input data and results, standardized data formats, and easier exchange and archival stability of data. In simple terms the main advantage of XML is that it is a human readable format; just by looking at an XML file, you can get an idea of the nature of the data.

The XML classes in the Open Source Physics library can be understood by reading the `XMLExampleApp` example. The XML API is very similar to the control API. For example, we use `setValue` to add data to an XML control, and we use `getInt`, `getDouble`, and `getString` to read data. We start by importing the necessary definitions from the controls package and defining the `main` method for the `ExampleXMLApp` class. Note that `XMLControl` defines an interface and `XMLControlElement` defines an implementation of this interface.

```
import org.opensourcephysics.controls.XMLControl;
import org.opensourcephysics.controls.XMLControlElement;

public class ExampleXMLApp {
   public static void main(String[] args) {
   ...
}
```

The following Java statements are placed in the body of the `main` method. An empty XML document is created using an `XMLControl` object by calling the `XMLControlElement` constructor without any parameters.

```
XMLControl xmlOut = new XMLControlElement();
```

Invoking the control's `setValue` method creates an XML element consisting of a tag and a data value. The tag is the first parameter and the data to be stored is the second. Data that can be stored includes numbers, number arrays, and strings. Because the tag is unique, the data can later be retrieved from the control using the appropriate get method.

```
xmlOut.setValue("comment", "An XML description of an array.");
xmlOut.setValue("x positions", new double[]{1,3,4});
xmlOut.setValue("x velocities", new double[]{0,-1,1});
```

Once the data has been stored in an `XMLControl` object, it can be exported to a file by calling the `write` method. In this example, the name of the file is `MDconfiguration.xml`.

```
xmlOut.write("MDconfiguration.xml");
```

An `XMLControl` can also be used to read XML data from a file. In the next example, we will read from the file that we just saved. We start by instantiating a new `XMLControl` named `xmlIn`.

```
XMLControl xmlIn = new XMLControlElement("particle_configuration.xml");
```

The new `XMLControl` object `xmlIn` contains the same data as the object we saved, `xmlOut`. Its data can be accessed using a tag name. Note that the `getObject` method returns a generic `Object` and must be cast to the appropriate data type.

```
System.out.println(xmlIn.getString("comment"));
double[] xPos = (double[])xmlIn.getObject("x positions");
double[] xVel = (double[])xmlIn.getObject("x velocities");
for(int i = 0; i < xPos.length; i++) {
   System.out.println("x[i] = " + xPos[i] +" vx[i] = " + xVel[i]);
}
```


Exercise 8.27 Saving XML data

(a) Combine the above statements to create a working `XMLControlApp` class. Examine the saved data using a text editor. Describe how the parameters are stored.

(b) Run `HardDisksApp` and save the control's configuration using the Save As item under the file menu in the toolbar. Examine the saved file using a text editor and describe how this file is different from the file you generated in part (a).

(c) What is the minimum amount of information that must be stored in a configuration file to specify the current `HardDisks` state?

(d) Add custom buttons to `HardDisksApp` to store and load the current `HardDisks` state. Test your code by showing that quantities, such as the temperature, remain the same if a configuration is stored and reloaded. ■

Open Source Physics user interfaces, such as a `SimulationControl`, store a program's configuration in two steps. During the first step, parameters from the graphical user interface are stored. During the second step, the model is given the opportunity to store runtime data using an `ObjectLoader`. Study the `LJParticlesLoader` class and note how storing and loading are done in the `saveObject` and `loadObject` methods, respectively. You will adapt this `ObjectLoader` to store `HardDisks` data in Problem 8.28. Additional information about how Open Source Physics applications store XML-based configurations is provided in *Open Source Physics: A User's Guide with Examples*.

Problem 8.28 Hard disk configuration

(a) Create a `HardDisksLoader` class that stores the `HardDisks` runtime data.

(b) Add the `getLoader` method to `HardDisksApp` and test the loader.

```
public static XML.ObjectLoader getLoader() {
   return new HardDisksLoader();
}
```

Method `XML.ObjectLoader getLoader` allows the `SimulationControl` to obtain the `HardDisksLoader`, which will be used to store the runtime data. Data written by the loader's `saveObject` method will be included in the output file when the user saves a program configuration. Describe how the initialization parameters and the runtime data are separated in the XML file. ■

Because XML allows for the creation of custom tags, various companies and professional organizations have defined other XML grammars, such as *MathML*. For another example of the use of XML in computational physics see `<http://xml.comp-phys.org/>`.

REFERENCES AND SUGGESTIONS FOR FURTHER READING

One of the best ways of testing your programs is by comparing your results with known results. The website `<http://www.cstl.nist.gov/lj>`, maintained by the National Institute of Standards and Technology (U.S.), provides some useful benchmark results for the Lennard–Jones fluid.