IBM Content Manager OnDemand

Using the XML Indexer

2/12/2021

Brian Hoyt

Senior Software Engineer - Content Manager OnDemand

Table of Contents

Overview

There are two ways to use the Content Manager OnDemand XML indexer. Both methods are covered in this document.

The first method is to use the XML indexer to index and load your XML files into Content Manager OnDemand. These XML files might contain hundreds of individual documents (such as bank account statements) or thousands of individual transactions (such as SEPA financial transactions). You must indicate to the XML indexer how to identify the individual documents or transactions within the input file, and how to identify the index values you wish to associate with the documents or transactions. This is accomplished by transforming your XML input into an interim format, and then loading it into Content Manager OnDemand.

The second method is to use the Generic XML Index File Format (GXIFF) for the input index file, which provides the same basic functions as the Content Manager OnDemand Generic indexer, but it uses XML for the index file instead of a text file. This method provides an easy alternative to the Generic indexer for loading a variety of input files such as .jpg, .tif, and so on.

Loading XML input files

The XML indexer requires up to five file types for processing:

Input files to the transformation processor (such as Saxon or Xalan):

1) .**xml** file – your input XML file to be loaded into Content Manager OnDemand
2) .**xslt** file – the transformation file that you create, which is used to identify the individual documents or transactions and the index values to be loaded into Content Manager OnDemand

Optional resource files:

3) .**css,** .**jpg**… (or other file type) files – your resource files, such as cascading style sheet(s) and images, that you wish to load along with your XML. These files are used in the presentation of your XML, typically in a browser, and are packaged as a zip file with a file extension of .**in.res** file (described in #4)
4) .**in.res** file – a zip file that contains your resource files (described in #3) that you wish to load along with your XML

Input file to the ARSLOAD program or Add Report (ADDRPTOND) command:

5) .**in** – the output file from the transformation processor (such as Saxon or Xalan)

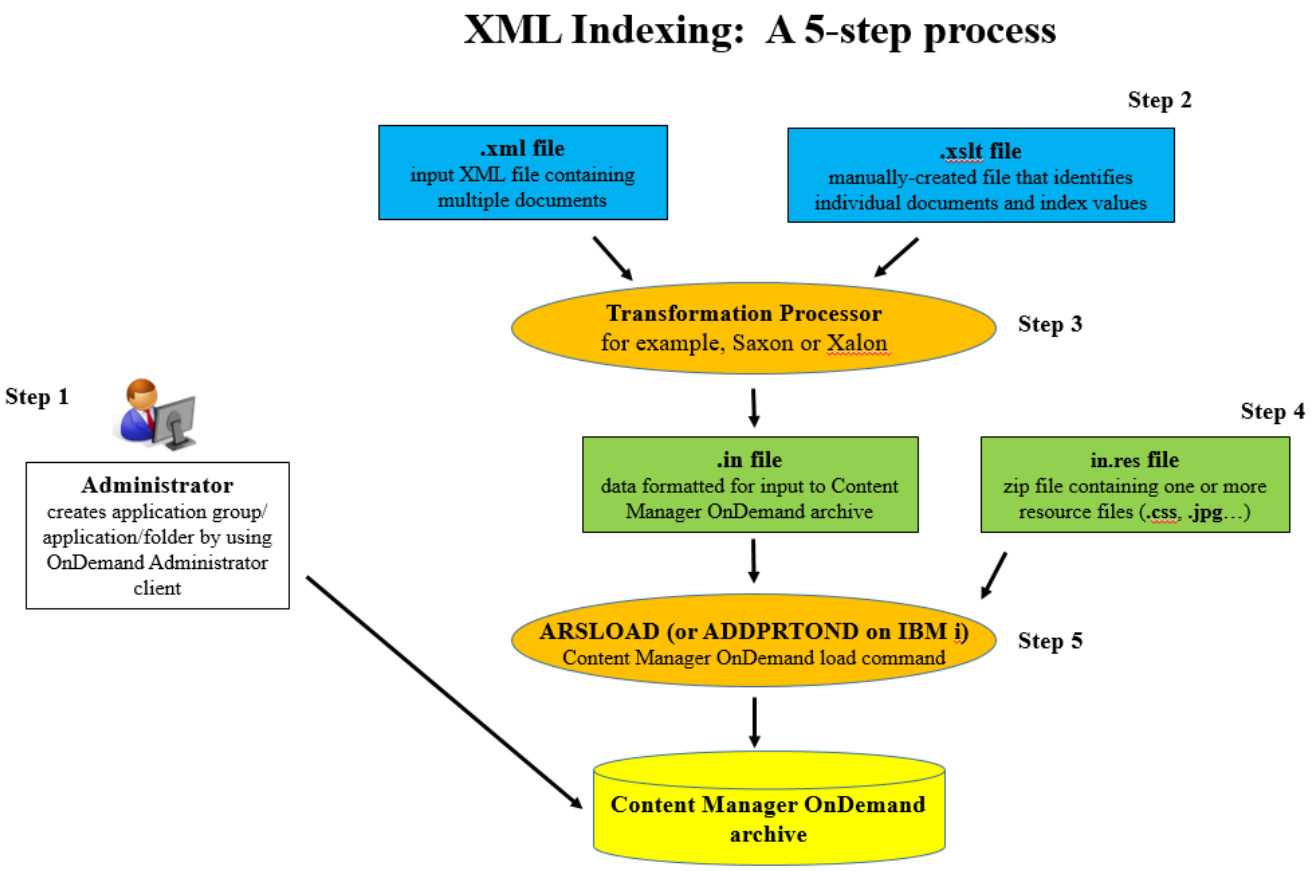
All five file types are further described below in the list of steps for XML indexer processing. Instructions for creating the required files are included in this document.

XML indexing involves up to five steps:

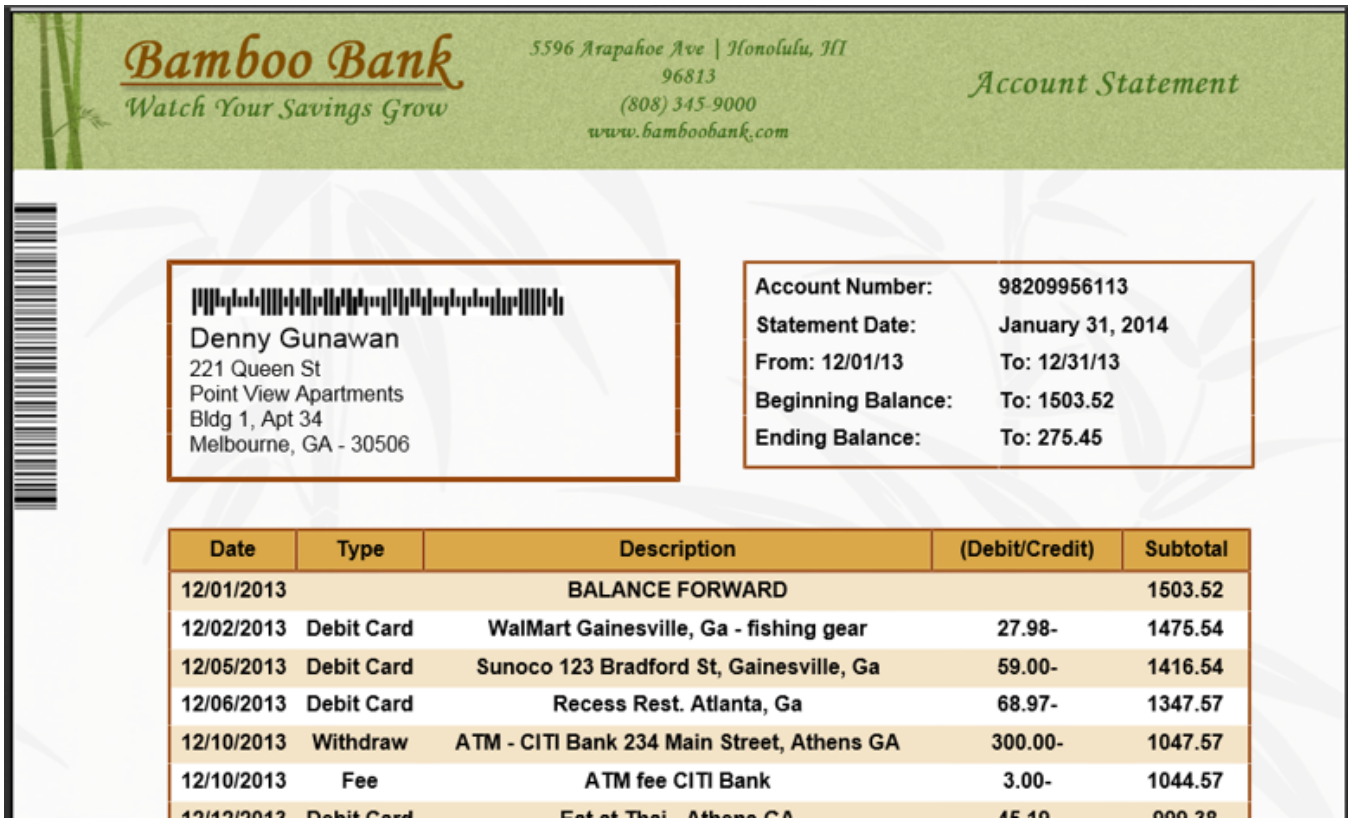
1) Create the Content Manager OnDemand application group, application, and folder.
2) Create a transformation file (file extension of .**xslt**) that is used in step 3 to identify the individual documents or transactions within your input XML file (file extension of .**xml**) and map the elements within your input XML file to the index fields defined in your Content Manager OnDemand application group.

3) Run the transformation (by using processors such as Saxon or Xalan), using your input XML file (file extension of .**xml**) and your transformation file (file extension of .**xslt**) as input. The transformation creates an output file (file extension of .**in**) that will be used as input in step 5.
4) Create a resource file (file extension of .**in.res**) that contains any resources you need to load along with your XML. These resources might include files such as cascading style sheets (file extension of .**css**) or images. These files are used in the presentation of your XML, typically in a browser, and are packaged as a zip file. This step is optional.
5) Run the ARSLOAD program or Add Report (ADDRPTOND) command (on IBM i) to load the XML and any related resources. The input to ARSLOAD or ADDRPTOND is the transformed file (file extension of .**in**) that was created in step 3 and the optional resource file (file extension of .**in.res**) that was created in step 4.

The relationships between the five file types and the five steps are shown below:



This document provides an example based on bank account statements from Bamboo Bank for the input XML file. The fully composed account statements look like this:



Bamboo Bank
Watch Your Savings Grow

5596 Arapahoe Ave | Honolulu, HI
96813
(808) 345-9000
www.bamboobank.com

Account Statement

Denny Gunawan
221 Queen St
Point View Apartments
Bldg 1, Apt 34
Melbourne, GA - 30506

Account Number:	98209956113
Statement Date:	January 31, 2014
From: 12/01/13	To: 12/31/13
Beginning Balance:	To: 1503.52
Ending Balance:	To: 275.45

Date	Type	Description	(Debit/Credit)	Subtotal
12/01/2013		BALANCE FORWARD		1503.52
12/02/2013	Debit Card	WalMart Gainesville, Ga - fishing gear	27.98-	1475.54
12/05/2013	Debit Card	Sunoco 123 Bradford St, Gainesville, Ga	59.00-	1416.54
12/06/2013	Debit Card	Recess Rest. Atlanta, Ga	68.97-	1347.57
12/10/2013	Withdraw	ATM - CITI Bank 234 Main Street, Athens GA	300.00-	1047.57
12/10/2013	Fee	ATM fee CITI Bank	3.00-	1044.57

Details of the steps required to define and load the Bamboo Bank account statements are as follows.

Step 1. Create the Content Manager OnDemand application group/application/folder

To use the XML indexer, you need to create your Content Manager OnDemand application definition with the following:

- Data Type = XML
- Indexer = XML

All other fields within the application group, application and folder definitions can be set as needed for your requirements.

Step 2. Create a transformation file

The next step is to create a transformation file (file extension of .**xslt**) that is used to identify the individual documents or transactions and the index fields and values that you want to use when you load your XML file into Content Manager OnDemand. The transformation can be done using tools such as XSLT and XQuery. In this example, we use XSLT. While there are many processor implementations of XSLT, Saxon and Xalan are two of the more popular open source versions.

When you run the transformation process, the original XML file is not changed, rather, a new XML file is created based on the content of the original file. The new XML file becomes the input to the ARSLOAD program or Add Report (ADDRPTOND) command (on IBM i).

Below is a listing of the input XML file to be loaded into Content Manager OnDemand with the XML indexer. In this example, this file name is Bbank.**xml**.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="bamboo.xsl"?>

<statements>
  <statement>
    <clientinfo>
      <acctnumber>98209956113</acctnumber>
      <sdate>January 31, 2014</sdate>
      <fromdate>12/01/13</fromdate>
      <todate>12/31/13</todate>
      <begbalance>1503.52</begbalance>
      <endbalance>275.45</endbalance>
      <client>Denny Gunawan</client>
      <addr1>221 Queen St</addr1>
      <addr2>Point View Apartments</addr2>
      <addr3>Bldg 1, Apt 34</addr3>
      <city>Melbourne</city>
      <state>GA</state>
      <zip>30506</zip>
    </clientinfo>
    <transaction>
      <tdate>12/02/2013</tdate>
      <ttype>Debit Card</ttype>
      <tdesc>WalMart Gainesville, GA – fishing gear</tdesc>
      <tdebit>27.98-</tdebit>
      <tsub>1475.54</tsub>
    </transaction>
    …
    …
  </statement>
  <statement>
    <clientinfo>
      …
    </clientinfo>
    <transaction>
      …
    </transaction>
    …
    …
  </statement>
  …
</statements>
```

Figure 1. ***Bamboo Bank Customer Statement XML (Bbank.xml)***

As you can see in *Figure 1*, the statement definition is straightforward. The **<clientinfo>** element contains customer and statement information. Following the **<clientinfo>** element are the customer transactions for the statement period. There can be zero or more of these with each one identifying the date, type, description, transaction amount, and subtotal.

In the example, the first customer statement is defined between the first pair of **<statement>** and **</statement>** tags**.** A second statement follows the first one, with its own pair of **<statement>** and **</statement>** tags. In fact, many statements could be included in this input XML file, all located within the one pair of **<statements>** and **</statements>** tags.

Below is a listing of an XSLT transformation file (file extension of .**xslt**) that was created in order to transform the input .**xml** file into the format required by the XML indexer. In this example, this file name is Bbank.**xslt**.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
<xsl:output method="xml" cdata-section-elements="COVER" omit-xml-declaration="no" indent="yes" />

<xsl:template match="node()|@*">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()" />
  </xsl:copy>
</xsl:template>

<xsl:template match="/statements">
  <odidx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xsl:apply-templates select="/statements/statement" />
</odidx>
</xsl:template>

<xsl:template match="/statements/statement">
<oddoc>
  <odxmldata>
    <xsl:copy-of select="."></xsl:copy-of>
  </odxmldata>
  <odindex>
    <xsl:attribute name="field">acctno</xsl:attribute>
    <xsl:attribute name="value">
      <xsl:value-of select="clientinfo/acctnumber" />
    </xsl:attribute>
  </odindex>
  <odindex>
    <xsl:attribute name="field">client</xsl:attribute>
    <xsl:attribute name="value">
      <xsl:value-of select="clientinfo/client" />
    </xsl:attribute>
  </odindex>
  <odindex>
    <xsl:attribute name="field">sdate</xsl:attribute>
    <xsl:attribute name="value">
      <xsl:value-of select="clientinfo/sdate" />
    </xsl:attribute>
  </odindex>
  <odindex>
    <xsl:attribute name="field">begbalance</xsl:attribute>
    <xsl:attribute name="value">
      <xsl:value-of select="clientinfo/begbalance" />
    </xsl:attribute>
  </odindex>
  <odindex>
    <xsl:attribute name="field">endbalance</xsl:attribute>
    <xsl:attribute name="value">
      <xsl:value-of select="clientinfo/endbalance" />
    </xsl:attribute>
  </odindex>
</oddoc>
</xsl:template>

</xsl:stylesheet>
```

Figure 2. *Bamboo Bank statement XSLT (Bbank.xslt)*

At a high level, the .**xslt** transformation file in *Figure 2* states that we are to do the following three things to the input XML file:

1. The first template (in red) states to match all nodes (attribute, root and other) in the source document. This will return all data in the document including processing instructions (stylesheets). The <xsl:copy> within this first template states to output the node. If another template is defined to handle the node, that template is called and the node will not be output by this directive. When you create your .**xslt** file, you should leave all of the lines in the first template unchanged. Include them in your .**xslt** file, and do not make any changes to these lines.
2. The second template (in blue) will handle all nodes matching **<statements>**. The "**/**" in front means to only look at top level nodes. If found, output the <odidx> element and then call another template for all **<statement>** nodes found underneath **<statements>**.
3. The third template (in green) is where the real work is performed. This template defines what to do for each **<statement>** node that is found. Each **<statement>** that is found will become its own Content Manager OnDemand document with metadata.

   For each **<statement>** that is found, an **<oddoc>** node will be created to define a new document. Next, all XML data within the **<statement>** (including the **<statement>** node itself) will be output inside the **<odxmldata>** node. After this, five **<odindex>** nodes will be created, which define the metadata to be associated with the document. In the example, the values of the account number, client name, statement date, beginning balance, and ending balance nodes are collected, and five index nodes are created that define the application group field (attribute called "field") to map to and the value to store (attribute called "value").

Step 3. Run the transformation process

To run the transformation process, which creates the .**in** input file for the ARSLOAD program or ADDRPTOND command, use the following command with the .**xml** and .**xslt** files as input:

```
java –classpath c:\saxon\saxon9he.jar net.sf.saxon.Transform
     –s:BBank.xml
     –xsl:BBank.xslt
     –o:BBank.xml.in
```

The resulting output file (Bbank.xml**.in**), shown in *Figure 3*, is now ready for loading into Content Manager OnDemand.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="bamboo.xsl"?>
<odidx xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance>
  <oddoc>
    <odxmldata>
      <statement>
        <clientinfo>
          <acctnumber>98209956113</acctnumber>
          <sdate>January 31, 2014</sdate>
          <fromdate>12/01/13</fromdate>
          <todate>12/31/13</todate>
          <begbalance>1503.52</begbalance>
          <endbalance>275.45</endbalance>
          <client>Denny Gunawan</client>
          …
        </clientinfo>
        <item>
          <tdate>12/02/2013</tdate>
          <ttype>Debit Card</ttype>
          <tdesc>WalMart Gainesville, Ga - fishing gear</tdesc>
          <tdebit>27.98-</tdebit>
          <tsub>1475.54</tsub>
        </item>
        …
      </statement>
    </odxmldata>
    <odindex field="account_number" value="98209956113"/>
    <odindex field="client_name" value="Denny Gunawan"/>
    <odindex field="statement_date" value="January 31, 2014"/>
    <odindex field="beg_balance" value="1503.52"/>
    <odindex field="end_balance" value="275.45"/>
  </oddoc>
  <oddoc>
     …
  </oddoc>
  …
</odidx>
```

Figure 3. *Bamboo Bank statement output from the transformation process; input to ARSLOAD or ADDRPTOND (Bbank.xml.in)*

As you can see in *Figure 3*, the content of each document is included between the **<odxmldata>** tags (in green) and the indexes for each document are defined in the **<odindex>** tags (in blue). Also note that the <xml-stylesheet> processing instruction in the input .**xml** file, which is used to define how to display the data, was also copied to the top of the .**in** document.

Step 4. Create a resource file (optional)

The resource file (file extension of .**in.res;** file name must match your **.in** file which was created during the transformation process in step 3) is a zip file that contains any resources you need to load along with your XML. These resources might include files such as cascading style sheets (.**css**), images, and any other files needed for presentation.

For example, to create the zip file (file extension of .**in.res**) that contains your resources, run a command similar to the following. Note that this is just an example of a zip command; it does not show a complete set of resources for the Bamboo bank statement as shown previously in this document.

```
zip Bbank.xml.in.res Bamboo.css Bbank1.jpg Bbank2.jpg
```

Remember that the name of the zip file (with the extension of .**in.res**) must match the name of your **.in** file which was created during the transformation process in step 3. For example, if the .**in** file name is Bbank.xml**.in**, the resource file name must be Bbank.xml**.in.res**.

Step 5. Run the ARSLOAD program or Add Report (ADDRPTOND) command

To run the XML indexer, run the ARSLOAD program or the Add Report (ADDRPTOND) command (on IBM i) with all the usual options and specify the .**in** file for the input. For example:

```
arsload -f -g Bbank-statements -I myinstance -n -u myuser -p arsload.stash -v
..\Bbank.xml.in
```

or (on IBM i):

```
ADDRPTOND APPGRP('Bbank-statements') INPUT(*STMF) STMF('..\Bbank.xml.in ')
INSTANCE(myinstance)
```

Advanced topics

Cascading style sheets – additional details

Cascading style sheets (CSS) are XML stylesheets which are used to describe the look and formatting of the document. Multiple CSS files can be defined per XML document. One such use of multiple stylesheets would be to define the formatting of the document for multiple end user devices, such as phone, tablet or desktop. Stylesheet files can reside locally or remotely. The XML indexer will only archive those stylesheets specified as local files. This means that if you want to view your documents via the web, the http reference in the stylesheet definition will point back to your web server where the stylesheet(s) reside.

For locally specified stylesheets, which are stylesheets prefixed with "file:", these stylesheets will be archived into a single zip file and stored as a resource for the Content Manager OnDemand application in which the XML documents are being stored.

Only a single level of stylesheets will be archived. If a specified stylesheet references images, for example, these images will not automatically be archived along with the stylesheet. You must manually add them to your resource zip file (file extension of **.in.res**). In other words, if you reference a CSS file in your XML file, Content Manager OnDemand will archive that CSS file for you, along with your XML. But if that CSS file has a reference to another file, Content Manager OnDemand does not parse the original CSS file to find the additional files.

Therefore, you must manually add any required files to the resource zip file (file extension of .**in.res**) for loading into Content Manager OnDemand.

XML indexer schema file

A schema file named odxmlidx.xsd, which ships with Content Manager OnDemand, defines the format of the file (file extension of .**in**) that is output from the transformation process and input to the ARSLOAD program/ADDRPTOND command. All input to the XML indexer has to follow this schema definition. Failure to comply with this format will result in parsing errors. If you need to review the schema file, it can be found in the following locations (where V10.5 or V10R5M0 is the server version of Content Manager OnDemand you are running):

AIX: `/opt/IBM/ondemand/V10.5/xml/odxmlidx.xsd`
Linux: `/opt/ibm/ondemand/V10.5/xml/odxmlidx.xsd`
Windows: `\Program Files\IBM\ondemand\V10.5\xml\odxmlidx.xsd`
z/OS: `/usr/lpp/ars/V10R5M0/bin/xml/odxmlidx.xsd`
IBM i: `/QIBM/ProdData/OnDemand/xml/odxmlidx.xsd`

Loading various files by using the Generic XML Index File Format (GXIFF)

The Generic XML Index File Format (GXIFF) provides a method for specifying the indexes in XML for any data type to be stored into Content Manager OnDemand. GXIFF provides the same basic functions as the Content Manager OnDemand Generic indexer, but uses XML for the index file instead of a text file that the Generic indexer uses.

In the example, we are loading two JPG files. The first example uses the Generic indexer with a .txt file to specify the indexing information. The second example uses the XML indexer with a GXIFF .xml file to specify the indexing information.

Generic indexer example

In the Content Manager OnDemand application definition, the Data Type is specified as JPG and the Indexer is specified as Generic.

```
COMMENT:
CODEPAGE:1208
COMMENT:
GROUP_FIELD_NAME:customer_name
GROUP_FIELD_VALUE:Brian Hoyt
GROUP_FIELD_NAME:account_number
GROUP_FIELD_VALUE:345987
GROUP_FIELD_NAME:statement_date
GROUP_FIELD_VALUE:2021-02-12
GROUP_OFFSET:0
GROUP_LENGTH:0
GROUP_FILENAME:image1.jpg
GROUP_FIELD_NAME:customer_name
GROUP_FIELD_VALUE:Darrell Bryant
GROUP_FIELD_NAME:account_number
GROUP_FIELD_VALUE:116590
GROUP_FIELD_NAME:statement_date
GROUP_FIELD_VALUE:2021-02-12
GROUP_OFFSET:0
GROUP_LENGTH:0
GROUP_FILENAME:image2.jpg
```

XML indexer example

In the Content Manager OnDemand application definition, the Data Type is specified as JPG and the Indexer is specified as XML.

```
<?xml version="1.0" encoding="UTF-8" ?>
<odidx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="odxmlidx.xsd">
<oddoc>
<oddataref file="image1.jpg" offset="0" length="0"/>
<odindex field="customer_name" value="Brian Hoyt"/>
<odindex field="account_number" value="345987"/>
<odindex field="statement_date" value="2021-02-12"/>
</oddoc>
<oddoc>
<oddataref file="image2.jpg" offset="0" length="0"/>
<odindex field="customer_name" value="Darrell Bryant"/>
<odindex field="account_number" value="116590"/>
<odindex field="statement_date" value="2021-02-12"/>
</oddoc>
</odidx>
```

GXIFF also supports loading multiple index rows for a single document. The file name, offset, and length must be the same, as shown in the following example. In the example, after the data is loaded, the image1.jpg file can be found by using either “345987” or “210654” as account number search criteria.

```
<?xml version="1.0" encoding="UTF-8" ?>
<odidx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="odxmlidx.xsd">
<oddoc>
<oddataref file="image1.jpg" offset="0" length="0"/>
<odindex field="customer_name" value="Brian Hoyt"/>
<odindex field="account_number " value="345987"/>
<odindex field="statement_date " value="2021-02-12"/>
</oddoc>
<oddoc>
<oddataref file="image1.jpg" offset="0" length="0"/>
<odindex field="customer_name" value="Brian Hoyt"/>
<odindex field="account_number" value="210654"/>
<odindex field="statement_date" value="2021-02-12"/>
</oddoc>
</odidx>
```

To run the XML indexer for this example, run the ARSLOAD program or the Add Report (ADDRPTOND) command (on IBM i) with all the usual options and specify the xml index file for the input. For example:

```
arsload -f -g Statements -I myinstance -n -u myuser -p arsload.stash -v
   ..\ImageIndexes.xml
```

or (on IBM i):

```
ADDRPTOND APPGRP('Statements') INPUT(*STMF) STMF('..\ImageIndexes.xml)
   INSTANCE(myinstance)
```