to hold the value and navigate from the current location to the attribute revision. The @ indicates this is an attribute not an element in the path.  Current location is dependent on the enclosing element, which is the for each for the ProductRevision elements |
| <xsl:variable name="itemXLID" select="substring-after(./@masterRef,'#')"/> | Using a XSLT version 1 function the # can be removed from the attribute masterRef |
| <xsl:variable name="item" select="/plm:PLMXML/plm:Product[@id=$itemXLID]"/> | Search for an element with has an attribute id that matches the id that is set in the variable. |
| <xsl:variable name="itemID" select="$item/@productId"/> | Set a variable itemID to the attribute productId from the item found on the previous line. |
| <xsl:variable name="statusXLID"  select="substring-after(./@releaseStatusRefs,'#')"/>  <xsl:variable name="statusObj"  select="/plm:PLMXML/plm:ReleaseStatus[@id=$statusXLID]"/>  <xsl:variable name="status" select="$statusObj/@name"/> | Using the same principal get the value for the status. |

Note for the some of these steps you could use a more complex line and not bother creating so many variables however, this leads more complex navigations that give you much more scope to just get nothing. This is what you get, if the format is valid but the navigation is not then it will not error, it will just give you a null value. So breaking is down is normally safer.

Format Date

The following is one method you could use to format the timestamp value into something that is better for display. There may be a neater method. You will see below I have used substring-before and substring-after which makes it a bit clumsy as it needs to be broken down gradually. The other thing to consider is that you need to do this twice, because there are two dates to format. To make it reusable a template is created.

|  |  |
| --- | --- |
| <xsl:variable name="last\_mod\_date">  <xsl:call-template name="sortDate">  <xsl:with-param name="inDate"  select="./plm:UserData/  plm:UserValue[@title='last\_mod\_date']/@value"/>  </xsl:call-template>  </xsl:variable> | Set a variable, only this time you cannot create and close the element on the same line  Instead of directly setting a value call a template.  The template needs some input, the element with-param is used to give a name and value, there can be multiple inputs, here we just need the date.  Close the template and variable elements |

The template sortDate is defined after the close for the main template that is being used to fetch the xml input. The value-of element at the end of the template defines what will be passed back.

<xsl:template name="sortDate">

<xsl:param name="inDate" />

<xsl:variable name="datebit1" select="substring-before($inDate,'T')"/>

<xsl:variable name="datebit2" select="substring-after($datebit1,'-')"/>

<xsl:variable name="yearbit" select="substring-before($datebit1,'-')"/>

<xsl:variable name="monthbit" select="substring-before($datebit2,'-')"/>

<xsl:variable name="daybit" select="substring-after($datebit2,'-')"/>

<xsl:variable name="timebit1" select="substring-after($inDate,'T')"/>

<xsl:variable name="timebit2" select="substring-after($timebit1,':')"/>

<xsl:variable name="hourbit" select="substring-before($timebit1,':')"/>

<xsl:variable name="minbit" select="substring-before($timebit2,':')"/>

<xsl:value-of select="concat($daybit,'-',$monthbit,'-',$yearbit,' ',$hourbit,':',$minbit)"/>

</xsl:template>

NOTE in practice you would not get all of the values then do the output. You would put something in the stylesheet to display things as you go along.

Text Output

For our text output the requirement is a delimited line of values for each revision, using ~ as the delimiter.

The output is of the form:

000017~A~20-02-2017 18:50~16-02-2017 15:34~Approved

000017~B~20-02-2017 18:50~16-02-2017 15:53~Rejected

000017~E~20-02-2017 18:50~16-02-2017 16:11~

000018~A~20-02-2017 18:50~16-02-2017 22:56~

000017~D~20-02-2017 18:50~16-02-2017 16:08~Pending

000017~C~20-02-2017 18:50~16-02-2017 16:02~Obsolete

000018~B~20-02-2017 18:50~20-02-2017 18:50~

000019~A~21-02-2017 15:40~21-02-2017 15:40~

So after all the variables are set (see above, section Retrieving Information), but before the end of the for each, add the following.

<xsl:value-of select="$itemID"/>~<xsl:value-of select="$revision"/>~<xsl:value-of select="$last\_mod\_date"/>~<xsl:value-of select="$creation\_date"/>~<xsl:value-of select="$status"/>

<xsl:text>

</xsl:text>

So for text use value-of elements and the variable name, with the delimiter in between.

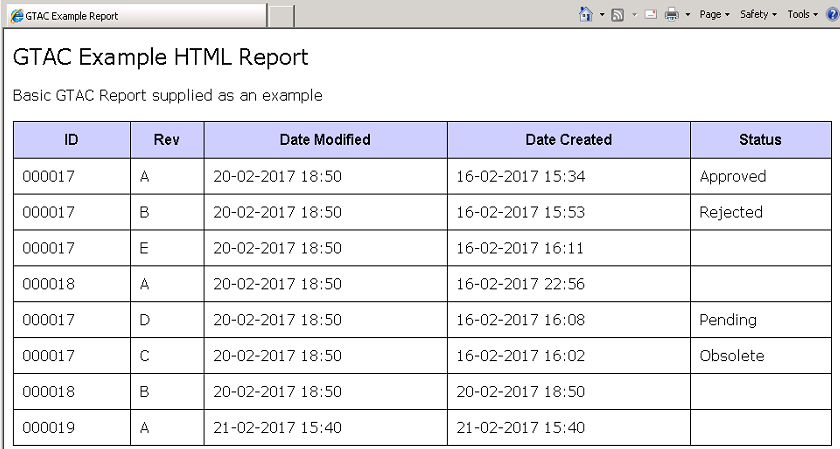
NOTE that the final text element is to make sure there is a new line for each entry.

The full text stylesheet is given in a later section, below.

HTML Output

The data retrieval is just same, the rest of the information required in the stylesheet is the formatting.

This is an example of the output:

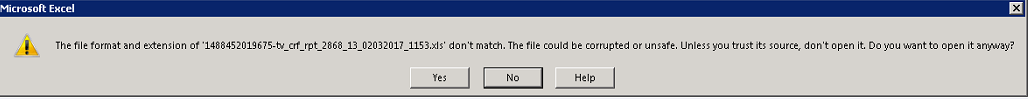


|  |  |  |
| --- | --- | --- |
| <xsl:output method="html" indent="yes"/> | The output is set to html, this important to make sure that the processor will correctly interpret the html tags you will put in the stylesheet | |
| <xsl:template match="/">  <html>  <head>  <title>GTAC Example Report</title>  <style>  ………..  </style>  </head>  <body>    <p class="pagetitle">  <left>GTAC Example HTML Report</left>  </p>  <p class="standard">  Basic GTAC Report supplied as an example  </p> | Once the template has started the next lines are all effectively static HTML. How you format is obviously up to you. In the example it uses style definitions, for the full input see the full stylesheet at the end of the document.  In addition to the head, the tags for the body and the title and any fixed text are added.  NOTE that when you are creating ordinary html you will get away with not closing some tags. You will not in XSLT, they must be correct for any HTML that is going to be parsed by the XSLT processor, so the body tag must be closed before the end of the template. Also note you cannot start a HTML tag in one template and close in another. The start and end must be with in the same template. | |
| <table class="main">  <tr>  <th class="main">ID</th>  <th class="main">Rev</th>  <th class="main">Date Modified</th>  <th class="main">Date Created</th>  <th class="main">Status</th>  </tr> | Create the table, and the header row. | |
| <xsl:for-each select="/plm:PLMXML/plm:ProductRevision">  ……….. | | Next do the for each loop for the revisions and set all of the variables, as described in the previous sections. |
| <tr>  <td class="main"><xsl:value-of select="$itemID"/></td>  <td class="main"><xsl:value-of select="$revision"/></td>  <td class="main"><xsl:value-of select="$last\_mod\_date"/></td>  <td class="main"><xsl:value-of select="$creation\_date"/></td>  <td class="main"><xsl:value-of select="$status"/></td>  </tr>  </xsl:for-each> | | Once all the variables are set then they can be printed out.  This obviously needs to be done inside the for each loop.  Just use value-of inside the correct html tags. |
| </table>  </body>  </html> | | Then close out the table, body and html tags |
|  | | Close the template and add the date format template to the stylesheet, as outlined in the section above. |

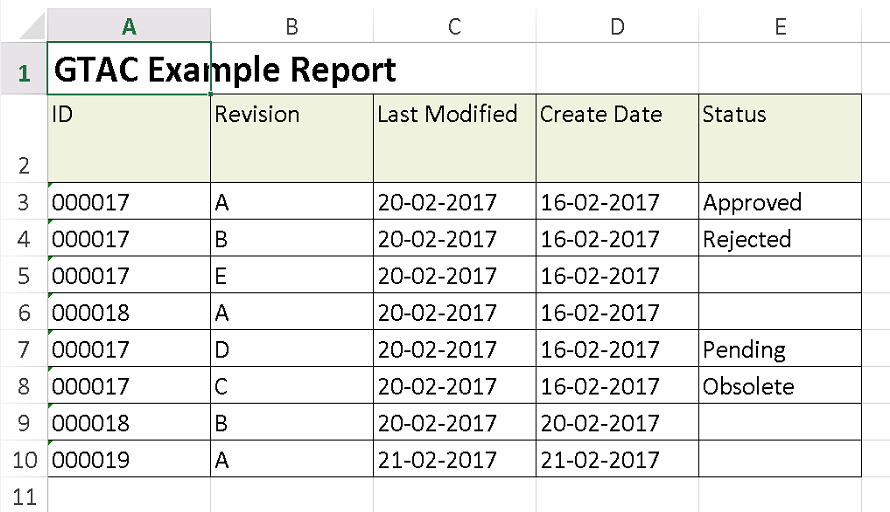
Excel Output

The data retrieval is just same, the rest of the information required in the stylesheet is the formatting.

When you run the report and output to Excel you will get an error, this is expected and you just need to confirm Yes.



This is an example of the output:



This are a lot more lines required to format the Excel spreadsheet than the other formats. Again there are lots of ways to do this.

|  |  |  |
| --- | --- | --- |
| <?xml version="1.0" encoding="utf-8" ?>  <xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  xmlns:plm="http://www.plmxml.org/Schemas/PLMXMLSchema"  xmlns="urn:schemas-microsoft-com:office:spreadsheet"  xmlns:o="urn:schemas-microsoft-com:office:office"  xmlns:x="urn:schemas-microsoft-com:office:excel"  xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"  xmlns:html="http://www.w3.org/TR/REC-html40"  xmlns:crf="http://ExternalFunction.setFile"> | | The initial stylesheet element has to be different for Excel. We add additional namespaces. Note for this example they are not all used, but it gives a good start for any stylesheets you create for Excel output. |
| <xsl:output method="xml" version="1.0" indent="yes" /> | | You do not use an output method of excel, you use xml.  This is because what is sent to Excel is not really xls, Teamcenter gives the file that extension so that it is launched in Excel, but the file format is actually 2003 xml.  And this fact is important for the supported Excel elements that you can use. |
| <xsl:strip-space elements="\*"/> | | Stripping extra white space may be desirable for any format, for Excel it can be more of an issue if you don’t. HTML for example will not care. |
| <xsl:template match="/">  <xsl:processing-instruction name="mso-application">progid="Excel.Sheet"</xsl:processing-instruction>  <Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"  xmlns:o="urn:schemas-microsoft-com:office:office"  xmlns:x="urn:schemas-microsoft-com:office:excel"  xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"  xmlns:html="http://www.w3.org/TR/REC-html40"> | All of this is used to format the first few lines of the xml output, this is required to make it valid for Excel to read in. | |
| <Styles>  ………..  ………..  </Styles> | Then open a Styles tag. This and the contents, is written directly into the ouput xml. It’s just to format the cells in Excel. As with the HTML using styles is just a choice to make it easier to format the content. For the full contents see the example at the end of the document. This is followed by more fixed text for the start of the worksheet and the first rows, see below. | |
| <Worksheet ss:Name="Example">  <Table ss:DefaultRowHeight="15">    <Column ss:AutoFitWidth="0" ss:Width="69.75" ss:Span="7"/>  <Row ss:Height="21">  <Cell ss:StyleID="Title1"><Data ss:Type="String">GTAC Example Report</Data></Cell>  </Row>    <Row ss:AutoFitHeight="0" ss:Height="36">  <Cell ss:StyleID="TableHeader"><Data ss:Type="String">ID</Data></Cell>  <Cell ss:StyleID="TableHeader"><Data ss:Type="String">Revision</Data></Cell>  <Cell ss:StyleID="TableHeader"><Data ss:Type="String">Last Modified</Data></Cell>  <Cell ss:StyleID="TableHeader"><Data ss:Type="String">Create Date</Data></Cell>  <Cell ss:StyleID="TableHeader"><Data ss:Type="String">Status</Data></Cell>  </Row> | | |
| <xsl:for-each select="/plm:PLMXML/plm:ProductRevision"> ………..  ……….. | Then the for each is entered with all of the variable setting as given in the pages above. | |
| After the variables are all set then the data row is entered followed by the close for the for each loop. | | |
| <Row>  <Cell ss:StyleID="TableEntry"><Data ss:Type="String"><xsl:value-of select="$itemID"/></Data></Cell>  <Cell ss:StyleID="TableEntry"><Data ss:Type="String"><xsl:value-of select="$revision"/></Data></Cell>  <Cell ss:StyleID="TableEntry"><Data ss:Type="String"><xsl:value-of select="$last\_mod\_date"/></Data></Cell>  <Cell ss:StyleID="TableEntry"><Data ss:Type="String"><xsl:value-of select="$creation\_date"/></Data></Cell>  <Cell ss:StyleID="TableEntry"><Data ss:Type="String"><xsl:value-of select="$status"/></Data></Cell>  </Row>  </xsl:for-each>  </Table> | | |
| <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">  <Unsynced/>  <Selected/>  <ProtectObjects>False</ProtectObjects>  <ProtectScenarios>False</ProtectScenarios>  </WorksheetOptions>  </Worksheet>  </Workbook> | WorksheetOptions does what you expect.  The worksheet and workbook are then closed out.  You need to add the close for the template and the template for the date format at the end. | |

Full Style Sheet – Example for Text Output

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

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:plm="http://www.plmxml.org/Schemas/PLMXMLSchema">

<xsl:output method="text" version="1.0" indent="yes"/>

<xsl:template match="/">

<xsl:for-each select="/plm:PLMXML/plm:ProductRevision">

<xsl:variable name="itemXLID" select="substring-after(./@masterRef,'#')"/>

<xsl:variable name="item" select="/plm:PLMXML/plm:Product[@id=$itemXLID]"/>

<xsl:variable name="itemID" select="$item/@productId"/>

<xsl:variable name="revision" select="./@revision"/>

<xsl:variable name="statusXLID" select="substring-after(./@releaseStatusRefs,'#')"/>

<xsl:variable name="statusObj" select="/plm:PLMXML/plm:ReleaseStatus[@id=$statusXLID]"/>

<xsl:variable name="status" select="$statusObj/@name"/>

<xsl:variable name="last\_mod\_date">

<xsl:call-template name="sortDate">

<xsl:with-param name="inDate" select="./plm:UserData/plm:UserValue[@title='last\_mod\_date']/@value"/>

</xsl:call-template>

</xsl:variable>

<xsl:variable name="creation\_date">

<xsl:call-template name="sortDate">

<xsl:with-param name="inDate" select="./plm:UserData/plm:UserValue[@title='creation\_date']/@value"/>

</xsl:call-template>

</xsl:variable>

<xsl:value-of select="$itemID"/>~<xsl:value-of select="$revision"/>~<xsl:value-of select="$last\_mod\_date"/>~<xsl:value-of select="$creation\_date"/>~<xsl:value-of select="$status"/>

<xsl:text>

</xsl:text>

</xsl:for-each>

</xsl:template>

<xsl:template name="sortDate">

<xsl:param name="inDate" />

<xsl:variable name="datebit1" select="substring-before($inDate,'T')"/>

<xsl:variable name="datebit2" select="substring-after($datebit1,'-')"/>

<xsl:variable name="yearbit" select="substring-before($datebit1,'-')"/>

<xsl:variable name="monthbit" select="substring-before($datebit2,'-')"/>

<xsl:variable name="daybit" select="substring-after($datebit2,'-')"/>

<xsl:variable name="timebit1" select="substring-after($inDate,'T')"/>

<xsl:variable name="timebit2" select="substring-after($timebit1,':')"/>

<xsl:variable name="hourbit" select="substring-before($timebit1,':')"/>

<xsl:variable name="minbit" select="substring-before($timebit2,':')"/>

<xsl:value-of select="concat($daybit,'-',$monthbit,'-',$yearbit,' ',$hourbit,':',$minbit)"/>

</xsl:template>

</xsl:stylesheet>

Full Style Sheet – Example for HTML Output

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

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:plm="http://www.plmxml.org/Schemas/PLMXMLSchema">

<xsl:output method="html" indent="yes"/>

<xsl:template match="/">

<html>

<head>

<title>GTAC Example Report</title>

<style>

table.main {

border-collapse: collapse;

border-spacing: 0pt;

width: 100%;

empty-cells: show;

border-color:#000000;

}

td.main {

border-style:solid;

border-width:0.2pt;

border-color:#000000;

font-family: Verdana;

font-size: 12pt;

padding: 10px;

}

th.main {

border-style:solid;

border-width:0.2pt;

border-color:#000000;

background-color: #CCCCFF;

font-family: Arial, Helvetica, sans-serif;

padding: 10px;

}

p.pagetitle {

font-family: Verdana;

font-size: 18pt;

}

p.standard {

font-family: Verdana;

font-size: 12pt;

}

</style>

</head>

<body>

<p class="pagetitle">

<left>GTAC Example HTML Report</left>

</p>

<p class="standard">

Basic GTAC Report supplied as an example

</p>

<table class="main">

<tr>

<th class="main">ID</th>

<th class="main">Rev</th>

<th class="main">Date Modified</th>

<th class="main">Date Created</th>

<th class="main">Status</th>

</tr>

<xsl:for-each select="/plm:PLMXML/plm:ProductRevision">

<xsl:variable name="itemXLID" select="substring-after(./@masterRef,'#')"/>

<xsl:variable name="item" select="/plm:PLMXML/plm:Product[@id=$itemXLID]"/>

<xsl:variable name="itemID" select="$item/@productId"/>

<xsl:variable name="revision" select="./@revision"/>

<xsl:variable name="statusXLID" select="substring-after(./@releaseStatusRefs,'#')"/>

<xsl:variable name="statusObj" select="/plm:PLMXML/plm:ReleaseStatus[@id=$statusXLID]"/>

<xsl:variable name="status" select="$statusObj/@name"/>

<xsl:variable name="last\_mod\_date">

<xsl:call-template name="sortDate">

<xsl:with-param name="inDate" select="./plm:UserData/plm:UserValue[@title='last\_mod\_date']/@value"/>

</xsl:call-template>

</xsl:variable>

<xsl:variable name="creation\_date">

<xsl:call-template name="sortDate">

<xsl:with-param name="inDate" select="./plm:UserData/plm:UserValue[@title='creation\_date']/@value"/>

</xsl:call-template>

</xsl:variable>

<tr>

<td class="main"><xsl:value-of select="$itemID"/></td>

<td class="main"><xsl:value-of select="$revision"/></td>

<td class="main"><xsl:value-of select="$last\_mod\_date"/></td>

<td class="main"><xsl:value-of select="$creation\_date"/></td>

<td class="main"><xsl:value-of select="$status"/></td>

</tr>

</xsl:for-each>

</table>

</body>

</html>

</xsl:template>

<xsl:template name="sortDate">

<xsl:param name="inDate" />

<xsl:variable name="datebit1" select="substring-before($inDate,'T')"/>

<xsl:variable name="datebit2" select="substring-after($datebit1,'-')"/>

<xsl:variable name="yearbit" select="substring-before($datebit1,'-')"/>

<xsl:variable name="monthbit" select="substring-before($datebit2,'-')"/>

<xsl:variable name="daybit" select="substring-after($datebit2,'-')"/>

<xsl:variable name="timebit1" select="substring-after($inDate,'T')"/>

<xsl:variable name="timebit2" select="substring-after($timebit1,':')"/>

<xsl:variable name="hourbit" select="substring-before($timebit1,':')"/>

<xsl:variable name="minbit" select="substring-before($timebit2,':')"/>

<xsl:value-of select="concat($daybit,'-',$monthbit,'-',$yearbit,' ',$hourbit,':',$minbit)"/>

</xsl:template>

</xsl:stylesheet>

Full Style Sheet – Example for Excel Output

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

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:plm="http://www.plmxml.org/Schemas/PLMXMLSchema"

xmlns="urn:schemas-microsoft-com:office:spreadsheet"

xmlns:o="urn:schemas-microsoft-com:office:office"

xmlns:x="urn:schemas-microsoft-com:office:excel"

xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"

xmlns:html="http://www.w3.org/TR/REC-html40"

xmlns:crf="http://ExternalFunction.setFile">

<xsl:output method="xml" version="1.0" indent="yes" />

<xsl:strip-space elements="\*"/>

<xsl:template match="/">

<xsl:processing-instruction name="mso-application">progid="Excel.Sheet"</xsl:processing-instruction>

<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"

xmlns:o="urn:schemas-microsoft-com:office:office"

xmlns:x="urn:schemas-microsoft-com:office:excel"

xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"

xmlns:html="http://www.w3.org/TR/REC-html40">

<Styles>

<Style ss:ID="Default" ss:Name="Normal">

<Alignment ss:Vertical="Bottom"/>

<Borders/>

<Font ss:FontName="Calibri" x:Family="Swiss" ss:Size="11" ss:Color="#000000"/>

<Interior/>

<NumberFormat/>

<Protection/>

</Style>

<Style ss:ID="TableHeader">

<Alignment ss:Vertical="Top" ss:WrapText="1"/>

<Borders>

<Border ss:Position="Bottom" ss:LineStyle="Continuous" ss:Weight="1"/>

<Border ss:Position="Left" ss:LineStyle="Continuous" ss:Weight="1"/>

<Border ss:Position="Right" ss:LineStyle="Continuous" ss:Weight="1"/>

<Border ss:Position="Top" ss:LineStyle="Continuous" ss:Weight="1"/>

</Borders>

<Interior ss:Color="#EAF1DD" ss:Pattern="Solid"/>

</Style>

<Style ss:ID="TableEntry">

<Alignment ss:Vertical="Top" ss:WrapText="1"/>

<Borders>

<Border ss:Position="Bottom" ss:LineStyle="Continuous" ss:Weight="1"/>

<Border ss:Position="Left" ss:LineStyle="Continuous" ss:Weight="1"/>

<Border ss:Position="Right" ss:LineStyle="Continuous" ss:Weight="1"/>

<Border ss:Position="Top" ss:LineStyle="Continuous" ss:Weight="1"/>

</Borders>

</Style>

<Style ss:ID="TableDateEntry">

<Alignment ss:Vertical="Top"/>

<Borders>

<Border ss:Position="Bottom" ss:LineStyle="Continuous" ss:Weight="1"/>

<Border ss:Position="Left" ss:LineStyle="Continuous" ss:Weight="1"/>

<Border ss:Position="Right" ss:LineStyle="Continuous" ss:Weight="1"/>

<Border ss:Position="Top" ss:LineStyle="Continuous" ss:Weight="1"/>

</Borders>

<NumberFormat ss:Format="dd/mm/yyyy;@"/>

</Style>

<Style ss:ID="Title1">

<Font ss:FontName="Calibri" x:Family="Swiss" ss:Size="16" ss:Color="#000000" ss:Bold="1"/>

</Style>

<Style ss:ID="Title2">

<Font ss:FontName="Calibri" x:Family="Swiss" ss:Size="14" ss:Color="#000000" ss:Bold="1"/>

</Style>

<Style ss:ID="DateFormat1">

<Font ss:FontName="Calibri" x:Family="Swiss" ss:Size="12" ss:Color="#000000"/>

<NumberFormat ss:Format="dd/mm/yyyy;@"/>

</Style>

</Styles>

<Worksheet ss:Name="Example">

<Table ss:DefaultRowHeight="15">

<Column ss:AutoFitWidth="0" ss:Width="69.75" ss:Span="7"/>

<Row ss:Height="21">

<Cell ss:StyleID="Title1"><Data ss:Type="String">GTAC Example Report</Data></Cell>

</Row>

<Row ss:AutoFitHeight="0" ss:Height="36">

<Cell ss:StyleID="TableHeader"><Data ss:Type="String">ID</Data></Cell>

<Cell ss:StyleID="TableHeader"><Data ss:Type="String">Revision</Data></Cell>

<Cell ss:StyleID="TableHeader"><Data ss:Type="String">Last Modified</Data></Cell>

<Cell ss:StyleID="TableHeader"><Data ss:Type="String">Create Date</Data></Cell>

<Cell ss:StyleID="TableHeader"><Data ss:Type="String">Status</Data></Cell>

</Row>

<xsl:for-each select="/plm:PLMXML/plm:ProductRevision">

<xsl:variable name="itemXLID" select="substring-after(./@masterRef,'#')"/>

<xsl:variable name="item" select="/plm:PLMXML/plm:Product[@id=$itemXLID]"/>

<xsl:variable name="itemID" select="$item/@productId"/>

<xsl:variable name="revision" select="./@revision"/>

<xsl:variable name="statusXLID" select="substring-after(./@releaseStatusRefs,'#')"/>

<xsl:variable name="statusObj" select="/plm:PLMXML/plm:ReleaseStatus[@id=$statusXLID]"/>

<xsl:variable name="status" select="$statusObj/@name"/>

<xsl:variable name="last\_mod\_date">

<xsl:call-template name="sortDate">

<xsl:with-param name="inDate" select="./plm:UserData/plm:UserValue[@title='last\_mod\_date']/@value"/>

</xsl:call-template>

</xsl:variable>

<xsl:variable name="creation\_date">

<xsl:call-template name="sortDate">

<xsl:with-param name="inDate" select="./plm:UserData/plm:UserValue[@title='creation\_date']/@value"/>

</xsl:call-template>

</xsl:variable>

<Row>

<Cell ss:StyleID="TableEntry"><Data ss:Type="String"><xsl:value-of select="$itemID"/></Data></Cell>

<Cell ss:StyleID="TableEntry"><Data ss:Type="String"><xsl:value-of select="$revision"/></Data></Cell>

<Cell ss:StyleID="TableEntry"><Data ss:Type="String"><xsl:value-of select="$last\_mod\_date"/></Data></Cell>

<Cell ss:StyleID="TableEntry"><Data ss:Type="String"><xsl:value-of select="$creation\_date"/></Data></Cell>

<Cell ss:StyleID="TableEntry"><Data ss:Type="String"><xsl:value-of select="$status"/></Data></Cell>

</Row>

</xsl:for-each>

</Table>

<WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">

<Unsynced/>

<Selected/>

<ProtectObjects>False</ProtectObjects>

<ProtectScenarios>False</ProtectScenarios>

</WorksheetOptions>

</Worksheet>

</Workbook>

</xsl:template>

<xsl:template name="sortDate">

<xsl:param name="inDate" />

<xsl:variable name="datebit1" select="substring-before($inDate,'T')"/>

<xsl:variable name="datebit2" select="substring-after($datebit1,'-')"/>

<xsl:variable name="yearbit" select="substring-before($datebit1,'-')"/>

<xsl:variable name="monthbit" select="substring-before($datebit2,'-')"/>

<xsl:variable name="daybit" select="substring-after($datebit2,'-')"/>

<xsl:variable name="timebit1" select="substring-after($inDate,'T')"/>

<xsl:variable name="timebit2" select="substring-after($timebit1,':')"/>

<xsl:variable name="hourbit" select="substring-before($timebit1,':')"/>

<xsl:variable name="minbit" select="substring-before($timebit2,':')"/>

<xsl:value-of select="concat($daybit,'-',$monthbit,'-',$yearbit,' ',$hourbit,':',$minbit)"/>

</xsl:template>

</xsl:stylesheet>

Rebecca French