IBM VisualAge® for Java™, Version 3.5

XML Parser for Java

Note!
Before using this information and the product it supports, be sure to read the general information under **Notices**.

Edition notice

This edition applies to Version 3.5 of IBM VisualAge for Java and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. XML Parser for Java: Overview

The XML Parser for Java provides a way for your applications to work with XML data on the Web. The XML Parser provides classes for parsing, generating, manipulating, and validating XML documents. You can include the XML Parser in Business-to-Business (B2B) and other applications that manage XML documents, work with metacontent, interface with databases, and exchange messages and data.

The XML Parser is written entirely in Java, and conforms to the XML 1.0 Recommendation and associated standards, such as Document Object Model (DOM) 1.0, Simple API for XML (SAX) 1.0, and the XML Namespaces Recommendation.

DOM implementations
The Document Object Model is an application programmer's interface to XML data. XML parsers produce a DOM representation of the parsed XML. Your application uses the methods defined by the DOM to access and manipulate the parsed XML.

The IBM XML Parser provides two DOM implementations:

- Standard DOM: provides the standard DOM Level 1 API, and is highly tuned for performance
- TX Compatibility DOM: provides a large number of features not provided by the standard DOM API, and is not tuned for performance.

You choose the DOM implementation you need for your application when you write your code. You cannot, however, use both DOM's in the XML Parser at the same time. In the XML Parser, the DOM API is implemented using the SAX API.

Modular design
The XML Parser has a modular architecture. This means that you can customize the XML Parser in a variety of different ways, including the following:

- Construct different types of parsers using the classes provided, including:
  - Validating and non-validating SAX parser
  - Validating and non-validating DOM parser
  - Validating and non-validating TXDOM parser

  To see all the classes for the XML Parser, look in the VisualAge for Java IDE for the IBM XML Parser for Java project and the com.ibm.xml.parsers package.
- Specify two catalog file formats: the SGML Open catalog, and the XCatalog format.
- Replace the DTD-based validator with a validator based on some other method, such as the Document Content Description (DCD), Schema for Object-Oriented XML (SOX), or Document Definition Markup Language (DDML) proposals under consideration by the World Wide Web Consortium (W3C).

Constructing a parser with only the features your application needs reduces the number of class files or the size of the JAR file you need. For more information about constructing the XML Parser, refer to the related tasks at the bottom of this page.

Constructing a parser
You construct a parser by instantiating one of the classes in the com.ibm.xml.parsers package. You can instantiate the classes in one of the following ways:

- Using a parser factory
- Explicitly instantiating a parser class
- Extending a parser class

For more information about constructing a parser, refer to the related tasks at the bottom of this page.

Samples
We provide the following sample programs in the IBM XML Parser for Java Examples project. The sample programs demonstrate the features of the XML Parser using the SAX and DOM APIs:

- SAXWriter and DOMWriter: parse a file, and print out the file in XML format.
- SAXCount and DOMCount: parse your input file, and output the total parse time along with counts of elements, attributes, text characters, and white space characters you can ignore. SAXCount and DOMCount also display any errors or warnings that occurred during the parse.
- DOMFilter: searches for specific elements in your XML document.
- TreeViewer: displays the input XML file in a graphical tree-style interface. It also highlights lines that have validation errors or are not well-formed.

To see the sample programs, you need to add the IBM XML Parser for Java Examples feature to the VisualAge for Java workspace. Refer to the Samples section for details.

RELATED CONCEPTS

Limitations

RELATED TASKS

Using a parser factory
Explicitly instantiating a parser class
Extending a parser class
Creating a DOM parser
Creating a SAX parser
Using catalog files
Using namespaces
Using the revalidation API
Handling errors

RELATED REFERENCES

Package-com.ibm.xml.dom
Package-com.ibm.xml.framework
Package-com.ibm.xml.parser
Package-com.ibm.xml.parser.util
Package-com.ibm.xml.parsers
Package-com.ibm.xml.xpointer
Package-org.w3c.dom
Package-org.xml.sax.helpers
Package-org.xml.sax

Limitations

The XML Parser has the following limitations:

- If there is an error in the encoding line, the XML Parser may report the location of the error as location -1,-1.
- When parsing unparsed entities that refer to notations declared after the entity reference, the XML Parser will report the error at the end of the DTD, not at the point where the unparsed entity was declared.
- This release of the XML Parser does not yet provide the readDTDStream() method in the TX Compatibility parser.

RELATED CONCEPTS

Overview

RELATED TASKS

Using a parser factory
Explicitly instantiating a parser class
Extending a parser class
Creating a DOM parser
Creating a SAX parser
Using catalog files
Using namespaces
Using the revalidation API
Handling errors

Chapter 2. Working with the XML Parser

Constructing a parser

Creating a DOM parser

You can construct a parser in your application in one of the following ways:

- Using a parser factory
- Explicitly instantiating a parser class
- Extending a parser class

To create a DOM parser, use one of the methods listed above, and specify com.ibm.xml.parsers.DOMParser to get a validating parser, or com.ibm.xml.parsers.NonValidatingDOMParser to get a non-validating parser.

To access the DOM tree, your application can call the getDocument() method on the parser.

For more information about constructing a parser, refer to the related tasks below.

RELATED CONCEPTS

Overview
Limitations

RELATED TASKS

Using a parser factory
Explicitly instantiating a parser class
Extending a parser class
Creating a SAX parser
Using catalog files
Using namespaces
Using the revalidation API
Handling errors

Creating a SAX parser

You can construct a parser in your application in one of the following ways:

- Using a parser factory
- Explicitly instantiating a parser class
- Extending a parser class

To create a SAX parser, use one of the methods listed above, and specify com.ibm.xml.parsers.ValidatingSAXParser to get a validating parser, or com.ibm.xml.parsers.SAXParser to get a non-validating parser.

Once your application creates the parser instance, it can use the standard SAX methods to set the various handlers provided by SAX.

For more information about constructing a parser, refer to the related tasks below.

RELATED CONCEPTS

Overview
Limitations

RELATED TASKS

Using a parser factory
Explicitly instantiating a parser class
Extending a parser class
Creating a DOM parser
Using catalog files
Using namespaces
Using the revalidation API
Handling errors

Using a parser factory

To construct a parser using a parser factory, call the makeParser () method of the org.xml.sax.helpers.ParserFactory class, passing a string that has the fully qualified name of the parser class that you are trying to instantiate. This method is useful if your application will need to switch between different parser configurations.

The following code example uses this method to construct a DOMParser:

```
import org.xml.sax.Parser;
import org.xml.sax.helpers.ParserFactory;
import com.ibm.xml.parsers.DOMParser;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;
import dom.DOMWriter;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
//Constructing parser using parser factory
public class example1 {
    static public void main( String[] argv ) {
       String parserClass = "com.ibm.xml.parsers.DOMParser";
       String xmlFile = "file:///xml_document_to_parse";
       Parser parser = null;
       try {
             parser = ParserFactory.makeParser( parserClass );
             parser.parse(xmlFile);
       } catch (SAXException se) {
             se.printStackTrace();
       } catch (IOException ioe) {
             ioe.printStackTrace();
       } catch (ClassNotFoundException ex ){
             ex.printStackTrace();
       } catch (IllegalAccessException ex ) {
             ex.printStackTrace();
       } catch (InstantiationException ex ) {
             ex.printStackTrace();
       } catch (ClassCastException ex ) {
             ex.printStackTrace();
       }
// The next lines are only for DOM Parsers
     Document doc = ((DOMParser) parser).getDocument();
     if ( doc != null ) {
         try {
```

```
                (new dom.DOMWriter( false ) ).print( doc ); // use print
                                                            // method from
dom.DOMWriter
            } catch ( UnsupportedEncodingException ex ) {
                ex.printStackTrace();
            }
        }
    }
}
```

Once your application has the XML document object, it can call any method on the document object that is defined by the DOM specification.

Running the code example in the Workbench
To run the code example in the Workbench, you need to do the following:

- Replace *xml_document_to_parse* with the name of the XML document you want to parse. To use the sample XML document we supply, specify the following:

  X:\IBMVJava\ide\project_resources\IBM XML Parser for Java Examples\data\personal.xml

  where *X:* is the drive where VisualAge for Java is installed.
- After you create the example1 class in the Workbench, specify the XML Parser in the class path for the example1 class. To specify the class path, do the following:
  1. In the Workbench, right-click the **example1** class, and in the pop-up menu click **Properties**.
  2. In the Properties dialog box, click the **Class Path** tab.
  3. Select the **Project path** check box, and click the **Compute Now** button. **IBM XML Parser for Java** appears in the Project path text field.

RELATED CONCEPTS

Overview
Limitations

RELATED TASKS

Explicitly instantiating a parser class
Extending a parser class
Creating a DOM parser
Creating a SAX parser
Using catalog files
Using namespaces
Using the revalidation API
Handling errors

Explicitly instantiating a parser class

Another way to instantiate a parser is to explicitly instantiate the parser class by making a new object instance of it. This method is useful when you know exactly which parser configuration you need, and you are sure that you will not need to switch configurations.

The following code example uses this method to construct a DOMParser:

```
import com.ibm.xml.parsers.DOMParser;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;
```

```
import java.io.IOException;
import java.io.UnsupportedEncodingException;
//Constructing parser by instantiating parser object
//In this case from DOMParser
public class example2 {
   static public void main( String[] argv ) {
      String xmlFile = "file:///xml_document_to_parse";
      DOMParser parser = new DOMParser();
      try {
           parser.parse(xmlFile);
      } catch (SAXException se) {
           se.printStackTrace();
      } catch (IOException ioe) {
           ioe.printStackTrace();
      }
// The next lines are only for DOM Parsers
      Document doc = ((DOMParser) parser).getDocument();
      if ( doc != null ) {
          try {
               (new dom.DOMWriter( false ) ).print( doc ); // use print
method from dom.DOMWriter
         } catch ( UnsupportedEncodingException ex ) {
                   ex.printStackTrace();
         }
      }
  }
}
```

Once your application has the XML document object, it can call any method on the document object that is defined by the DOM specification.

Running the code example in the Workbench
To run the code example in the Workbench, you need to do the following:

- Replace *xml_document_to_parse* with the name of the XML document you want to parse. To use the sample XML document we supply, specify the following:

  X:\IBMVJava\ide\project_resources\IBM XML Parser for Java Examples\data\personal.xml

  where *X:* is the drive where VisualAge for Java is installed.
- After you create the example1 class in the Workbench, specify the XML Parser in the class path for the example1 class. To specify the class path, do the following:
  1. In the Workbench, right-click the **example1** class, and in the pop-up menu click **Properties**.
  2. In the Properties dialog box, click the **Class Path** tab.
  3. Select the **Project path** check box, and click the **Compute Now** button. **IBM XML Parser for Java** appears in the Project path text field.

RELATED CONCEPTS

Overview
Limitations

RELATED TASKS

Using a parser factory
Extending a parser class

Creating a DOM parser
Creating a SAX parser
Using catalog files
Using namespaces
Using the revalidation API
Handling errors

Extending a parser class

You can also construct a parser by extending a parser class we supply. Constructing a parser by extending or subclassing an existing parser class, such as the DOMParser class, enables you to use inheritance to extend the parser or to override its default behaviour.

The following code example uses this method to construct a DOMParser:

```
import com.ibm.xml.parsers.DOMParser;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
//Constructing parser by instantiating parser object
//In this case from DOMParser
public class example3 extends DOMParser {
   static public void main( String[] argv ) {
      String xmlFile = "file:///xml_document_to_parse";
      example3 parser = new example3();
      try {
           parser.parse(xmlFile);
      } catch (SAXException se) {
           se.printStackTrace();
      } catch (IOException ioe) {
           ioe.printStackTrace();
      }
// The next lines are only for DOM Parsers
      Document doc = ((DOMParser) parser).getDocument();
      if ( doc != null ) {
         try {
             (new dom.DOMWriter( false ) ).print( doc ); // use print
method from dom.DOMWriter
         } catch ( UnsupportedEncodingException ex ) {
                  ex.printStackTrace();
         }
      }
   }
}
```

Once your application has the XML document object, it can call any method on the document object that is defined by the DOM specification.

Running the code example in the Workbench
To run the code example in the Workbench, you need to do the following:

- Replace *xml_document_to_parse* with the name of the XML document you want to parse. To use the sample XML document we supply, specify the following:

  X:\IBMVJava\ide\project_resources\IBM XML Parser for Java Examples\data\personal.xml

where *X:* is the drive where VisualAge for Java is installed.

- After you create the example1 class in the Workbench, specify the XML Parser in the class path for the example1 class. To specify the class path, do the following:
  1. In the Workbench, right-click the **example1** class, and in the pop-up menu click **Properties**.
  2. In the Properties dialog box, click the **Class Path** tab.
  3. Select the **Project path** check box, and click the **Compute Now** button. **IBM XML Parser for Java** appears in the Project path text field.

RELATED CONCEPTS

Overview
Limitations

RELATED TASKS

Using a parser factory
Explicitly instantiating a parser class
Creating a DOM parser
Creating a SAX parser
Using catalog files
Using namespaces
Using the revalidation API
Handling errors

Using catalog files

The XML Parser supports the following catalog file formats:

- SGML Open catalog
- XCatalog

The following sections explain how to use these catalogs formats.

Using the SGML Open catalog format
To use the SGML Open catalog file format, set a TXCatalog instance as the XML Parser's EntityResolver. For example:

```
XMLParser parser = new DOMParser();
Catalog catalog = new TXCatalog(parser.getParserState());
parser.getEntityHandler().setEntityResolver(catalog);
```

Once the catalog is installed, catalog files that conform to the TXCatalog format can be appended to the catalog by calling the loadCatalog () method on the parser or the catalog instance. The following example loads the contents of two catalog files:

```
parser.loadCatalog(new InputSource("catalogs/cat1.xml"));
parser.loadCatalog(new
InputSource("http://host/catalogs/cat2.xml"));
```

Using the XCatalog format
The current version of the XCatalog catalog supports the XCatalog proposal draft 0.2. XCatalog is an XML representation of the SGML Open Technical Resolution TR9401:1997 catalog format. The current proposal supports public identifier maps, system identifier aliases, and public identifier prefix delegates. To see the full

specification of this catalog format, go to the SCatalog DTD at http://www.ccil.org/~cowan/XML/XCatalog.html.

To use the XCatalog catalog, you must first have a catalog file in the XCatalog format. When you write XCatalog catalog files, your file must conform to these requirements:

- Use the XCatalog grammar.
- Specify the <!DOCTYPE> line with the PUBLIC specified as "-//DTD XCatalog//EN" or make sure that the system identifier is able to locate the XCatalog 0.2 DTD. XCatalog 0.2 DTD is included in the JAR file containing the com.ibm.xml.internal.XCatalog class. For example:

  ```
  <!DOCTYPE
  XCatalog
  PUBLIC "-//DTD XCatalog//EN"
  "com/ibm/xml/internal/xcatalog.dtd">
  ```

- The enclosing document root element is not optional. It must be specified.
- The Version attribute of the has been modified from '#FIXED "1.0"' to '(0.1|0.2) "0.2"'.

To use the XCatalog catalog in the XML Parser, set an XCatalog instance as the XML Parser's EntityResolver. For example:

```
XMLParser parser = new SAXParser();
Catalog catalog = new XCatalog(parser.getParserState());
parser.getEntityHandler().setEntityResolver(catalog);
```

Once they are installed, catalog files that conform to the XCatalog grammar can be appended to the catalog by calling the loadCatalog () method on the parser or the catalog instance. The following example loads the contents of two catalog files:

```
parser.loadCatalog(new InputSource("catalogs/cat1.xml"));
parser.loadCatalog(new InputSource("http://host/catalogs/cat2.xml"));
```

Limitations

The following are the current limitations of this XCatalog implementation:

- No error checking is done to avoid circular Delegate or Extend references.
- You cannot specify a combination of catalog files that reference each other.

RELATED CONCEPTS

Overview
Limitations

RELATED TASKS

Using a parser factory
Explicitly instantiating a parser class
Extending a parser class
Creating a DOM parser
Creating a SAX parser
Using namespaces
Using the revalidation API
Handling errors

Using namespaces

The easiest way to get namespace support is to use the TX compatibility classes that provide an API for dealing with namespace information. There are no standard API's for namespace manipulation in the standard DOM and SAX packages. The TX Compatibility classes provide additional, non-standard API's to work with namespaces.

The XML Namespace Recommendation does not currently specify the behavior of validation in the presence of namespaces. The behavior of all validating parsers, not just the IBM XML Parser, when namespaces are in use, is currently undefined. Additionally, when using the standard DOM API, element names containing colons (:) are treated as normal element names.

If you want to use namespace-like element names (for example, `a:foo`) with validation, create a new DTD that contains fully-qualified names from all the DTD's in use. Since the colon character is treated as a normal element name character, this merged DTD will allow you to do validation using these namespace-like names.

RELATED CONCEPTS

Overview
Limitations

RELATED TASKS

Using a parser factory
Explicitly instantiating a parser class
Extending a parser class
Creating a DOM parser
Creating a SAX parser
Using catalog files
Using the revalidation API
Handling errors

Using the revalidation API

You can validate a document after it has been parsed and converted to a DOM tree by using the RevalidatingDOMParser or the TXRevalidatingDOMParser classes. The validate () method in these classes takes a DOM node as an argument, and performs a validity check on the DOM tree rooted at that node, using the DTD of the current document. Currently, the native DOM prevents the insertion of invalid nodes, so this feature is not as useful for the native DOM.

The sample program below parses a document, inserts an illegal node into the TX DOM, and then tries to re-validate the document:

```
import java.io.IOException;
import org.xml.sax.SAXException;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import com.ibm.xml.parsers.TXRevalidatingDOMParser;
import com.ibm.xml.parser.TXElement;
public class RevalidateSample {
public static void main(String args[]) {
```

```
     String xmlFile = "file:///d:/xml4j_2_0_15/data/personal.xml";
     TXRevalidatingDOMParser parser = new TXRevalidatingDOMParser();
      try {
           parser.parse(xmlFile);
      }
      catch (SAXException se) {
      System.out.println("SAX error: caught "+se.getMessage());
      se.printStackTrace();
      }
      catch (IOException ioe) {
          System.out.println("I/O Error: caught "+ioe);
          ioe.printStackTrace();
      }
      Document doc = parser.getDocument();
      System.out.println("Doing initial validation");
      Node pos = parser.validate(doc.getDocumentElement());
      if (pos == null) { System.out.println("ok."); }
     else {
         System.out.println("Invalid at " + pos);
         System.out.println(pos.getNodeName());
     }
    // Now insert dirty data
    Node junk = new TXElement("bar");
    Node corrupt = doc.getDocumentElement();
    System.out.println("Corrupting: "+corrupt.getNodeName());
    corrupt.insertBefore(junk,corrupt.getFirstChild().getNextSibling());
    System.out.println("Doing post-corruption validation");
    Node position = parser.validate(doc.getDocumentElement());
    if (position == null) {
         System.out.println("ok.");
    }
     else {
          System.out.println("Invalid at " + position);
          System.out.println(position.getNodeName());
    }
}
}
```

The sample program should return the following result:

```
Doing initial validation
ok.
Corrupting: personnel
Doing post-corruption validation
Invalid at com.ibm.xml.parser.TXElement@f33ada64
bar
```

RELATED CONCEPTS

Overview
Limitations

RELATED TASKS

Using a parser factory
Explicitly instantiating a parser class
Extending a parser class

Handling errors

When you create an XML Parser instance, the default error handler does nothing. This means that your program will fail silently when it encounters an error.

To handle errors, you should register an error handler with the XML Parser by supplying a class that implements the org.xml.sax.ErrorHandler interface. This is true regardless of whether your XML Parser is DOM-based or SAX-based.

RELATED CONCEPTS

Overview
Limitations

RELATED TASKS

Using a parser factory
Explicitly instantiating a parser class
Extending a parser class
Creating a DOM parser
Creating a SAX parser
Using catalog files
Using namespaces
Using the revalidation API

Chapter 3. Samples

Before you start

Before you can run the sample programs, you need to add the IBM XML Parser for Java Examples feature to the VisualAge for Java workspace.

To add the feature to the workspace, do the following:

1. In the VisualAge for Java Workbench, click **File > Quick Start**. The Quick Start window opens.
2. In the Quick Start window, click **Features** in the left column, click **Add Feature** in the right column, and click **OK**.
3. In the Selection Required page, select **IBM XML Parser for Java Examples**, then click **OK**. The feature is added to the workspace.

Once the feature is added to the workspace, you can run the sample programs.

RELATED CONCEPTS

Overview
Limitations

RELATED TASKS

Using a parser factory
Explicitly instantiating a parser class
Extending a parser class
Creating a DOM parser
Creating a SAX parser
Using catalog files
Using namespaces
Using the revalidation API
Handling errors

SAXWriter and DOMWriter

SAXWriter and DOMWriter parse your input file and print it out in XML format. You can use a command line option to print in a canonical XML format, so you can use the output to compare XML documents. SAXWriter and DOMWriter also display any errors or warnings that occurred during the parse.

SAXWriter uses either the validating or non-validating SAX parser. DOMWriter uses either the validating or non-validating DOM parser.

Source code
To see all the sample programs, look in the VisualAge for Java Workbench for the IBM XML Parser for Java Examples project.

Running SAXWriter
To run SAXWriter, do the following:

1. In the VisualAge for Java Workbench, expand the **IBM XML Parser for Java Examples** project.
2. Expand the **sax** package.

3. Right-click the **SAXWriter** class, and in the pop-up menu click **Properties**.
4. In the Properties dialog box, click the **Program** tab.
5. The **Command line arguments** field contains the name of the XML document SAXWriter will parse:

   ```
   data\personal.xml
   ```

   You can leave the default name as is or specify a different XML document to parse. You can also specify other command line options. Refer to Command line options for SAXWriter (page 16) for details.
6. Click **OK**.
7. Right-click the **SAXWriter** class, and in the pop-up menu click **Run > Run main**.
8. The Console window displays the output from the parser.

Command line options for SAXWriter
SAXWriter supports the following command line options:

-p *parserName*	Specify the parser class to be used. The available parsers are: • com.ibm.xml.parsers.SAXParser (default) • com.ibm.xml.parsers.ValidatingSAXParser
-h	Display the SAXWriter help information in the Console window. The default is no help.
-c *XMLdocument*	Output in canonical format. The default is normal format.

Running DOMWriter
To run DOMWriter, do the following:

1. In the VisualAge for Java Workbench, expand the **IBM XML Parser for Java Examples** project.
2. Expand the **dom** package.
3. Right-click the **DOMWriter** class, and in the pop-up menu click **Properties**.
4. In the Properties dialog box, click the **Program** tab.
5. The **Command line arguments** field contains the name of the XML document DOMWriter will parse:

   ```
   data\personal.xml
   ```

   You can leave the default name as is or specify a different XML document to parse. You can also specify other command line options. Refer to Command line options for DOMWriter (page 16) for details.
6. Click the **OK**.
7. Right-click the **DOMWriter** class, and in the pop-up menu click **Run > Run main**.
8. The Console window displays the output from the parser.

Command line options for DOMWriter
DOMWriter supports the following command line options:

-p *parserName*	Specify the parser class to be used. The available parsers are: • dom.wrappers.DOMParser (default) • dom.wrappers.NonValidatingDOMParser • dom.wrappers.TXParser

-h	Display the DOMWriter help information in the Console window. The default is no help.
-c *XMLdocument*	Output in canonical format. The default is normal format.
-e *encodingName*	Output using the specified encoding. The default is UTF8. Specifying the **-e** option with no encoding name displays a list of valid encoding names.

RELATED CONCEPTS

Overview
Limitations

RELATED TASKS

Using a parser factory
Explicitly instantiating a parser class
Extending a parser class
Creating a DOM parser
Creating a SAX parser
Using catalog files
Using namespaces
Using the revalidation API
Handling errors

SAXCount and DOMCount

SAXCount and DOMCount invoke the parser on an XML document and print out interesting information about the document, such as the total parse time, along with counts of elements, attributes, text characters, and ignorable whitespace characters. SAXCount and DOMCount also display any errors or warnings that occurred during the parse.

SAXCount uses either the validating or non-validating SAX parser. DOMCount uses either the validating or non-validating DOM parser.

Source code
To see the sample programs, look in the VisualAge for Java IDE for the IBM XML Parser for Java Examples project.

Running SAXCount
To run SAXCount, do the following:

1. In the VisualAge for Java Workbench, expand the **IBM XML Parser for Java Examples** project.
2. Expand the **sax** package.
3. Right-click the **SAXCount** class, and in the pop-up menu click **Properties**.
4. In the Properties dialog box, click the **Program** tab.
5. The **Command line arguments** field contains the name of the XML document SAXCount will parse:

   ```
   data\personal.xml
   ```

   You can leave the default name as is or specify a different XML document to parse. You can also specify other command line options. Refer to Command line options for SAXCount (page 18) for details.
6. Click **OK**.

7. Right-click the **SAXCount** class, and in the pop-up menu click **Run > Run main**.
8. The Console window displays the output from the parser.

Command line options for SAXCount
SAXCount supports the following command line options:

-p *parserName*	Specify the parser class to be used. The available parsers are: • com.ibm.xml.parsers.SAXParser (default) • com.ibm.xml.parsers.ValidatingSAXParser
-h	Print the SAXWriter help information in the Console window. The default is no help.

Running DOMCount
To run DOMCount, do the following:

1. In the VisualAge for Java Workbench, expand the **IBM XML Parser for Java Examples** project.
2. Expand the **dom** package.
3. Right-click the **DOMCount** class, and in the pop-up menu click **Properties**.
4. In the Properties dialog box, click the **Program** tab.
5. The **Command line arguments** field contains the name of the XML document DOMCount will parse:

   ```
   data\personal.xml
   ```

   You can leave the default name as is or specify a different XML document to parse. You can also specify other command line options. Refer to Command line options for DOMCount (page 18) for details.
6. Click **OK**.
7. Right-click the **DOMCount** class, and in the pop-up menu click **Run > Run main**.
8. The Console window displays the output from the parser.

Command line options for DOMCount
SAXCount supports the following command line options:

-p *parserName*	Specify the parser class to be used. The available parsers are: • dom.wrappers.DOMParser (default) • dom.wrappers.NonValidatingDOMParser • dom.wrappers.TXParser
-h	Print the DOMWriter help information in the Console window. The default is no help.

RELATED CONCEPTS

Overview
Limitations

RELATED TASKS

Using a parser factory
Explicitly instantiating a parser class
Extending a parser class

Creating a DOM parser
Creating a SAX parser
Using catalog files
Using namespaces
Using the revalidation API
Handling errors

TreeViewer

TreeViewer displays the input file in a graphical tree-based interface. This sample highlights the error handling capabilities of the parser, demonstrating how the parser can recover from many types of common errors.

Source code
To see the sample programs, look in the VisualAge for Java IDE for the IBM XML Parser for Java Examples project.

Running TreeViewer
To run TreeViewer, do the following:

1. In the VisualAge for Java Workbench, expand the **IBM XML Parser for Java Examples** project.
2. Expand the **ui** package.
3. Right-click the **TreeViewer** class, and in the pop-up menu click **Properties**.
4. In the Properties dialog box, click the **Program** tab.
5. The **Command line arguments** field contains the name of the XML document TreeViewer will parse:

   ```
   data\personal.xml
   ```

   You can leave the default name as is or specify a different XML document to parse.
6. Click **OK**.
7. Right-click the **TreeViewer** class, and in the pop-up menu click **Run > Run main**.
8. The graphical user interface displays a tree-view of the XML document in the left pane, and a XML source-view in the right pane.

RELATED CONCEPTS

Overview
Limitations

RELATED TASKS

Using a parser factory
Explicitly instantiating a parser class
Extending a parser class
Creating a DOM parser
Creating a SAX parser
Using catalog files
Using namespaces
Using the revalidation API
Handling errors

DOMFilter

DOMFilter parses an XML document, searching for specific elements by name, or elements with specific attributes. It uses the getElementsByTagName() method to traverse the DOM tree, looking for the elements or attributes that match your specification.

Source code
To see the sample programs, look in the VisualAge for Java IDE for the IBM XML Parser for Java Examples project.

Running DOMFilter
To run DOMFilter, do the following:

1. In the VisualAge for Java Workbench, expand the **IBM XML Parser for Java Examples** project.
2. Expand the **dom** package.
3. Right-click the **DOMFilter** class, and in the pop-up menu click **Properties**.
4. In the Properties dialog box, click the **Program** tab.
5. The **Command line arguments** field contains the name of the XML document DOMFilter will parse:

   ```
   data\personal.xml
   ```

   You can leave the default name as is or specify a different XML document to parse. You can also specify other command line options. Refer to Command line options for DOMFilter (page 20) for details.
6. Click **OK**.
7. Right-click the **DOMFilter** class, and in the pop-up menu click **Run > Run main**.
8. The Console window displays the output from the parser.

Command line options for DOMFilter
DOMFilter supports the following command line options:

-p *parserName*	Specify the parser class to be used. The available parsers are: • dom.wrappers.DOMParser (default) • dom.wrappers.NonValidatingDOMParser • dom.wrappers.TXParser
-h	Print the DOMWriter help information in the Console window. The default is no help.
-e *elementName*	Specify the name of the element for which to search. The default is to match all elements.
-a *attributeName*	Specify the name of the attribute for which to search. The default is to match all attributes.

RELATED CONCEPTS

Overview
Limitations

RELATED TASKS

Using a parser factory
Explicitly instantiating a parser class

Extending a parser class
Creating a DOM parser
Creating a SAX parser
Using catalog files
Using namespaces
Using the revalidation API
Handling errors

Notices

Note to U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.
IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.
This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM Canada Ltd.
1150 Eglinton Avenue East
Toronto, Ontario M3C 1H7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.
The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:
© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 1997, 2000. All rights reserved.

Programming interface information

Programming interface information is intended to help you create application software using this program.

General-use programming interfaces allow the customer to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

- AIX
- AS/400
- DB2
- CICS
- CICS/ESA
- IBM
- IMS
- Language Environment
- MQSeries
- Network Station
- OS/2
- OS/390
- OS/400
- RS/6000
- S/390
- VisualAge
- VTAM
- WebSphere

Lotus, Lotus Notes and Domino are trademarks or registered trademarks of Lotus Development Corporation in the United States, or other countries, or both.

Tivoli Enterprise Console and Tivoli Module Designer are trademarks of Tivoli Systems Inc. in the United States, or other countries, or both.

Encina and DCE Encina Lightweight Client are trademarks of Transarc Corporation in the United States, or other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

ActiveX, Microsoft, SourceSafe, Visual C++, Visual SourceSafe, Windows, Windows NT, Win32, Win32s and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, or other countries, or both.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Intel and Pentium are trademarks of Intel Corporation in the United States, or other countries, or both.

Other company, product, and service names, which may be denoted by a double asterisk(**), may be trademarks or service marks of others.