

Description: Guide to XML file-based data transfer for outputting user-selectable data.

OS Requirement: Win XP/Server 2003/ Server 2008/Vista/Windows 7

General Requirement: Knowledge of ICONICS BridgeWorX and BizViz suite of products, familiarity with XPath queries.

Introduction

This application note provides details for using the XML Parser Block in BridgeWorX. The XML Parser Block reads XML files and can insert their content into a database table, write to an OPC server, write to MS Excel reports, etc. Any application that involves the processing of XML files can benefit from this task.



You might use the XML Parser Block to bridge production data:

- If your process control applications output production information or PLC hardware sensor values to XML files, and you wish to integrate that data into batch databases, etc.
- To provide distributed application communication where firewalls and security are enforced.
- For remote facility management, monitoring and billing. Use BridgeWorX to upload XML files containing remote production data or even sensor data, and then merge all of the information into your centralized, corporate databases.

The XML Parser Block transfers XML data to OPC using a simple scalar-to-scalar mapping, or to a database using a matrix-to-matrix mapping. The XML Parser Block can parse simple and advanced XML documents. In addition:

- Each XML document must be well formed and include a root element.
- The XML file can be in XML format, or it can contain a data set.
- The parser’s result set is an instance of a .NET DataSet with columns and rows.

There are two operations you can use for parsing an XML file:

- ParseXML: You provide the static (that is, unchanging) path and filename along with the XPath query to pull data from the XML file.
- ParseXMLEx: You can parameterize the path and filename along with the XPath query. This lets you feed dynamic (that is, changing) information into the operation via parameters.

NOTE: XPath is a language that finds elements and attributes in an XML document. You will use it to configure any XML Parser Block. If you are unfamiliar with XPath, consult a book or website that explains XPath syntax.

The format of the XML document you use will determine the XPath query you will use to extract data. Figure 1 shows an example of the contents of a XML document from the food industry.

```
<Recipes>
  <DonutRecipe>
    <RecipeName>Commuter's special</RecipeName>
    <Flavor>Mint</Flavor>
    <Flour>20</Flour>
    <Sugar>10</Sugar>
    <BakeTime>10</BakeTime>
  </DonutRecipe>
  <DonutRecipe>
    <RecipeName>Three wonders</RecipeName>
    <Flavor>Lemon</Flavor>
    <Flour>10</Flour>
    <Sugar>19</Sugar>
    <BakeTime>80</BakeTime>
  </DonutRecipe>
</Recipes>
```

Figure 1 - Sample XML Document

Note the parent node XML (and also the root element) called **<Recipes>**, followed by the child node XML **<DonutRecipe>**. Submitting **/Recipes/DonutRecipe** as the XPath query would yield the following result:

Column1	Column2	Column3	Column4	Column5
Commuter's special	Mint	20	10	10
Three wonders	Lemon	10	19	80

Figure 2 - Resulting Rows and Columns

Tips! The source XML file does not have to be available at the time you configure a task that parses the file, but if a sample of the XML file is available you can test your configuration’s ability to parse it and see a result like the example shown above. Furthermore, if a sample file is available, the configuration process can also browse for the elements and attributes of the sample and load them to help you in your configuration.

Now, let’s look at a second example that uses following XML file content for a bookstore.

```
<?xml version="1.0" encoding="utf-8"?>
<bookstore>
  <book genre="autobiography" publicationdate="1981-03-22"
ISBN="1-861003-11-0">
    <title>The Autobiography of Benjamin Franklin</title>
    <author>
      <first-name>Benjamin</first-name>
      <last-name>Franklin</last-name>
    </author>
    <price>8.99</price>
  </book>
  <book genre="novel" publicationdate="1967-11-17" ISBN="0-
201-63361-2">
    <title>The Confidence Man</title>
    <author>
      <first-name>Herman</first-name>
      <last-name>Melville</last-name>
    </author>
    <price>11.99</price>
  </book>
  <book genre="science fiction" publicationdate="2008-01-
09" ISBN="1-605-06568-4">
    <title>Frankenstein</title>
    <author>
      <first-name>Mary</first-name>
      <last-name>Shelley</last-name>
    </author>
    <price>4.88</price>
  </book>
</bookstore>
```

Figure 3 - XML Content for a Bookstore

In this content, the parent node XML called **<bookstore>** is followed by the child node XML **<book>**, but notice that the **<book>** node contains the XML attributes of **genre**, **publicationdate**, and **ISBN**. When submitting an XPath query via the XML Parser Block, you can specify whether you want to retrieve the values of the *XML child nodes* or the *XML attributes*, using the following option, which is available to you in the XML Parser Block Configurator.

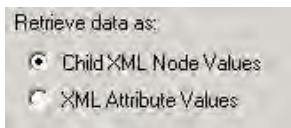


Figure 4 - Child Node Values v. Attribute Values

The XPath query **/bookstore/book** can yield two results – one for *Child Node Values* (Figure 5) and one for *XML attribute values* (Figure 6).

title	author	first-name	last-name	price
The Autobiograp...	BenjaminFranklin	Benjamin	Franklin	8.99
The Confidence ...	HermanMelville	Herman	Melville	11.99
Frankenstein	MaryShelley	Mary	Shelley	4.88

Figure 5 - Results showing XML Child Node Values

genre	publicationdate	ISBN
autobiography	1981-03-22	1-861003-11-0
novel	1967-11-17	0-201-63361-2
science fiction	2008-01-09	1-605-06568-4

Figure 6 - Results showing XML attribute values

Tip! By default, all available columns are retrieved from the XML file, but you can limit the number of columns.

The remainder of this application note will describe how to configure a transaction task that parses an XML document. How you do this depends on whether you are using:

- A static filename, path, and XPath query for the XML file.
- Parameters to identify a dynamic (that is, changing) filename, path, or XPath query for the XML file.

Inputting a Static Filename, Path, and XPath Query (ParseXML method)

To use as input a static filename, path, and XPath query, you will use the **ParseXML** operation in the following sequence of steps:

1. Launch the BridgeWorX Configurator application and add a new transaction or select an existing one.
2. Launch the Transaction Diagram Designer.
3. From the Transaction Diagram Designer, click the **Integration Tasks** button, then drag an XML File Parser task onto the work surface, as shown in Figure 7.

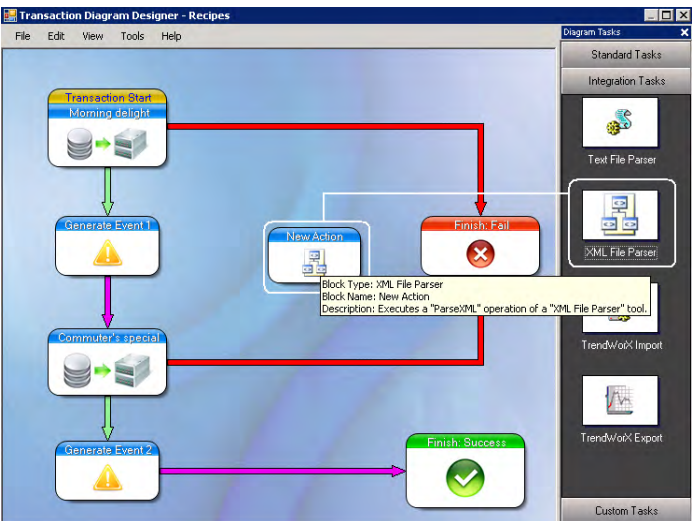


Figure 7 - Adding an XML File Parser Task

4. Double-click on the task to open the Transaction Task Properties dialog box. Complete the **General** tab as appropriate for the task you are adding.
5. Next, go to the **XML File Parser Properties** tab, shown in Figure 8. Leave **ParseXML** in the **Operation to Execute** field.

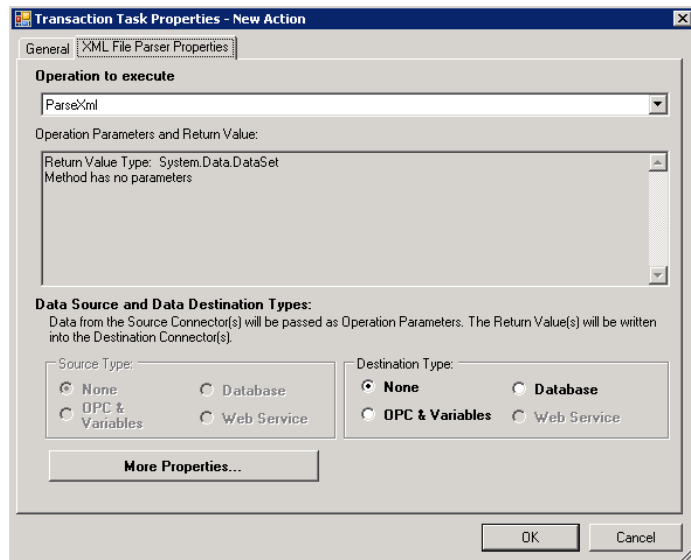


Figure 8 - ParseXML Operation

6. Choose the **Destination Type** – either **OPC & Variables** or **Database**.
7. Click the **More Properties** button. The XML Parser Configuration dialog box opens, shown in Figure 9.

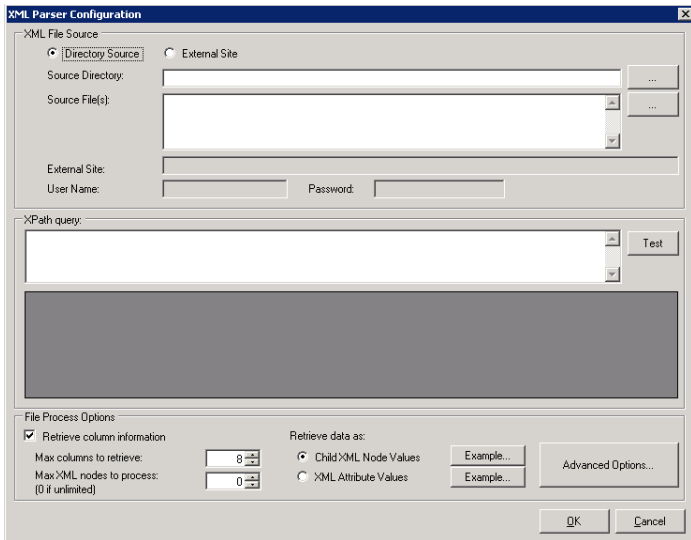


Figure 9 - XML Parser Configuration Dialog

8. In the **XML File Source** section, specify the location and filename of the XML source file that holds the data you want to import.
9. Next, skip ahead and select a **Retrieve data as** option. Each option has an **Example** button that you can click for an explanation of the data that will be retrieved.

Selecting **Child XML Node Values** retrieves values from *XML child nodes*, as illustrated in Figure 10.



Figure 10 - Values Retrieved from XML Child Nodes

Selecting **XML Attribute Values** retrieves values from *attributes*, as shown in Figure 11.

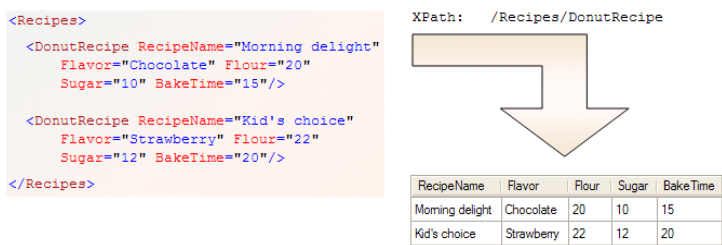


Figure 11 - Values retrieved from XML attributes

10. In the space provided, type the **XPath query** you want to use and click the **Test** button. The output of your selections displays *below* the **XPath query** box and will look something like the following:

Column1	Column2	Column3	Column4	Column5
Commuter's special	Mint	20	10	10
Three wonders	Lemon	10	19	80

Figure 12 - Sample Xpath query results

11. To refine the output, use the remaining options in the **File Process Options** box.
12. Click the **Advanced Options** button to open the XML File Processing Options dialog box, shown in Figure 13.



BridgeWorX –Using the XML Parser Block



APPLICATIONS NOTE

October 2013

13. Using the table at the top of the dialog box, define each column in the result set:
- Set the heading name for each column (note that if a sample XML file is in the path you provided, headings display; you can use them or change them).
 - (Optional) Enter a default value to be used if no value is provided for the column in the source XML file.
 - (Optional) Specify a value to be used if there's an error in processing the column.
 - (Recommended) Specify the column's data type (that is, if it is not the default of System String).

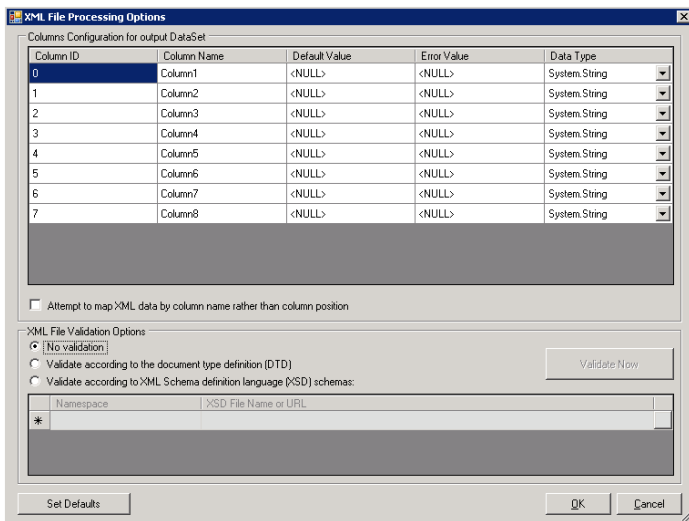


Figure 13 - XML File Processing Options Dialog

14. By default, BridgeWorX maps XML data to columns in the sequence in which they appear in the source XML file. This assumes that XML nodes or attributes are always presented in the same sequence in the source XML file. If there is a chance that their sequence will vary, select the **Attempt to map XML data by column name rather than column position** checkbox.
15. In validating XML documents, you can specify either of the following file types if applicable: document type definition (DTD) or XML schema definition (XSD).

Each of these file types defines the structure used in the XML documents that the Parser Block will be getting as input. These file types are used for integrating external applications. You decide which format to use for your XML documents.

NOTE: If you are unfamiliar with DTDs and XSDs, please consult one of the many books and websites that explain them, their uses, and their syntax.

Samples of each type of format are provided next.

Sample DTD File

In a DTD file format, the DTD header expresses the schema through a series of element and attribute declarations, and appears at the start of your XML documents. The following example shows the content of a sample XML file that includes a DTD header.

```
<!DOCTYPE store [  
  <!ELEMENT store (item)*>  
  <!ELEMENT item (name,dept,price)>  
  <!ATTLIST item type CDATA #REQUIRED>  
  <!ELEMENT name (#PCDATA)>  
  <!ELEMENT price (#PCDATA)>]>  
<store>  
  <item type="supplies" ISBN="2-3631-4">  
    <name>paint</name>  
    <price>16.95</price>  
  </item>  
</store>
```

Figure 14 - Sample DTD File

Sample XSD File

In an XSD file format, a separate .xsd file is used for defining the format of XML files. The following example shows the content of a sample .xsd file being used to validate the bookstore.xml file.

```
<?xml version="1.0" encoding="utf-8"?>  
<xs:schema attributeFormDefault="unqualified" xmlns="urn:bookstore-schema"  
  elementFormDefault="qualified" targetNamespace="urn:bookstore-schema"  
  xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <xs:element name="bookstore">  
    <xs:complexType>  
      <xs:sequence>  
        <xs:element maxOccurs="unbounded" name="book">  
          <xs:complexType>  
            <xs:sequence>  
              <xs:element name="title" type="xs:string" />  
              <xs:element name="author">  
                <xs:complexType>  
                  <xs:sequence>  
                    <xs:element minOccurs="0"  
                      name="name" type="xs:string" />  
                    <xs:element minOccurs="0"  
                      name="first-name" type="xs:string" />  
                    <xs:element minOccurs="0"  
                      name="last-name" type="xs:string" />  
                  </xs:sequence>  
                </xs:complexType>  
              <xs:element name="price" type="xs:decimal" />  
              <xs:sequence>  
                <xs:attribute name="genre" type="xs:string"  
                  use="required" />  
                <xs:attribute name="publicationdate" type="xs:int"  
                  use="required" />  
                <xs:attribute name="ISBN" type="xs:string"  
                  use="required" />  
              </xs:sequence>  
            </xs:complexType>  
          </xs:element>  
        </xs:sequence>  
      </xs:complexType>  
    </xs:element>  
  </xs:schema>
```

Figure 15 - Sample XSD File

You must select which type of file format is to be validated:

- For a DTD validation, select the **Validate according to the document type definition (DTD)** option. It is assumed that the DTD header will be included in each XML document. If you have provided a sample XML

document that has a DTD header, click the **Validate Now** button.

- For an XSD validation, select the **Validate according to XML Schema definition language (XSD)** option. In the row(s) beneath the XSD file option, you must also specify the namespace and the filename (or the URL) of the .xsd file, as shown in the example Figure 16 below, then click the **Validate Now** button.

NOTE: The namespace is provided in the xmlns attribute in the .xsd file. In the sample XSD file shown on the prior page, on the second line you can see xmlns="urn:bookstore-schema", therefore urn:bookstore-schema is entered as the namespace.

XML File Validation Options

☐ No validation
☒ Validate according to the document type definition (DTD)
☒ Validate according to XML Schema definition language (XSD) schemas:

Namespace	XSD File Name or URL
urn:bookstore-schema	bookstore.xsd

Figure 16 - Namespace and Filename for an .xsd File

- When you are done, click **OK** to close the XML File Processing Options dialog box.
- Click **OK** to close the XML Parser Configuration dialog box. This returns you to the Transaction Task Properties dialog box.
- Click the **Next** button to format the destination. The **Return Values Mapping** tab appears. Depending on the Destination Type selected on the XML File Parser Properties tab, it looks like the tab shown in either Figure 18 or Figure 19.
- Using the **Return Values Mapping** tab, bind to the data destinations that you want – such as global variables. Click a column's ellipsis [...] button to set up the column. For example, for each **Data Source**, choose an XML column; for **Formula**, specify an expression for conversion (if any); and for each row's **Bind To**, choose an OPC tag or a variable to bind the value to or a database column. The example in Figure 17 shows the **Data Destination Browser** being used for selecting OPC tags for binding the values.

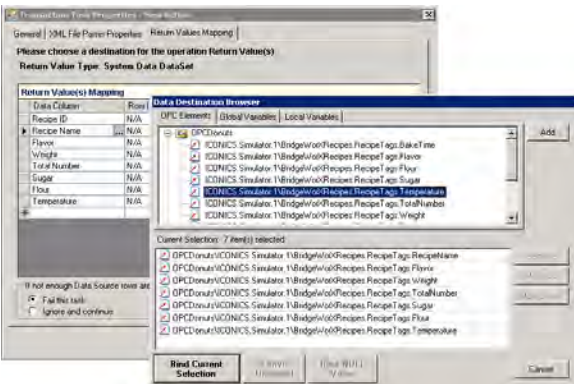


Figure 17 - Example of Binding to OPC Tags

Figure 18 and Figure 19 show the **Return Values Mapping** tab with values bound to OPC tags and database columns.

- Select any additional options you want to use.

NOTE: If return values are going to a database, the **Enable transaction scope for transfers** option refers to the **database** transaction scope for multiple inserts.

- Click the **OK** button to save the options and close the dialog box.

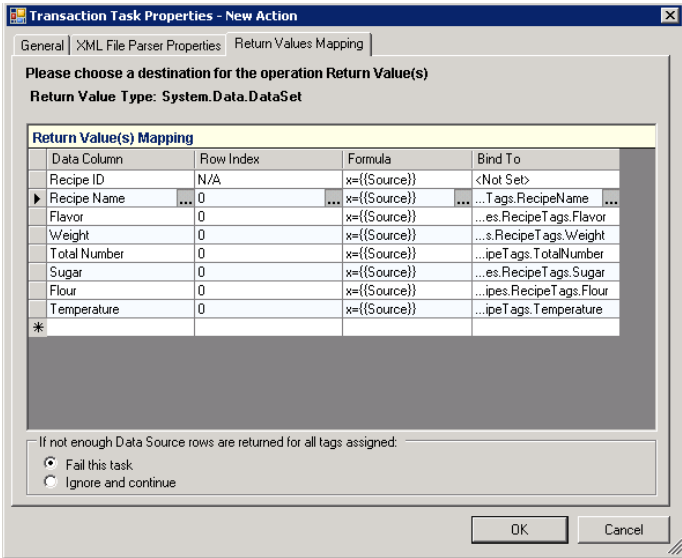


Figure 18 - Mapping Values for OPC & Variables

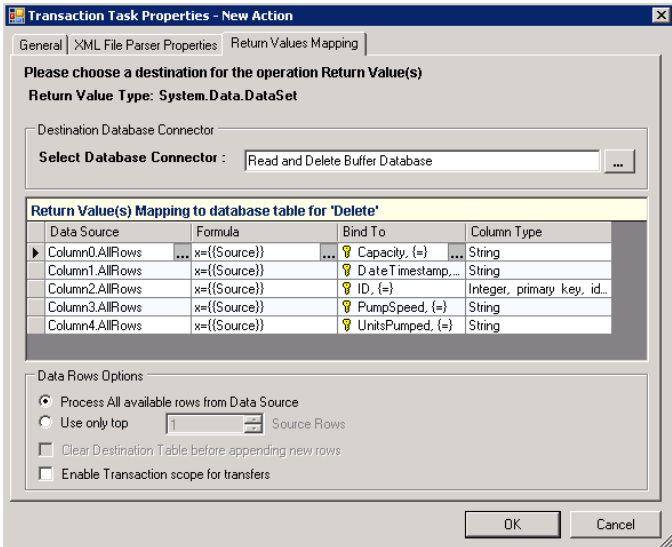


Figure 19 - Mapping Values to the Database

Inputting a Dynamic File, Path, and XPath Query (ParseXMLeX method)

To use as input a dynamic filename, path, or XPath query, you will use the **ParseXMLeX** operation. Many of the steps are the same as those used for the ParseXML operation. This section highlights how the steps differ.

1. Once you have added an XML File Parser task box onto the work surface inside the Transaction Diagram Designer, you should again double-click on it to open the Task Properties.
2. Complete the **General** tab as appropriate for the task that you are adding.
3. Go to the **XML File Parser Properties** tab shown below in Figure 20. Choose **ParseXMLeX** as the **Operation to Execute**.

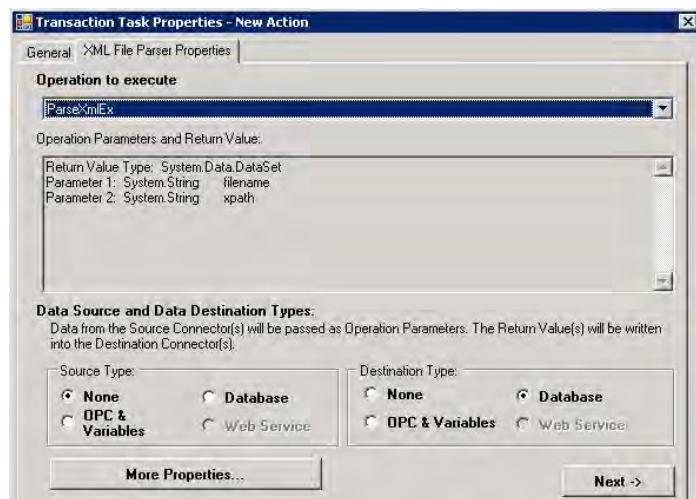


Figure 20 - ParseXMLeX Operation for Providing Input

4. Click the **More Properties** button and complete the dialog boxes to design the output. This information is provided in **Steps 7 through 17** on pages 3 to 5 of this application note.
5. Choose the **Source Type** – either **OPC & Variables** or **Database**. The Source parameters will be used for mapping to the filename and Xpath query parameters for the task. For example you can have the filename coming from a String OPC tag, or you can have the Xpath query coming from an XML blob field in a database somewhere.
6. Choose the **Destination Type** – either **OPC & Variables** or **Database**.
7. Click the **Next** button. Two additional tabs appear: **Parameters Mapping** (see Figure 21 and Figure 22) and **Return Values Mapping**.
8. On the **Parameters Mapping** tab, use the **Operation Parameters Mapping** section to map to an appropriate string for the filename and an appropriate string for the XPath query. If you are providing database parameters, use

the **Database Command Parameter Mapping** section to specify any parameters for database commands.

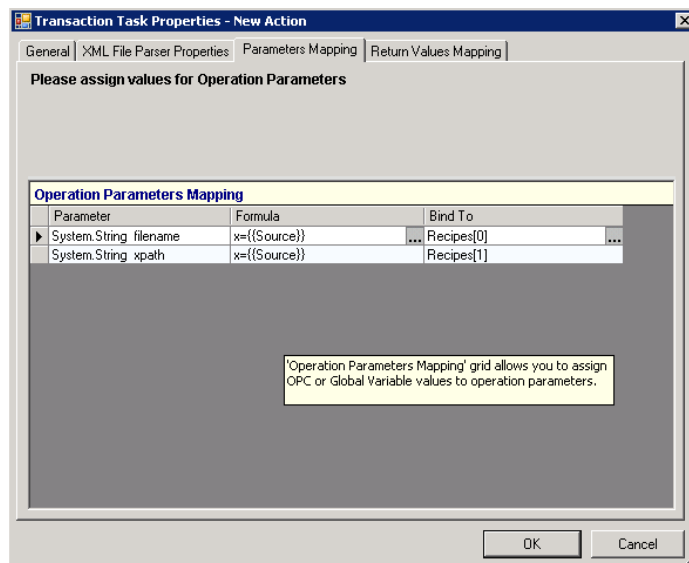


Figure 21 - Parameters for OPC & Variables

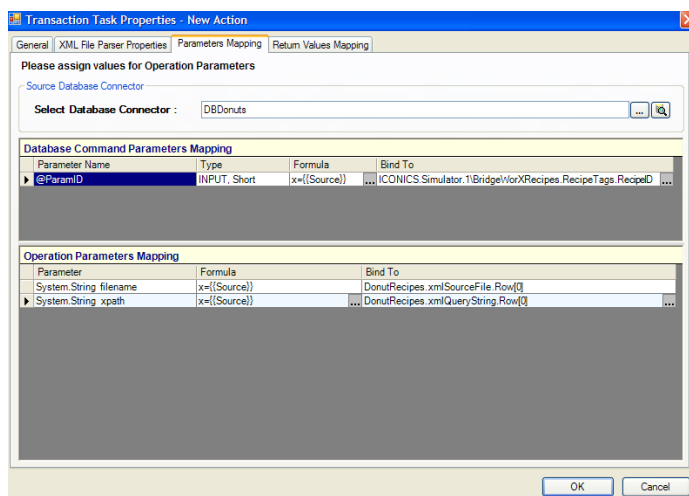


Figure 22 - Parameters for the Database

9. On the **Return Values Mapping** tab, enter mapping values and select the options you want to use. For more information, refer to steps 19 through 21 on page 5 of this application note.
10. Click the **OK** button to save the options and close the dialog box.

You are now ready to run your Transaction with the XML File Parser Task!