

# Investigating use of XML in Java application codes.

**Project Title:** ??

**Document Title:** Investigating use of XML in Java application codes.

**Document Identifier:** EPCC-UKHEC-XML-REPORT 1.1

**Distribution Classification:** ??

**Authorship:** Lindsay Pottage

## Document history:

Personnel	Date	Summary	Version
Lindsay Pottage	15 June 2001	First Draft	1.0
Lindsay Pottage	29 June 2001	Revision of First Draft	1.1

## Approval List:

David Henty

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>XML - Extensible Markup Language</b>	<b>3</b>
2.1	Background . . . . .	3
2.2	Data representation in XML . . . . .	3
<b>3</b>	<b>Java and XML</b>	<b>5</b>
3.1	SAX and DOM parsers . . . . .	5
3.2	SAX-based Tutorial . . . . .	6
3.2.1	Data representation in XML . . . . .	6
3.2.2	Setting up the SAX parser . . . . .	8
3.2.3	Parsing the XML document . . . . .	9
3.2.4	Accessing data from the object model . . . . .	14
3.2.5	Writing XML in Java applications . . . . .	17
<b>4</b>	<b>Conclusions</b>	<b>18</b>
	<b>References</b>	<b>18</b>
	<b>Appendix A UML notation</b>	<b>19</b>

# 1 Introduction

This document is a practical overview of how to use XML in user applications. Within the context of XML we will focus on Web-based Java applications since these two technologies constitute the core of Web services today; Java providing the tools to develop secure, portable and platform-independent Web-based applications and XML their means of communication via structured, platform-independent data representation and exchange [1].

In this document we describe how XML and Java work together using, as a case study, software recently developed at EPCC for a Web-based prototype ePortal to HPC applications [2].

We start by presenting a brief overview of XML in section 2 and follow on in section 3 to detail how XML and the data it represents can be manipulated and used by Java. Finally we discuss ... in section 4.

## 2 XML - Extensible Markup Language

### 2.1 Background

XML (the Extensible Markup Language) is a restricted form of SGML (the Standard Generalized Markup Language) specially designed for Web applications [4]. SGML, in general terms, is a system which enables documents to describe their own grammar and structural relationships via a set of tags [5]. However, the full SGML standard is very complex and contains many infrequently used features.

XML was developed to address the limitations of HTML (Hypertext Markup Language), the now familiar SGML-based language used for storage and transmission of documents over the web [5]. HTML is a specific application of SGML, which defines simple hypertext and multimedia supportive documents of fixed type via a hardwired SGML-conforming tag set [6]. However, due to its simplicity, HTML cannot offer the functionality demanded by Web applications today, namely:

- *Extensibility* HTML does not allow users to specify their own tags or attributes in order to parameterize or otherwise semantically qualify their data.
- *Structure* HTML does not support the specification of deep structures needed to represent database schemas or object-oriented hierarchies.
- *Validation* HTML does not support the kind of language specification that allows applications to check data for structural validity on importation [5].

Subsequently, XML was specifically devised to retain these features already provided by SGML via a simplified language designed for straightforward use over the Internet [5]. XML is a language for defining markup languages; it is more like SGML light, than HTML++ [7].

### 2.2 Data representation in XML

Please note that the description provided here is only intended to present a general overview of XML for the 'first-timer'. For more detailed and accurate information on XML please refer to

the XML 1.0 specification document [4].

XML defines a class of data objects, termed XML documents, which store language- and platform-independent information in plain text [8].

Textual information within XML documents takes either the form of character data or markup, where markup provides a description of the document's layout composition and logical structure. Markup may take the form of start-tags, end-tags, character references, comments, processing instructions and document type declarations [4].

As one of the principle qualities of XML enables the user to define their own markup language, valid XML documents should begin with a DTD (Document Type Declaration) which provides a definition of the markup language used in the ensuing document. Figure 1 describes a simple but valid XML document.

```
<?xml version='1.0' standalone='yes'?>

<!--Comment example: DTD-->

<!DOCTYPE phonebook [
  <!ELEMENT person (name, number, address)+>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT number (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
]>

<!--Comment example: Body of XML document-->

<phonebook>

  <person>

    <name>Smith.S</name>
    <number>0131 678 0985</number>
    <address>8 FountainPark Ave, EH9 1PQ</address>

  </person>

  <person>

    <name>Brown.B</name>
    <number>0131 235 1981</number>
    <address>2/1 39 Dundee Terrace, EH3 3FG</address>

  </person>

</phonebook>
```

Figure 1: A valid XML document.

In the DTD, `<!ELEMENT>` describes a document element which has a type, identified by name, and may also have a list of attributes specified by name and value. This declaration predefines the storage units of the subsequent XML document, the structure of which must comply with the DTD to be rendered valid [4]. For more details of the syntax see sub section 3.2.

So far we have provided a brief description of how data is represented in XML. Next we will

move on to detail how that data can be accessed and utilised by software applications.

As a pre-introduction, an XML processor is required to access and read XML documents. This utility is employed on behalf of the software application, which can then manipulate and use the data retrieved to its own advantage. In section 3 we describe how this is achieved in detail, using specific examples of Web-enabled Java software developed in-house at EPCC [2].

### 3 Java and XML

It is worth noting that any programming language can output data from an XML document [6]; however Java offers the most compatible environment to XML in terms of platform-neutrality, object-oriented structure and the industry-wide adoption of such technologies as Enterprise JavaBeans, RMI, JDBC and servlets [1] for distributed network and Web-enabled applications.

As mentioned previously, programme access to the data contained within XML documents can only be gained through an XML processor or “parser”, of which there are two very different types; SAX (Simple API for XML) and DOM (Document Object Model) [9].

#### 3.1 SAX and DOM parsers

SAX and DOM API's represent open, language-independent interfaces to XML document parsing developed by the XML-DEV and W3C groups respectively [11] and have been implemented in several languages; Java, C++, Perl and Python [12]. However that is where their similarity ends.

The DOM API provides an all-inclusive interface to XML document processing:

- It serially reads the XML document.
- It provides, and *forces*, a default object model which encapsulates the data within a tree structure of objects based on its hierarchy within the XML document.
- It provides standard mechanisms for accessing and manipulating the data.

The DOM API is best suited for applications which wish to preserve the hierarchical structure of the data within the XML document, i.e. document data, where the data should be preserved in the sequence it is defined.

In contrast, the SAX API provides minimal functionality but at the same time enables maximum flexibility for data encapsulation:

- It serially reads the XML document.
- It fires events for each element tag encountered.

It is then the responsibility of the software designer to interpret each event and generate an appropriate object model to store the data retrieved. Thus the SAX API enables developers to customize the object model used by the application to access and manipulate the data [10].

This API is best suited for applications dealing with structured data, i.e. Java object properties stored in XML format, which would require customized data structures and classes to store the information appropriately [12].

It is beyond the scope of this report to provide a more detailed comparison of these two APIs. However, the SAX API appears the more natural for parsing the simple parameter files typically used in scientific applications. For this reason we will continue to discuss SAX in a detailed tutorial format. For more information on DOM and relevant application areas please reference [11] and [12].

## 3.2 SAX-based Tutorial

We recently developed a Web-based software system utilising Java and XML called the *ePortal*; a fully service-based and user-driven Grid portal for large-scale computational science applications. The system is designed to be as general as possible, but its development was driven by the requirements of a specific HPC research group: the MHD Consortium [3]. Java and XML were used to satisfy the system requirements of platform neutrality and ease of extension [2].

The system architecture is based upon a three-tier generic client - portal - High Performance Computing (HPC) server structure. XML is used as the means of data exchange between each tier and, in particular, provides the client tier with information necessary for the user to run large-scale computational science applications on the back end HPC servers. The client itself constitutes a Java applet within a Web browser.

We will now present how this information is stored, parsed and encapsulated into an object model for use by the client applet.

### 3.2.1 Data representation in XML

The client applet requires information on distinct scientific 'consortia', including lists of consortium-associated scientific codes and HPC servers on which these codes can run. To store this information, an XML document was generated of which a representative subsection is shown in figure 2. This describes a code, Lare2D, written by Tony Arber of the MHD Consortium which runs in parallel on the PPARC funded Compaq MHD Cluster in St. Andrews.

The DTD of this document describes a container, "userServices", which holds consortium and site (HPC server) elements. It also denotes that a consortium element must contain two further elements; "consort\_name" and "code" which define the name of the consortium in general character data and a scientific code respectively. General character data is defined in the DTD using the #PCDATA (parsed character data) declaration. A code element in turn, must contain three elements; "code\_name", "code\_desc" and "code\_site" which define character data representing the name and description of the code and the name of the sites on which the code may run respectively. Lastly, a site element may contain up to seven elements; "site\_name", "remote", "host\_name", "port", "hardware\_desc", "software\_desc" and "unspec\_desc" which define character data representing the name of the site, whether it is remote or not, the server host if it is remote and port number, a description of the hardware and software available or a general description if not applicable. This format is followed in the ensuing document in which all relative data is contained.

Please note, in some cases an operator is assigned to an element. Each operator denotes how often that element may occur in the XML document; one or more times (+), zero or more times (\*) and zero or once (?). The absence of an operator simply means that the element must occur

```
<?xml version = ``1.0`` standalone=``yes``?>
<!DOCTYPE userServices [
  <!ELEMENT userServices (consortium+, site+)>
  <!ELEMENT consortium (cosort_name, code+)>
  <!ELEMENT consort_name (#PCDATA)>
  <!ELEMENT code (code_name, code_description, code_site+)>
  <!ELEMENT code_name (#PCDATA)>
  <!ELEMENT code_description (#PCDATA)>
  <!ELEMENT code_site (#PCDATA)>
  <!ELEMENT site (site_name, remote, host_name?, port?, hardware_desc?,
    software_desc?, unspec_desc?)>
  <!ELEMENT site_name (#PCDATA)>
  <!ELEMENT remote (#PCDATA)>
  <!ELEMENT host_name (#PCDATA)>
  <!ELEMENT port (#PCDATA)>
  <!ELEMENT hardware_desc (#PCDATA)>
  <!ELEMENT software_desc (#PCDATA)>
  <!ELEMENT unspec_desc (#PCDATA)>
]>

<userServices>

  <consortium>
    <consort_name>MHD Consortium</consort_name>
    <code>
      <code_name>Lare2d</code_name>
      <code_description>Numerical simulation of solar coronal
        loops</code_description>
      <code_site>St_Andrews</code_site>
    </code>
  </consortium>

  <site>
    <site_name>St_Andrews</site_name>
    <remote>yes</remote>
    <host_name>mhd0.st-and.ac.uk</host_name>
    <port>8567</port>
    <hardware_desc>Hardware: Six Compaq ES40s (each with 4 Digital ev6 CPU's and
      4Gbytes of RAM).</hardware_desc>
    <software_desc>Software Installed: f77, f90, HPF, MPI, OpenMP, c,
      c++.</software_desc>
  </site>

</userServices>
```

Figure 2: ePortal User Services XML document.

exactly once [4]. In addition, although all data is defined in plain text, it may be converted to any data type by the application during parsing.

Consortium, code, site elements and their inner 'contents' require to be converted into individual Java classes and associated attributes respectively for use by the applet to enable user interaction. This is achieved as follows.

### 3.2.2 Setting up the SAX parser

Parsing of the XML document requires a Java implementation of the SAX parser. IBM XML for Java, OpenXML parser, DataChannel - Microsoft XML Parser for Java (Beta 2), Sun Java Project X[XML library] parser and Oracle's XML Parser for Java v2 represents just a selection of XML parsers written in Java that are available [14]. We used Sun's XML parser for Java [11].

The first order of business is to import the correct packages into the class which will undertake parsing of the XML document.

```
import java.io.*;
import org.xml.sax.*;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
```

The `org.xml.sax` package contains the class `HandlerBase`, which provides empty implementations for four interfaces required by the SAX parser to handle all possible events fired: `DocumentHandler`, `EntityResolver`, `DTDHandler` and `ErrorHandler` [9]. By extending the `HandlerBase` class, you negate the need to implement each method in the four interfaces, implementing only the methods relevant to your application.

```
import java.io.*;
import org.xml.sax.*;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class ePUIDataParser extends HandlerBase {
}
```

Now, we need to create the SAX parser. As mentioned before, the basic model is that an event will be fired for each element tag encountered which the application must handle appropriately.

An instance of the SAX parser can be retrieved from the `javax.xml.parsers.SAXParserFactory` class. Once retrieved, the SAX parser requires to be given the class which will handle the events fired and the data to be parsed. Data may be passed to the parser as an `InputStream`, `URL` (`String`) or `File` object: we used a `ByteArrayInputStream`.

```
import java.io.*;
import org.xml.sax.*;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
```

```
public class ePUIDataParser extends HandlerBase {

    public ePUIDataParser (byte[] data_input){

        SAXParserFactory factory = SAXParserFactory.newInstance();

        try{

            /* Create ByteArrayInputStream to pass xml data. */
            ByteArrayInputStream xml_data =
                new ByteArrayInputStream(data_input,0,data_input.length);

            /* Create a new instance of the SAX parser to parse the input. */
            SAXParser sax_parser = factory.newSAXParser();
            sax_parser.parse(xml_data, this);
        }
        catch(Throwable t){
            t.printStackTrace();
        }
    }
}
```

### 3.2.3 Parsing the XML document

Next, we need to handle the events fired by the parser we have just created. For our purposes we implemented methods of the `DocumentHandler` class, namely: `startElement`, `endElement` and `characters`. For further information on event-handling methods for any of the aforementioned interfaces, please reference [13].

`startElement`, `endElement` and `characters` methods are notified when the parser encounters the beginning of an element, the end of an element and character data respectively. As will become clear in the following example, the status of the parser is managed by the `startElement` and `endElement` methods, whereby the name of the tag is passed as an argument to the appropriate method. Character data situated in between a start and end tag is passed as a character array to the `characters` method to be utilised as desired.

We required the data contained in XML format (figure 2) to be encapsulated into the object model described in figure 3 (appendix A provides a definition of UML notation if required). This model demonstrates that individual consortium, code and site elements require to be converted in instances of `ePConsortium`, `ePCode` and `ePHpcSite` classes respectively. In addition, instances of each class are contained within, and accessed by the applet through, the `ePUIDataContainer` class. The code required to generate this object model is as follows:

Instances of each Java class are initialised prior to parsing of the XML document.

```
import java.io.*;
import org.xml.sax.*;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class ePUIDataParser extends HandlerBase {

    /* Declare class instances. */
```

```
private ePUIDataContainer ep_datacontainer;
private ePConsortium ep_consortium;
private ePHpcSite ep_hpcsite;
private ePCode ep_code;

/* Declare String to hold the name of current element (status). */
private String current_element;

/* Declare Strings to reference XML element tags. */
private static final String CONSORTIUM = "consortium";
private static final String CONSORTIUM_NAME = "consort_name";
private static final String CODE = "code";
private static final String CODE_NAME = "code_name";
private static final String CODE_DESC = "code_description";
private static final String CODE_SITE = "code_site";
private static final String SITE = "site";
private static final String SITE_NAME = "site_name";
private static final String REMOTE = "remote";
private static final String HOST_NAME = "host_name";
private static final String PORT = "port";
private static final String HARDWARE_DESC = "hardware_desc";
private static final String SOFTWARE_DESC = "software_desc";
private static final String UNSPEC_DESC = "unspec_desc";

/**
 * Constructor.
 * Create SAXParser objects to parse the argument passed.
 * @param servcies_input references byte[] array containing the
 * XML services data.
 */
public ePUIDataParser (byte[] data_input){

    /* Initialise object instances. */
    ep_datacontainer = new ePUIDataContainer();
    ep_consortium = null;
    ep_code = null;
    ep_hpcsite = null;

    current_element = null; /* Initialise String status object. */

    /* Create a new SAX parser instance. */
    SAXParserFactory factory = SAXParserFactory.newInstance();

    try{

        /* Create ByteArrayInputStream to pass xml data. */
        ByteArrayInputStream data_xml =
            new ByteArrayInputStream(data_input,0,data_input.length);

        /* Create a new instance of the SAX parser to parse the input. */
        SAXParser sax_parser = factory.newSAXParser();
        sax_parser.parse(data_xml, this);
    }
    catch(Throwable t){
        t.printStackTrace();
    }
}
```

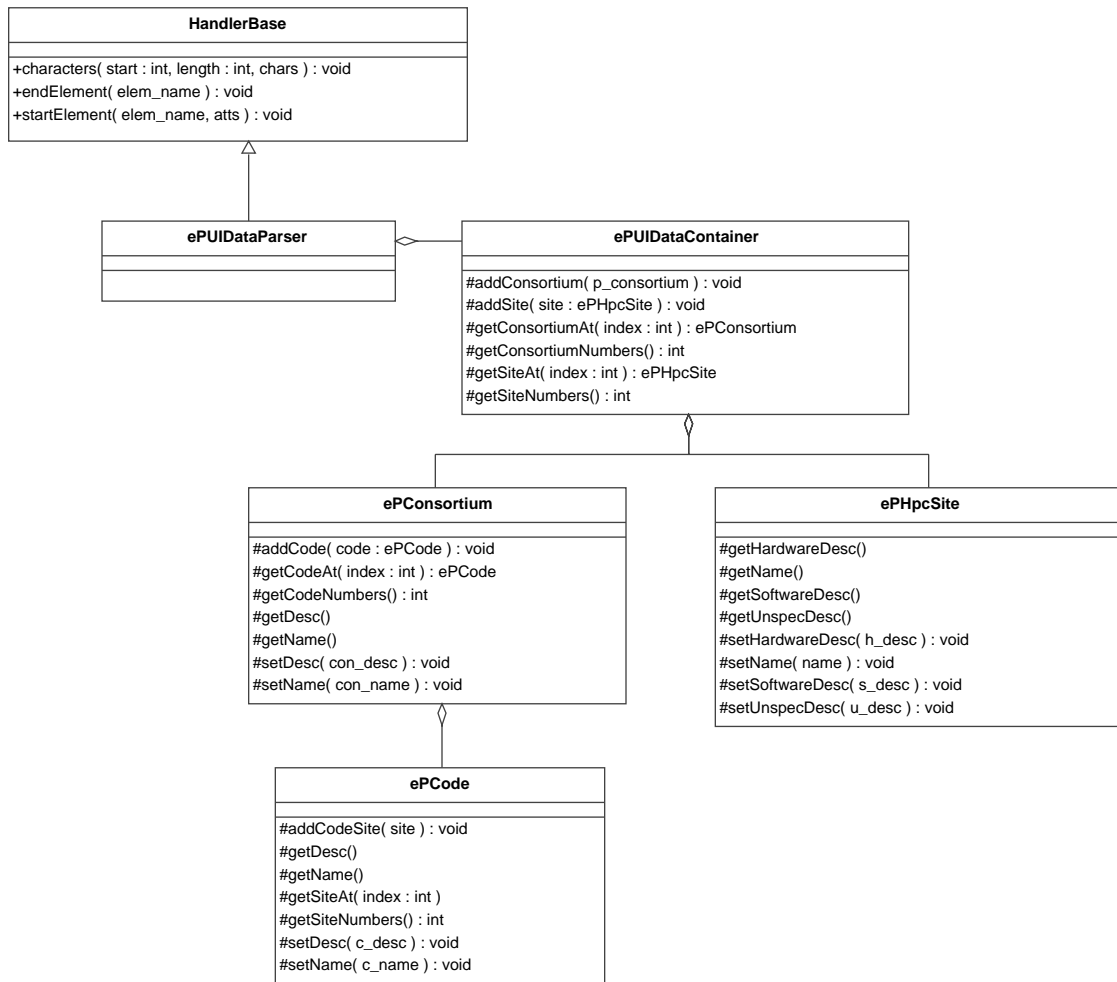


Figure 3: Object Model of XML document data.

Once parsing has commenced, instances of the appropriate `ePConsortium`, `ePCode` or `ePHpcSite` class are created when the respective XML element start tag is encountered. The name of all other elements, once encountered, is stored by the `current_element` variable for use within the `characters` method.

```

/**
 * startElement Method.
 * This method is called when the SAX parser encounters an open element
 * tag.
 * @param elem_name String object representing the name of the tag element.
 * @param atts      AttributeList object reference to any associated
 *                  attributes.
 */
public void startElement(String elem_name, AttributeList atts)
    throws SAXException {

    if(elem_name.equalsIgnoreCase(CONSORTIUM)){

        /* Create an object of the ePConsortium class.*/
        ePConsortium = new ePConsortium();
    }
}

```

```

    }
    else if(elem_name.equalsIgnoreCase(CONSORTIUM_NAME)){
        current_element = CONSORTIUM_NAME;
    }
    else if(elem_name.equalsIgnoreCase(CODE)){

        /* Create an object of the ePCode class.*/
        ep_code = new ePCode();
    }
    else if(elem_name.equalsIgnoreCase(CODE_NAME)){
        current_element = CODE_NAME;
    }
    else if(elem_name.equalsIgnoreCase(CODE_DESC)){
        current_element = CODE_DESC;
    }
    else if(elem_name.equalsIgnoreCase(CODE_SITE)){
        current_element = CODE_SITE;
    }
    else if(elem_name.equalsIgnoreCase(SITE)){

        /* Create an object of the ePHpcSite class.*/
        ep_hpcsite = new ePHpcSite();
    }
    else if(elem_name.equalsIgnoreCase(SITE_NAME)){
        current_element = SITE_NAME;
    }
    else if(elem_name.equalsIgnoreCase(REMOTE)){
        current_element = REMOTE;
    }
    else if(elem_name.equalsIgnoreCase(HOST_NAME)){
        current_element = HOST_NAME;
    }
    else if(elem_name.equalsIgnoreCase(PORT)){
        current_element = PORT;
    }
    else if(elem_name.equalsIgnoreCase(HARDWARE_DESC)){
        current_element = HARDWARE_DESC;
    }
    else if(elem_name.equalsIgnoreCase(SOFTWARE_DESC)){
        current_element = SOFTWARE_DESC;
    }
    else if(elem_name.equalsIgnoreCase(UNSPEC_DESC)){
        current_element = UNSPEC_DESC;
    }
}

```

When the parser encounters character data it notifies the `characters` method. By storing the name of the element start tag just encountered within the `current_element` variable, we now know to which object instance this data should be assigned.

```

/**
 * characters Method.
 * This method is called when the SAX parser encounters #PCDATA.
 * @param chars char array of data contained between element tags.
 * @param start starting index of char array to be read.
 * @param length length of char array to be read.
 */
public void characters(char chars[], int start, int length)

```

```
throws SAXException {

/* Create String object using arguments passed. */
String value_retrieved = new String(chars, start, length);

if(!value_retrieved.trim().equals("")){

    if(current_element.equalsIgnoreCase(CONSORTIUM_NAME)){

        /* Set the name of a particular ePConsortium object.*/
        ep_consortium.setName(value_retrieved);
    }
    else if(current_element.equalsIgnoreCase(CODE_NAME)){

        /* Set the name of a particular ePCode object.*/
        ep_code.setName(value_retrieved);
    }
    else if(current_element.equalsIgnoreCase(CODE_DESC)){

        /* Set the description of a particular ePCode object.*/
        ep_code.setDesc(value_retrieved);
    }
    else if(current_element.equalsIgnoreCase(CODE_SITE)){

        /*
        * Add the name of a ePHpcSite object to a particular
        * ePCode object.
        */
        ep_code.addCodeSite(value_retrieved);
    }
    else if(current_element.equalsIgnoreCase(SITE_NAME)){

        /* Set the name of a particular ePHpcSite object.*/
        ep_hpcsite.setName(value_retrieved);
    }
    else if(current_element.equalsIgnoreCase(HARDWARE_DESC)){

        /*
        * Set the hardware description of a particular ePHpcSite
        * object.
        */
        ep_hpcsite.setHardwareDesc(value_retrieved);
    }
    else if(current_element.equalsIgnoreCase(SOFTWARE_DESC)){

        /*
        * Set the software description of a particular ePHpcSite
        * object.
        */
        ep_hpcsite.setSoftwareDesc(value_retrieved);
    }
    else if(current_element.equalsIgnoreCase(UNSPEC_DESC)){

        /*
        * Set the unspecified description of a particular ePHpcSite
        * object.
        */
        ep_hpcsite.setUnspecDesc(value_retrieved);
    }
}
```

```

    }
}

```

Lastly within the endElement method, each instance of either the ePConsortium, ePCode or ePHpcSite class is added to it's appropriate 'container' when the respective XML element end tag is encountered.

```

/**
 * endElement Method.
 * This method is called when the SAX parser encounters an end element
 * tag.
 * @param elem_name String object representing the name of the tag element.
 */
public void endElement(String elem_name) throws SAXException {

    if(elem_name.equalsIgnoreCase(CONSORTIUM)){

        /* Add ePConsortium object to vector of ePUIDataContainer object.*/
        ep_datacontainer.addConsortium(ep_consortium);
        ep_consortium = null; /* Reset ep_consortium object. */
    }
    else if(elem_name.equalsIgnoreCase(CODE)){

        /* Add ePCode object to vector of ePConsortium object.*/
        ep_consortium.addCode(ep_code);
        ep_code= null; /* Reset ePCode object. */
    }
    else if(elem_name.equalsIgnoreCase(SITE)){

        /* Add ePHpcSite object to vector of ePUIDataContainer object.*/
        ep_datacontainer.addSite(ep_hpcsite);
        ep_hpcsite = null; /* Reset ePHpcSite object. */
    }
}
}
}

```

### 3.2.4 Accessing data from the object model

The sample code presented here consititutes a very simple example of how a handler class and its methods can be implemented to yield an object model representation of data contained in XML format.

To complete the tutorial we have included sample code to demonstrate how this data can be accessed from the object model. In this example, a complete version of the ePortal services XML document is parsed and a list of possible HPC sites available, their hardware and software descriptions and all codes which run on them, is produced.

```

public ePUIDataParser (byte[] data_input){

    /* Initialise object instances. */
    ep_datacontainer = new ePUIDataContainer();
    ep_consortium = null;
    ep_code = null;
    ep_hpcsite = null;
}

```

```
current_element = null; /* Initialise String status object. */

/* Create a new SAX parser instance. */
SAXParserFactory factory = SAXParserFactory.newInstance();

try{

    /* Create ByteArrayInputStream to pass xml data. */
    ByteArrayInputStream data_xml =
        new ByteArrayInputStream(data_input,0,data_input.length);

    /* Create a new instance of the SAX parser to parse the input. */
    SAXParser sax_parser = factory.newSAXParser();
    sax_parser.parse(data_xml, this);
}
catch(Throwable t){
    t.printStackTrace();
}

/* Once parsing is complete retrieve all sites and relevent codes. */

int number_of_sites = ep_datacontainer.getSiteNumbers();
ePHpcSite site = null;
ePConsortium consortium = null;
ePCode code = null;

for(int i=0; i<number_of_sites; i++) {

    site = ep_datacontainer.getSiteAt(i);

    String site_name = site.getName();
    String site_hardware = site.getHardwareDesc();
    String site_software = site.getSoftwareDesc();

    System.out.println(``Site name: `` + site_name);
    System.out.println();
    System.out.println(site_hardware);
    System.out.println();
    System.out.println(site_software);
    System.out.println();

    System.out.println(``Codes available at this site: ``);

    int consortium_numbers = ep_datacontainer.getConsortiumNumbers();

    for(int j=0; j<consortium_numbers; j++) {

        consortium = ep_datacontainer.getConsortiumAt(i);
        int code_numbers = consortium.getCodeNumbers();

        for(int k=0; k<code_numbers; k++) {

            code = consortium.getCodeAt(i);
            int site_numbers = code.getSiteNumbers();

            for(int m=0; m<site_numbers; m++) {

                String code_site = code.getSiteAt(m);
```

```
        if(code_site.equals(site_name)){
            System.out.println(code.getName());
        }
    }
    code = null; /* Reset code. */
}
consortium = null; /* Reset consortium. */
}
site = null; /* Reset site. */

System.out.println();
System.out.println();
}
}
```

Output:

Site name: EPCC T3E

Hardware: Total of 368 Digital 450 MHz Alpha (EV5.6) processors, each having a peak performance of 900 MFlops. The majority of processors are configured with 128 Mbytes or above of memory.

Software Installed: f90, HPF, MPI, c.

Codes available at this site:

Lare2d

Site name: St Andrews

Hardware: Six Compaq ES40s (each with 4 Digital ev6 CPU's and 4Gbytes of RAM).

Software Installed: f77, f90, HPF, MPI, OpenMP, c, c++.

Codes available at this site:

Lare2d

Lare3d

Hydra\_mpi

Site name: UCL

Hardware: Two Silicon Graphics Origin 2000 Supercomputers. One with 48,300 MHz IP27 MIPS R12000 processor chips and 24 Gbytes of distributed main memory and the other 24,300 MHz IP27 MIPS R12000 processor chips and 6 Gbytes of distributed main memory.

Software Installed: Both computers have MIPSpro compilers installed (v7.2.1) which include f77, f90, c, c++.

Codes available at this site:

Lare3d

### 3.2.5 Writing XML in Java applications

From a reverse angle and as an additional point, XML can also be generated very simply by Java classes. In our system, XML is used as the means of data exchange between each tier, and subsequently Java classes at each tier require to generate XML.

The following method is implemented by a Java class on the client side which requires to notify the middle-tier of the service selection made by the user:

```
/**
 * writeXml Method.
 * @return String containing all xml data required to specify all
 * ePortal selections made by the user.
 */
public String writeXml() {

    String xml_data = "";
    StringBuffer xml_data_sb = new StringBuffer(xml_data);

    /* Write DTD for XML file.*/
    xml_data = "<?xml version = ``1.0`` standalone='`yes`'?>";
    xml_data_sb.append(xml_data);

    xml_data = "<!DOCTYPE USERSELECTION [ ";
    xml_data_sb.append(xml_data);

    xml_data = "<!ELEMENT userSelection (code, site)>";
    xml_data_sb.append(xml_data);

    xml_data = "<!ELEMENT code (#PCDATA)>";
    xml_data_sb.append(xml_data);

    xml_data = "<!ELEMENT site (#PCDATA)>";
    xml_data_sb.append(xml_data);

    xml_data = " ]>";
    xml_data_sb.append(xml_data);

    /* Write XML file.*/
    xml_data = "<userSelection>";
    xml_data_sb.append(xml_data);

    xml_data = "    <code>" + getCodeSelected() + "</code>";
    xml_data_sb.append(xml_data);

    xml_data = "    <site>" + getSiteSelected() + "</site>";
    xml_data_sb.append(xml_data);

    xml_data = "</userSelection>";
    xml_data_sb.append(xml_data);

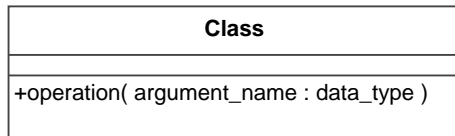
    String all_xml = xml_data_sb.toString();
    return all_xml;
}
```

## 4 Conclusions

### References

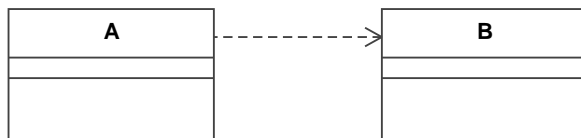
- [1] Sun Microsystems, Inc, **Using XML and Java technology to Develop Web Services: Keeping Migration Paths Open**, February 2001.
- [2] R.M.Baxter, S.P.Booth, E. Maura, L. Pottage, **Grid-Enabled Science: The EPCC ePortal**, Super Computing 2001 Proposal, Edinburgh Parallel Computing Centre, May 2001.
- [3] [http://www.epcc.ed.ac.uk/t3d/pparc\\_support/mhd.html](http://www.epcc.ed.ac.uk/t3d/pparc_support/mhd.html)
- [4] T.Bray, J.Paoli, C.M.Sperberg-McQueen, **Extensible Markup Language (XML) 1.0**, [www.w3.org/TR/1998/REC-xml-19980210](http://www.w3.org/TR/1998/REC-xml-19980210), February 1998.
- [5] J.Bosak, **XML, Java and the future of the Web**, Sun Microsystems, March 1997.
- [6] World Wide Web Consortium's XML Special Interest Group, **Frequently Asked Questions about the Extensible Markup Language**, June 1999.
- [7] L.M.Garshol, **Introduction to XML**, [http://www.stud.ifi.uio.no/lmar-iusg/download/xml/xml\\_eng.html](http://www.stud.ifi.uio.no/lmar-iusg/download/xml/xml_eng.html), August 1999.
- [8] N.Idris, **Benefits of using XML**, The Bean Factory, LLC, <http://www.developerlife.com>, June 1999.
- [9] N.Idris, **SAX Tutorial 1**, The Bean Factory, LLC, <http://www.developerlife.com>, May 1999.
- [10] N.Idris, **XML, Java, databases and the Web**, The Bean Factory, LLC, <http://www.developerlife.com>, June 1999.
- [11] **An overview of the APIs**
- [12] N.Idris, **Sould I use SAX or DOM**, The Bean Factory, LLC, <http://www.developerlife.com>, May 1999.
- [13] The Sun JAXP API, <http://java.sun.com/xml/jaxp-1.0.1/docs/api>
- [14] XML/XSL/XLink Software, <http://xml.coverpages.org/xml.html#xmlSoftware>

## Appendix A UML notation



### CLASS

Represents a class and it's operations. Each operation details the name and data type of all arguments.



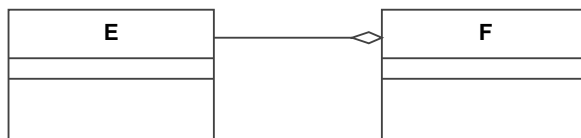
### DEPENDENCY

A dashed arrow represents a dependency between classes. Class A depends on class B.



### INHERITANCE

An open and solid arrow represents inheritance. Class C inherits from class D whereby class D is a more general class and class C is a specialized class.



### AGGREGATION

An open diamond represents aggregation between objects of classes. An object of class E is part of an object of class F.