

The OpenMath Guide

A practical guide on using OpenMath

Manfred N. Riem

May 12, 2004

Contents

1	Introduction	5
2	An OpenMath Overview	9
2.1	Basic OpenMath objects	9
2.1.1	OpenMath ByteArray	10
2.1.2	OpenMath Float	10
2.1.3	OpenMath Integer	11
2.1.4	OpenMath String	11
2.1.5	OpenMath Symbol	11
2.1.6	OpenMath Variable	11
2.2	Constructors of OpenMath objects	11
2.2.1	OpenMath Application	12
2.2.2	OpenMath Attribution	12
2.2.3	OpenMath Binding	13
2.2.4	OpenMath Error	13
3	The Basics	15
3.1	Basic OpenMath objects	15
3.1.1	OpenMath ByteArray	16
3.1.2	OpenMath Float	17
3.1.3	OpenMath Integer	17
3.1.4	OpenMath String	18
3.1.5	OpenMath Symbol	19
3.1.6	OpenMath Variable	19
3.2	Compound OpenMath objects	20
3.2.1	OpenMath Application	20
3.2.2	OpenMath Attribution	21
3.2.3	OpenMath Binding	23
3.2.4	OpenMath Error	24

4	Reading and Writing	25
4.1	Reading and Writing using SAX	25
4.1.1	Reading an OpenMath object using a SAX source	26
4.1.2	Writing an OpenMath object to a String	26
4.1.3	Both reading and writing	27
4.2	Reading and Writing using DOM	27
5	Encoding and decoding	29
5.1	Design of the Codec Classes	29
5.1.1	The Codec class	30
5.1.2	The CDCCodec class	30
5.1.3	The CodecException classes	31
5.2	Some example implementations	31
5.2.1	Encoding	32
5.2.2	Decoding	34
5.2.3	Decoding by parsing	34
6	Using Phrasebooks	37
6.1	Design of the Phrasebook classes	37
6.1.1	The Phrasebook class	38
6.1.2	The PhrasebookException class	38
6.2	Some example implementations	38
6.2.1	An 'Echo' phrasebook	39
6.2.2	An 'Gap Socket' Phrasebook	40
7	Links	43
7.1	Design of the Link Class	43
7.1.1	The Link class	43
7.2	Some example implementations	45
7.2.1	Gap Link	45
7.2.2	Mathematica Link	49
7.2.3	Singular Link	50
8	Services	55
8.1	Design of the Service classes	55
8.1.1	The Service class	56
8.1.2	The Server class	57
8.1.3	The Socket Service class	59
8.2	An example implementation	59
8.2.1	A Gap Socket Service	59

9	Shells	63
9.1	Using GAP	63
9.1.1	Setting up	63
9.1.2	Querying GAP	66

Chapter 1

Introduction

‘What is the use of a book,’ thought Alice, ‘without pictures or conversations?’

–Lewis Carroll (1832-1898; Alice in Wonderland)

The RIACA OM library ROML is a piece of Java software for creating a programming environment in which a mathematical software package (referred to as a Mathematical Backengine or as Your Application) can be enabled to communicate with other software in OpenMath. Such a resulting programming environment will be called an OpenMath solution. In the course of this guide, we will discuss example OpenMath solutions using the Backengines GAP, Mathematica, and Singular.

This guide is meant for both application programmers and users of existing OpenMath solutions. For users, only Chapter 2, which explains OpenMath, and Chapter 9 on Shells and Services, are relevant. Application programmers are advised to read the whole guide, with a possible exception of the next chapter, most of whose notions are also introduced in the subsequent chapter.

The OpenMath library consists of the following parts.

- lang: handling OpenMath objects (e.g., creating Java objects representing OpenMath and accessing subobjects);
- io: reading XML strings and/or DOM trees and translating them into OpenMath objects, and vice versa;
- codec: translating OpenMath syntax to the syntax of Your Application, and vice versa;

- phrasebook: specifying the communication channel. The kind of communication depends on the phrasebook. Examples are sockets, and direct communication within one phrasebook (similar to JLink for Mathematica).

This document explains these four parts and their interaction. In Chapter 2, we discuss the Basic OpenMath objects. In Chapter 3, we explain their use and lang.

In Chapter 4, we discuss Reading and Writing of OpenMath objects, and explain the use of io.

In Chapter 5, we discuss Encoding and Decoding of OpenMath objects, and explain the use of codec.

In Chapter 6, we discuss the use and programming of Phrasebooks, and explain the code in phrasebook.

In Chapter 7, we discuss the link subproject. This helps to set up the connection with Your Application.

In Chapter 8, we discuss the service subproject. This helps to start Your Application as a service on the internet. That is, it will start up a shell that will enable to pass incoming commands to Your Applications, and to send out answers from Your Application to the internet.

Once you have read these parts of the guide, you can make a phrasebook for Your Application using ROML, as follows.

- Step 1 is to select a means for communication with Your Application. The choices are sockets, Jlink type, etc. See *** for examples.
- Step 2 is to select the Content Dictionaries you want to use. You can visit the most standard ones at the OpenMath Home page, or borrow from elsewhere, and/or create your own. Usually, the programs that Your Application should communicate with will be using Content Dictionaries that are imperative for Your Application.
- Step 3 is to copy a phrasebook implementation from the RIACA website which is as close to what you want to realize as possible. Alternatively take the example phrasebook from Appendix *****
- Step 3 is to adapt the example to Your Application and your own needs. In the codec you need to provide a translation for each symbol defined in each of the CDs that you intend to use. In the ideal case, you will be able to define the translations back and forth in one and the same files (one file for each CD). In cases where the output of Your Application is not uniformly parsable, you may need to write a separate parser and plug it into the Master Codec (see Chapter 3 for an explanation).

- Step 4 is to rename the package and class names in the example phrase-book in accordance with your set-up. Now you are ready to set up a service allowing Your Application to communicate in OpenMath.
- Step 5 is to copy the files in the link and service subproject to your own directories and ****

Usually, there is no need to adapt lang and io.

discuss variations of the rule **as for GAP in separate chapter on "Complications"

Chapter 2

An OpenMath Overview

This chapter explain the XML encoding of the OpenMath language. For more details see The OpenMath home site. The most important OpenMath notions are

- Content Dictionaries (CDs for short). These are documents defining OpenMath constructs and symbols.
- Phrasebooks. These are programs enabling communication with software applications in OpenMath speak. The symbols understood by the application are indicated by a list of CDs that the application is able to understand.
- The language. It is built up of expressions with six atoms and four constructors. The remainder of this chapter is devoted to these 10 key parts of the OpenMath language, encoded in XML.

The OpenMath Atoms.

The OpenMath Constructors.

2.1 Basic OpenMath objects

This section explains the various OpenMath atoms. The following list gives some short characterizations about the basic objects. For a more detailed explanation click the link.

ByteArray

A byte-array is an array of bytes which can be used in applications to store arbitrary data in a more native format.

Float

A float is a floating-point number either in decimal encoding or in a hexadecimal encoding.

Integer

An integer is an infinite precision integer.

String

A string is a Unicode string

Symbol

A symbol encodes two fields, a name and a Content Dictionary.

Variable

A variable consists of a name.

2.1.1 OpenMath ByteArray

An OpenMath ByteArray is the way OpenMath allows you to model arbitrary data using an array of bytes. This will make it possible to include non-mathematical data within an OpenMath object. E.g., a picture describing a given equation could be stored in a byte-array.

An example in XML encoding is:

```
<OMB>1010101101001</OMB>
```

2.1.2 OpenMath Float

An OpenMath Float models a IEEE floating point number. The OpenMath standard allows you to specify the given float by either a decimal encoding or a hexadecimal encoding. The floating point number is a double precision floating point number following the IEEE 754-1985 standard.

An example in XML encoding is:

```
<OMF dec="1.01213"/>
```

2.1.3 OpenMath Integer

An OpenMath Integer is an infinite precision integer and as such can be used to stored infinite huge numbers.

An example in XML encoding is:

```
<OMI>11203912</OMI>
```

2.1.4 OpenMath String

An OpenMath String is a Unicode character string. In the XML encoding of the OpenMath standard this corresponds to the 'characters' definition.

An example in XML encoding is:

```
<OMSTR>This is a string</OMSTR>
```

2.1.5 OpenMath Symbol

An OpenMath Symbol is used within OpenMath to be able to extend the 'language' with mathematical constructs. Eg. the plus operator is described by an OpenMath symbol.

```
<OMS cd="arith1" name="plus"/>
```

2.1.6 OpenMath Variable

An OpenMath variable is used to describe the notion of variables within a compound OpenMath object. Eg. a bound variable within a summation, or a unbound variable in function definition. Or just a variable in a specified context.

```
<OMV name="VarName"/>
```

2.2 Constructors of OpenMath objects

This section explains OpenMath constructors.

Application

An Application constructs an OpenMath object from a sequence of one or more OpenMath objects.

Attribution

An Attribution decorates an OpenMath object with a sequence of one or more pairs. The pairs are made up of an OpenMath symbol and an OpenMath object.

Binding

A Binding is constructed from an OpenMath object, followed by a sequence of zero or more variables followed by another OpenMath object.

Error

An Error is made up of an OpenMath symbol followed by a sequence of zero or more OpenMath objects.

2.2.1 OpenMath Application

The standard specifies an application as an object having a head and a tail. We will construct a OpenMath application that models $-10 + 10$. The object is given in XML encoding below.

```
<OMOBJ>
  <OMA>
    <OMS cd="arith1=" name="plus"/>
    <OMI>-10</OMI>
    <OMI>10</OMI>
  </OMA>
</OMOBJ>
```

2.2.2 OpenMath Attribution

Attribution allows you to attach attribution pairs to an OpenMath object.

An example in XML encoding is:

```

<OMOBJ>
  <OMATTR>
    <OMATP>
      <OMS cd="type" name="enumerated"/>
      <OMS cd="set1" name="N"/>
    </OMATP>
    <OMI>10</OMI>
  </OMATTR>
</OMOBJ>

```

2.2.3 OpenMath Binding

A Binding has a binder, variables and a body.

An example in XML encoding is:

```

<OMOBJ>
  <OMBIND>
    <OMS cd="fns1" name="lambda"/>
    <OMBVAR>
      <OMV name="x"/>
    </OMBVAR>
    <OMV name="x"/>
  </OMBIND>
</OMOBJ>

```

2.2.4 OpenMath Error

An Error is constructed from a symbol and zero or more OpenMath objects.

An example in XML encoding is:

```

<OMOBJ>
  <OME>
    <OMS cd="error" name="runtime"/>
    <OMSTR>general error 10</OMSTR>
  </OME>
</OMOBJ>

```

</OME>
</OMOBJ>

Chapter 3

The Basics

Long is the road learned by lessons, short and helpful by use of examples.

–Seneca

The lessons in this chapter show you how to use the OpenMath library in its basic form. It explains some of the design decisions made so far and what impact it has on how to use this library.

Using OpenMath Atoms shows you how to interact with the most basic set of objects that are defined in the OpenMath standard.

Using OpenMath Composite Objects shows you how to interact with the composite objects that can be constructed as defined by the OpenMath standard.

3.1 Basic OpenMath objects

This section explains the OpenMath Atoms and their usage in Mathematics and in code. The following list gives some short characterizations about the basic objects.

ByteArray

A byte-array is an array of bytes which can be used in applications to store arbitrary data in a more native format.

Float

A float is a floating-point number either in decimal encoding or in a hexadecimal encoding.

Integer

An integer is an infinite precision negative or positive number.

String

A string is a Unicode string

Symbol

A symbol encodes two fields, a name and a Content Dictionary.

Variable

A variable consists of a name.

3.1.1 OpenMath ByteArray

An OpenMath ByteArray is the way OpenMath allows you to model arbitrary data using an array of bytes. This will make it possible to include non-mathematical data within an OpenMath object. E.g., a picture describing a given equation could be stored in a byte-array.

The following code snippet shows you how to code using a byte-array. One can set and get the byte array data which is stored as an array of bytes internally in the object by using the appropriate methods.

```
import nl.tue.win.riaca.openmath.lang.OMByteArray;

OMByteArray obyteArray = new OMByteArray();
byte[]      obytes      = ....

/*
 * Set the byte-array using a JVM array of bytes.
 */

obyteArray.setByteArray( bytes );

/*
 * Set the byte-array using a String.
 */

obyteArray.setByteArray( "1234" );
```

You can get the value of the `OMByteArray` by storing it in a JVM array of bytes or in a `String`.

```
// continuing from above.  
  
byte[] bytes = oByteArray.getBytes();
```

Look at the API for the complete description of an `OMByteArray`

3.1.2 OpenMath Float

An `OpenMath Float` models a IEEE floating point number. The `OpenMath` standard allows you to specify the given float by either a decimal encoding or a hexadecimal encoding. The floating point number is a double precision floating point number following the IEEE 754-1985 standard.

The code snippet below uses the decimal encoding to set the value of the floating point number.

```
import nl.tue.win.riaca.openmath.lang.OMFloat;  
  
OMFloat ofloat = new OMFloat();  
ofloat.setFloat( "1.234", "dec" );
```

You can get the value of the `OMFloat` by storing it in a `float` or in a `double` as defined by the Java™ Virtual Machine specification.

```
// continuing from above.  
  
float float = ofloat.floatValue();  
double double = ofloat.doubleValue();
```

Look at the API for the complete description of an `OMFloat`

3.1.3 OpenMath Integer

An `OpenMath Integer` is an infinite precision integer and as such can be used to stored infinite huge numbers.

The code snippet below shows you how you can get a basic JVM typed integer from the `OpenMath Integer`. Note that the exception handling code is omitted.

```
import nl.tue.win.riaca.openmath.lang.OMInteger;

OMInteger ointeger = new OMInteger();
ointeger.setInteger( "1234" );
```

You can get the value of the OMInteger by storing it in a long or in an integer as defined by the Java™ Virtual Machine specification.

```
// continuing from above.

long nlong    = ointeger.longValue();
int  ninteger = ointeger.intValue();
```

You can also give the OMInteger an Integer as its parameter to the constructor, see the API for more details about that.

```
import nl.tue.win.riaca.openmath.lang.OMInteger;

OMInteger ointeger = new OMInteger( new Integer( 2 ) );
```

Look at the API for the complete description of an OMInteger

3.1.4 OpenMath String

An OpenMath String is a Unicode character string. In the XML encoding of the OpenMath standard this corresponds to the 'characters' definition.

The code snippet shows you how to instantiate an OMString and assigns a value of "Hello World!" to it.

```
import nl.tue.win.riaca.openmath.lang.OMString;

OMString ostring = new OMString();
ostring.setString( "Hello World!" );
```

You can get the value of the OMString by storing it in a Java String.

```
// continuing from above.

String string = ostring.getString();
```

Look at the API for the complete description of an OMString

3.1.5 OpenMath Symbol

An OpenMath Symbol is used within OpenMath to be able to extend the 'language' with mathematical constructs. Eg. the number Pi is described by an OpenMath symbol.

The following code snippet shows you how to construct a symbol. In this case it is the symbol 'arith.plus' which is used to model the arithmetic plus.

```
import nl.tue.win.riaca.openmath.lang.OMSymbol;

OMSymbol symbol = new OMSymbol();
symbol.setCD( "arith1" );
symbol.setName( "plus" );
```

You can get the CD of the OMSymbol and the name of the OMSymbol by storing each of them in a Java String.

```
// continuing from above.

String cd    = symbol.getCD();
String name = symbol.getName();
```

Look at the API for the complete description of an OMSymbol

3.1.6 OpenMath Variable

An OpenMath variable is used to describe the notion of variables within a compound OpenMath object. Eg. a bound variable within a summation, or a unbound variable in function definition.

The example block of code instantiates an OMVariable and assigns the name of "x" to it.

```
import nl.tue.win.riaca.openmath.lang.OMVariable;

OMVariable variable = new OMVariable();
variable.setName( "x" );
```

You can get the value of the OMVariable by storing it in a String.

```
// continuing from above.  
  
String string = variable.getName();
```

Look at the API for the complete description of an OMVariable

3.2 Compound OpenMath objects

This section explains OpenMath compound objects and how you can use them in your code.

Application

An Application constructs an OpenMath object from a sequence of one or more OpenMath objects.

Attribution

An Attribution decorates an OpenMath object with a sequence of one or more pairs. The pairs are made up of an OpenMath symbol and an OpenMath object.

Binding

A Binding is constructed from an OpenMath object, followed by a sequence of zero or more variables followed by another OpenMath object.

Error

An Error is made up of an OpenMath symbol followed by a sequence of zero or more OpenMath objects.

3.2.1 OpenMath Application

The standard specifies an application having a head and a tail. Our implementation abstracts from this by using a generic framework by looking at it as a Vector that has zero or more elements.

We will construct a OpenMath application that models $-10 + 10$. The first listing will be the OpenMath object encoding in XML syntax.

```

<OMOBJ>
  <OMA>
    <OMS cd="arith1" name="plus"/>
    <OMI>-10</OMI>
    <OMI>10</OMI>
  </OMA>
</OMOBJ>

```

The following listing constructs the object described before.

```

OMApplication application = new OMApplication();

application.addElement( new OMSymbol( "arith1", "plus" ) );
application.addElement( new OMInteger( "-10" ) );
application.addElement( new OMInteger( "10" ) );

/*
 * Gets the head (which in this case is the OMSymbol,
 * but it could be any OMObject).
 */

OMObject head = application.firstElement();

/*
 * Gets the first element from the tail.
 */

OMObject element = application.getElementAt( 1 );

```

Note: the exception handling code has been omitted from the example. Please see the API for more details!

Look at the API for the complete description of an OMApplication

3.2.2 OpenMath Attribution

According to the standard an attribution has a constructor and attribute pairs. Note that the attribute pairs are internally stored in a Hashtable structure. With the current version of the library you can either inspect the constructor or the entire hashtable of attributes.

First we will describe the OpenMath object we are going to construct in code using the XML syntax.

```
<OMOBJ>
  <OMATTR>
    <OMATP>
      <OMS> cd="type" name="enumerated"/>
      <OMS> cd="set1" name="N"/>
    </OMATP>
    <OMI>10</OMI>
  </OMATTR>
</OMOBJ>
```

The following code snippet constructs the previously mentioned OpenMath object.

```
OMAttribution attribution = new OMAttribution();
OMSymbol      key          = new OMSymbol( "type", "enumerated" );
OMSymbol      value        = new OMSymbol( "set1", "N" );

tOMAttribution.setConstructor( new OMInteger( "10" ) );
tOMAttribution.put( key, value );

/*
 * Gets the constructor (which is the OMInteger).
 */

OMObject constructor = attribution.getConstructor();

/*
 * Gets the hashtable of attributions (which contains the attribution).
 */

Hashtable attributions = attribution.getAttributions();
```

Look at the API for the complete description of an OMAttribution

3.2.3 OpenMath Binding

A Binding has a binder, variables and a body. With the current version of the library you can either inspect the binder, the body or the entire vector of variables.

The following XML snippet defines the OpenMath object we are going to create later on using code.

```
<OMOBJ>
  <OMBIND>
    <OMS cd="fns1" name="lambda"/>
    <OMBVAR>
      <OMV name="x"/>
    </OMBVAR>
    <OMV name="x"/>
  </OMBIND>
</OMOBJ>
```

```
OMBinding binding = new OMBinding();
Vector variables = new Vector();

variables.addElement( new OMVariable( "x" ) );

binding.setBinder( new OMSymbol( "fns1", "lambda" ) );
binding.setBody( new OMVariable( "x" ) );
binding.setVariables( variables );

/*
 * Gets the binder (which is the OMSymbol).
 */

OMObject binder = tOMBinding.getBinder();

/*
 * Gets the vector of variables.
 */

Vector vars = tOMBinding.getVariables();

/*
 * Gets the body.
```

```

    */

    OMObject body = tOMBinding.getBody();

```

Look at the API for the complete description of an OMBinding

3.2.4 OpenMath Error

An Error is constructed from a symbol and zero or more OpenMath objects.

The following snippet describe a user created OpenMath error with a runtime error and associated string.

```

<OMOBJ>
  <OME>
    <OMS cd="error" name="runtime"/>
    <OMSTR>general error: 10</OMSTR>
  </OME>
</OMOBJ>

OMError error = new OMError();

error.addElement( new OMSymbol( "error", "runtime" ) );
error.addElement( new OMString( "general error: 10" ) );

/*
 * Gets the symbol.
 */

OMObject symbol = tOMError.firstElement();

/*
 * Gets the string.
 */

OMObject string = tOMError.getElementAt( 1 );

```

Note: the exception handling code has been omitted from the example. Please see the API for more details!

Look at the API for the complete description of an OMError

Chapter 4

Reading and Writing

The lessons in this chapter show you how to use the library to read and write OpenMath objects. The current version allows you to read them using SAX and DOM. Future versions of the library might extend this to include more formats.

Reading and Writing using SAX shows you how to use the library to read in an OpenMath object which is encoded using XML, and how to write an OpenMath object to XML.

Reading and Writing using DOM shows you how to use the library to read in an OpenMath object from a DOM object, and how to write an OpenMath object to a DOM object.

4.1 Reading and Writing using SAX

This section explains how to read an OMOBJECT from a SAX source, and how to write out an OMOBJECT.

Reading an OpenMath object

This explains you how to use the library to read in an OpenMath object from an SAX source.

Writing out an OpenMath object

This explains you how to use the library to write out an OpenMath object to a String.

Both reading and writing

Both reading and writing in one go. Just a short example that shows you that the library is capable of reading in the given OpenMath object and writing it out also.

4.1.1 Reading an OpenMath object using a SAX source

The code snippet below shows you a way you could read an OpenMath object using a SAX source. See for more information about SAX your Java2 documentation. (Note: the exception handling code is omitted)

```
import nl.tue.win.riaca.openmath.io.*;
import org.xml.sax.*;

InputStream input = ....
OMXMLReader reader =
    new OMXMLReader( new InputSource( inputStream ) );

OMObject object = reader.readObject();
```

Look at the API for the complete description of an OMXMLReader

4.1.2 Writing an OpenMath object to a String

The following code snippet shows you how to write an OpenMath object to a Java String object. The Java String object will contain the XML encoded OpenMath object. (Note: the exception handling code is omitted)

```
import nl.tue.win.riaca.openmath.io.*;

OMObject object = ....
OutputStreamWriter writer = new OutputStreamWriter( System.out );
OMXMLWriter xmlWriter = new OMXMLWriter( writer );

xmlWriter.writeObject( object );
xmlWriter.flush();
```

Look at the API for the complete description of an OMXMLWriter

4.1.3 Both reading and writing

For convenience the code necessary for reading and writing using SAX is presented below (Note: the exception handling code is omitted)

```
import nl.tue.win.riaca.openmath.io.*;
import org.xml.sax.*;

/*
 * Reading
 */

InputStream input = ....
OMXMLReader reader = new OMXMLReader( new InputSource( input ) );
OMObject object = tReader.readObject();

/*
 * Writing
 */

OutputStreamWriter writer = new OutputStreamWriter( System.out );
OMXMLWriter xmlWriter = new OMXMLWriter( writer );

xmlWriter.writeObject( object );
xmlWriter.flush();
```

Look at the API for the complete description of an OMXMLWriter

4.2 Reading and Writing using DOM

This section explains how to read an OpenMath object from a DOM object, and how to write out an OpenMath object to a DOM object.

Reading an OpenMath object

This explains you how to use the library to read in an OpenMath object from a DOM object.

Writing out an OpenMath object

This explains you how to use the library to write an OpenMath object out to a DOM object.

Both reading and writing

Both reading and writing in one go. Just a short example that shows you that the library is capable of reading in the OpenMath object and writing it out also.

Chapter 5

Encoding and decoding

The lessons in this chapter show you how to use the library to encode and decode OpenMath objects. Note: that this only explains the design of the Encode and Decode classes. The library itself does NOT contain any implementations.

Design of the Codec classes explains the design of the Codec classes and how an implementer should use them to make Codecs that are easy to use.

A Codec example: OMLisp Codec tells you how to implement a Codec by showing how it is done for OpenMath to LISP-notation and vice-versa.

5.1 Design of the Codec Classes

The Codec class

This class is an abstract class that specifies the basic set of methods a Codec class needs to implement. Note that this is all a Codec needs to implement. Note that most codecs will use the Codec/CDCCodec structure described later.

The CD Codec class

This class is a helper class that will handle the encoding and decoding for a particular OpenMath Content Dictionary. Hence the name CDCCodec (where CD is Content Dictionary).

The Codec Exception classes

These set of classes describe the encoding or decoding errors that can occur while encoding and decoding is in progress. Note that all the Exceptions thrown in CDCCodecs and/or the master Codec need to be wrapped by a CodecException of some sort.

5.1.1 The Codec class

Every Codec needs to deliver an implementation of this class. It is used to encode from OpenMath to a specific syntax, and vice-versa.

For the encoding the class has 2 methods:

- encode
- encodeOMObject

The last one is the one that usually contains the real encoding, while the first one is probably the one that is used the most as it accepts a XML string encoded OpenMath object. These methods are to be subclassed by an implementation of the Codec class.

For the decoding the class also has 2 methods:

- decode
- decodeOMObject

The last one is the one that usually contains the real decoding, while the first one is probably the one that is used the most as it returns a XML string encoded OpenMath object. These methods are to be subclassed by an implementation of the Codec class. In most cases the real decoding is done by a specialized parser class that is used as a blackbox by the decodeOMObject method.

Look at the API for the complete description of a Codec

5.1.2 The CDCCodec class

Because a CDCCodec class is a specialized subclass of the Codec class it has the same encode and decode methods. However this class has 2 additional extra methods called:

- setParent
- getParent

The first sets the parent of this CDCCodec (generally the master Codec), while the last returns the parent previously set.

Look at the API for the complete description of a CDCCodec

5.1.3 The CodecException classes

While encoding and decoding exceptions can occur. To make codec implementations transparent to the codec user (generally the phrasebook) the following exception classes are available.

- CodecEncodingException class
- CodecDecodeException class
- CodecException class

The first one is used when an error occurs in the encoding stage, the second is used when an error occurs in the decoding stage. And the third one can be used when the needed resources for the Codec are unavailable (eg. if the decoding is done by using a parser and the parser cannot be found).

5.2 Some example implementations

This section gives you some example implementations with respect to encoding and decoding of OpenMath object to 'Your application' syntax. The example consist of code of GAP and Mathematica.

Encoding example

This example describes how you set up encoding in the Codec/CDCCodec structure we have used for our implementations.

Decoding example.

This example describes the decoding step in the Codec class. Note that this example assumes that your favorite application has a tree structured syntax. Because this is not generally the case you might be better of with the next example

Decoding by parsing

This example describes how you can plugin a parser and at which level you will need to do that, so you can parse back to OpenMath. Note that the example features a GAP back parsing parser.

5.2.1 Encoding

The following code examples give you an idea how easy it is to write your own Codec. It is a 2 step process. Step 1 is to write the master Codec. Step 2 is to write a CDCCodec for each CD you want to support.

This code snippet below, which is of a master Codec shows that the encoding method called depends on the type of the object passed into the main encodeOMObject method.

Some methods have a simple encoding (like OMInteger below) and thus are handled at the master Codec level, while others need CDCCodecs to resolve the encoding (like OMAApplication below).

```
...

/**
 * Encodes the OMInteger to Gap. <p>
 *
 * @since 1.0
 */

public String encodeOMInteger( OMObject object )
    throws CodecEncodeException {
    // System.out.println( "GapCodec.encodeOMInteger " );

    if ( object instanceof OMInteger ) {
        OMInteger integer = (OMInteger) object;

        return tOMInteger.getInteger();
    }

    throw new CodecEncodeException( "Unable to encode: " + object );
}

...

/**
 * Encodes the OMAApplication to a Gap-command. <p>
 *
 * @since 1.0
 */
```

```

public String encodeOMApplication( OMOject object )
    throws CodecEncodeException {
    // System.out.println( "GapCodec.encodeOMApplication" );

    if ( object instanceof OMAApplication ) {
        OMAApplication application = (OMAApplication) object;
        OMOject      tobject      = application.getElementAt( 0 );

        if ( tobject instanceof OMSymbol ) {
            OMSymbol symbol = (OMSymbol) tobject;
            String className      = "";
            String classLocation = "";

            if ( mCDs.get( symbol.getCD() ) != null ) {
                String tLocationClass = (String)
                    mCDs.get( symbol.getCD() );

                className = tLocationClass.substring(
                    tLocationClass.lastIndexOf( ":" ) + 1 );

                classLocation = tLocationClass.substring(
                    0, tLocationClass.lastIndexOf( ":" ) );
            }
            else
                className = "nl.tue.win.riaca.gap.codec.core." +
                    symbol.getCD() + "Codec";

            try {
                if ( classLocation.equals( "" ) ) {
                    Class clazz = Class.forName( className );
                    Object tObject = clazz.newInstance();
                    CDCCodec tCDCCodec = (CDCCodec) tObject;

                    tCDCCodec.setParent( this );
                    return tCDCCodec.encodeOMOject( application );
                }
                else {
                    URLClassLoader classLoader = new URLClassLoader(
                        new URL[] { new URL( classLocation ) } ,
                        this.getClass().getClassLoader() );
                }
            }
        }
    }
}

```

```

        Class clazz = tClassLoader.loadClass( className );
        Object tObject = clazz.newInstance();
        CDCCodec tCDCCodec = (CDCCodec) tObject;

        tCDCCodec.setParent( this );
        return tCDCCodec.encodeOMObject( application );
    }
}
catch( MalformedURLException tMalformedURLException ) {
    throw new CodecEncodeException(
        "Unable to parse location URL codec class: " +
        classLocation );
}

... more exception handling ...
}
else if ( tobject instanceof OMVariable ) {
    ... handling variable applied on a application ..
}
}

throw new CodecEncodeException( "Unable to encode: " + object );
}

```

The code above uses Java URL classloading to use the new Codecs that have been added by a Codec programmer. Note that you can add and/or remove Codecs at runtime.

So far we have only covered Step 1. To make the setup really work you will have to write a CDCCodec. The following example shows part of a possible implementation for the arith1 CD. See the appendix for the entire class.

For more information about codecs, see the Codec API.

5.2.2 Decoding

Describe how tree structured decoding works.

5.2.3 Decoding by parsing

Generally the decoding of the result of your favorite application cannot be done easily and the use of a parser is needed. The structure of the Codec

has been setup in such a way that you can easily plugin your own parser at the appropriate place. The code below shows you how to do it.

As before the master code (in this case GapCodec) has a decodeOMObject method. This is the method where you need to bootstrap your parser and call it.

```
... imports ...

public class GapCodec extends Codec {

    ...

    /**
     * Decodes the Gap-result to an OM-object. <p>
     */

    public OMObject decodeOMObject( String syntax )
    throws CodecDecodeException {
        // System.out.println( "GapCodec.decodeOMObject" );

        try {
            GapOMLexer lexer = new GapOMLexer(
                new StringReader( syntax ) );

            GapOMParser parser = new GapOMParser( lexer );
            parser.expr();

            CommonAST      ast      = (CommonAST) parser.getAST();
            GapOMTreeWalker walker = new GapOMTreeWalker();
            String          result = walker.expr( ast );

            StringReader stringReader =
                new StringReader( result );

            InputSource source = new InputSource( stringReader );
            OMXMLReader reader = new OMXMLReader( source );
            OMObject object  = reader.readObject();

            return object;
        }
        catch( Exception exception ) {
```

```
        throw new CodecDecodeException(
            "Unable to parse: " + result + " due to " +
            exception.getClass() + ": " +
            exception.getMessage() );
    }
}
```

Note that the body of this method is specific to our implementation of the parser (we are using ANTLR to do the parsing).

Chapter 6

Using Phrasebooks

The lessons in this chapter show you how to use a phrasebook to interact with another program. Note: that this only explains the design of the Phrasebook class. The OpenMath library itself does NOT contain any implementations.

Design of the Phrasebook class explains the design of the Phrasebook class and how an implementer should extend it to make Phrasebooks that are easy to use.

Some example implementations gives you 2 examples that show you how you write your own phrasebook.

6.1 Design of the Phrasebook classes

This section describes the design of the phrasebook. Note that these classes are not to be modified by a programmer that wishes to implement his own phrasebook. See the next section for 2 examples that implement custom phrasebooks..

The Phrasebook class

This class is an abstract class specifying what an implementation of a phrasebook needs to have to properly function.

The PhrasebookException class

This class is used to wrap all the exceptions a phrasebook can throw while using it. Each method that implements a method defined in the abstract class should throw a PhrasebookException when a major problem occurs.

6.1.1 The Phrasebook class

A Phrasebook needs to deliver an implementation of this class. It is used to communicate OpenMath and/or the native syntax to the application and back. Note that some applications will support several methods of communicating, while others will only support a specific subset.

The phrasebook has 4 basic methods:

- perform
- execute
- addCD
- removeCD

The first two methods are used for performing or executing a query, while the last two methods are used for configuring the phrasebook. A phrasebook should be capable of handling new CDs and as such adding and removing CDs is added to this level. Most of the implementations of a phrasebook will delegate this responsibility to the internal Codec class. Note however that this is not enforced.

Look at the API for the complete description of a Phrasebook

6.1.2 The PhrasebookException class

While using a phrasebook an exception can experience an error it cannot recover from. An implementation of the phrasebook should wrap the Exception and/or Throwable inside a PhrasebookException.

Look at the API for the complete description of a PhrasebookException

6.2 Some example implementations

To implement a phrasebook one needs to implement all the abstract methods in the base class, and extend other methods in the base class. In general it is enough to program your own implementation of either the perform method or the execute method and also for the addCD and removeCD methods.

An 'Echo' Phrasebook

This example implements a phrasebook that echos back to the user what the user had entered as input.

An 'Gap Socket' Phrasebook

This example implements a phrasebook that calls GAP that is running on a socket. Note that this example uses a specific kind of Service which is describe in a later chapter.

6.2.1 An 'Echo' phrasebook

For a phrasebook that just echos the input, a possible implementation is:

```
import nl.tue.win.riaca.openmath.phrasebook.*;
import nl.tue.win.riaca.openmath.lang.*;

public class EchoPhrasebook extends Phrasebook {

    public String perform( String method, Vector arguments ) {
        if ( method.equalsIgnoreCase( "EVAL" ) ) {
            if ( arguments.elementAt( 0 ) instanceof String )
                return (String) arguments.elementAt( 0 );

            throw new PhrasebookException( "Incorrect input type" );
        }
    }

    public abstract void addCD( String name,
                               String location,
                               String className )
        throws PhrasebookException {
    }

    public abstract void removeCD( String name )
        throws PhrasebookException {
    }
}
```

Note that this phrasebook does NOT use any codec. Generally a phrasebook will use a codec to do the translation from the one syntax to the other.

6.2.2 An 'Gap Socket' Phrasebook

The code below is a part of the phrasebook that makes up the entire GapSocketPhrasebook class. The entire code is for sake of completeness added as an appendix to the book.

```

... imports ...

public class GapSocketPhrasebook
    extends Phrasebook
{
    /**
     * Stores the codec.
     *
     * @since 1.0
     */

    protected GapCodec codec = null;

    ...

    /**
     * Perform a command and get the result. <p>
     *
     * @since 1.0
     */

    public String perform( String method, Vector arguments )
        throws PhrasebookException
    {
        // System.out.println( "GapSocketPhrasebook.perform" );

        if ( method.equalsIgnoreCase( "EVAL" ) )
            return performEval( arguments );

        ....
    }

    /**
     * Send OpenMath and get back OpenMath. <p>
     *
     * @since 1.0

```

```
    */

protected String performEval( Vector arguments )
    throws PhrasebookException
{
    // System.out.println( "GapSocketPhrasebook.performEval" );

    Object element0 = arguments.elementAt( 0 );
    String command = "";

    try {
        command = codec.encode( (String) element0 );
    }
    catch( CodecEncodingException codecEncodingException ) {
        throw new PhrasebookException(
            codecEncodingException.getClass() + ":" +
            codecEncodingException.getMessage() );
    }

    String returnString = performCall( command );

    if ( returnString.endsWith( ";" ) == false )
        returnString = returnString + ";";

    String decode;

    try {
        decode = codec.decode( returnString );
    }
    catch( CodecDecodeException decodeException )
    {
        throw new PhrasebookException(
            decodeException.getClass() + ":" +
            decodeException.getMessage() );
    }

    return decode;
}
}
```

Note that the code here uses a codec to do the translation from OpenMath syntax to Gap syntax and vice versa. Also note that we have implemented another method that does the actual call. For sake of brevity this was omitted, see the appendix for more details.

Chapter 7

Links

The lessons in this chapter show you how to implement Links. These Links are small utility programs that allow you to 'link' a legacy (non-java) program to Java. Note: while this is not really a part of OpenMath it is useful to have a generic structure on which a programmer can build to create more links to legacy programs.

Design of the Link class explains the design of the Link class and how an implementer could program it to make a Link to a legacy program.

Some example implementations gives you 3 examples that show you how we have written links to some legacy programs.

7.1 Design of the Link Class

The Link class

The Link class is not implemented as a part of the OpenMath library. It is a separate utility class which is integrated into several of the Services (those services are described in a later chapter). Note: a future version of the OpenMath library will contain an abstract implementation of the Link class.

7.1.1 The Link class

Every Link needs to implement 3 methods.

- start, which starts the legacy process so we can start issuing requests to it. Note that some programs take a while to initialize and some programs emit additional information upon startup.

- stop, which stops the legacy process. A stopped link will not be able to execute requests.
- execute, which executes a request by sending the input String to the legacy program and capturing the output in an output String and returning that as the result.

The following code snippet can be used as a template to implement a Link.

```
public class MyLink {

    /**
     * Starts the link. <p>
     *
     * @since 1.0
     */

    public synchronized boolean start( String parameters ) {
        // System.out.println( "Link.start" );

        return true;
    }

    /**
     * Stops the link. <p>
     *
     * @since 1.0
     */

    public synchronized boolean stop() {
        // System.out.println( "Link.stop" );

        return true;
    }

    /**
     * Executes the particular command and returns the result. <p>
     *
     * @since 1.0
     */
}
```

```
public synchronized String execute( String command ) {
    // System.out.println( "Link.execute" );

    return null;
}
}
```

7.2 Some example implementations

This section gives you some example implementations of Links to legacy programs.

Linking to GAP

This example describes the implementation of the Link to GAP.

Linking to Mathematica

This example describes the implementation of the Link to Mathematica.

Linking to Singular

This example describes the implementation of the Link to Singular.

7.2.1 Gap Link

The following code snippet contains the most important parts of the GAP Link. Note that parts of the methods are abbreviated and that the clean method has been omitted all for sake of brevity. See the appendix for more details.

```
... imports ...

public class GapLink {

    ... class variables ...

    /**
     * @since 1.0
     */
```

```
public synchronized boolean start( String parameters ) {
    // System.out.println( "GapLink.start" );

    if ( process == null ) {
        try {
            /*
             * Start gap with the given command.
             */

            process = Runtime.getRuntime().exec( parameters );

            /*
             * We need the process output-stream to write the request to
             * the process, and the process input-stream to read the
             * response.
             */

            processInput = process.getOutputStream();
            processOutput = process.getInputStream();

            .... removed reading of startup message ...

        }
        catch( IOException exception ) {
            exception.printStackTrace();
            return false;
        }
    }

    return true;
}

/**
 * @since 1.0
 */

public synchronized boolean stop() {
    // System.out.println( "GapLink.stop" );

    if ( process != null ) {
```

```
        execute( "quit;" );
        process.destroy();
        process = null;
    }

    return true;
}

/**
 * @since 1.0
 */

public synchronized String execute( String fCommand ) {
    // System.out.println( "GapLink.execute" );

    if ( process != null ) {
        try {
            /*
             * Write the command.
             */

            PrintStream tPrintStream = new PrintStream( processInput );

            if ( !fCommand.endsWith( ";" ) )
                fCommand = fCommand + ";";

            tPrintStream.println( fCommand );
            tPrintStream.flush();

            /*
             * Reads the response.
             */

            StringBuffer tStringBuffer = new StringBuffer();

            char tPrevious = '\0';
            char tChar      = (char) processOutput.read();

            while( true ) {
                if ( ( tPrevious == '@' && tChar == 'i' ) ||
                    ( tPrevious == '@' && tChar == 'e' ) )
```

```

        break;

tStringBuffer.append( tChar );

tChar = (char) processOutput.read();
tPrevious = tStringBuffer.charAt(
    tStringBuffer.length() - 1 );
}

tStringBuffer.append( tChar );

/*
 * Break out of break state if we are in it.
 */

if ( tStringBuffer.toString().indexOf( "brk>" ) != -1 ) {
    tPrintStream.println( "quit;" );
    tPrintStream.flush();

    StringBuffer tBreakBuffer = new StringBuffer();

    tPrevious = '\0';
    tChar      = (char) processOutput.read();

    while( true ) {
        if ( ( tPrevious == '@' && tChar == 'i' ) ||
            ( tPrevious == '@' && tChar == 'e' ) )
            break;

        tBreakBuffer.append( tChar );

        tChar = (char) processOutput.read();
        tPrevious = tBreakBuffer.charAt(
            tBreakBuffer.length() - 1 );
    }
}

String tResult = tStringBuffer.toString();
tResult = tResult.substring( tResult.indexOf( '\n' ) + 1 );
tResult = clean( tResult );
tResult = tResult.trim();

```

```

        return tResult;
    }
    catch( IOException tException ) {
        tException.printStackTrace();
    }
}

return null;
}

..... snipped out clean method .....
}

```

Note that the Link will return null on an invocation of execute when the connection was not previously established. In this implementation we also handle a specific application level error, by issuing the correct command to reset the legacy program to the correct state (break out of break state).

7.2.2 Mathematica Link

The following code snippet contains the most important parts of the Mathematica Link. Note that parts of the class have been abbreviated for sake of brevity. See the appendix for more details.

```

... imports ...

/**
 * @since 1.0
 */

public synchronized boolean start( String fParameters ) {
    // System.out.println( "MathematicaLink.start" );

    try {
        mKernelLink = MathLinkFactory.createKernelLink( fParameters );
        mKernelLink.discardAnswer();
        mKernelLink.newPacket();
    }
    catch( MathLinkException tMathLinkException ) {

```

```

        tMathLinkException.printStackTrace();
        return false;
    }

    return true;
}

/**
 * @since 1.0
 */

public synchronized boolean stop() {
    // System.out.println( "MathematicaLink.stop" );

    return true;
}

/**
 * @since 1.0
 */

public synchronized String execute( String fCommand ) {
    // System.out.println( "MathematicaLink.execute" );

    return mKernelLink.evaluateToOutputForm( fCommand, 0 );
}
}

```

The implementation of this Link is a lot shorter because Mathematica delivers J/Link as a way to link their product to Java. This class basically wraps a simple Link around their implementation. See the appendix for more details.

7.2.3 Singular Link

The following code snippet contains the most important parts of the Singular Link. Note that parts of the class have been abbreviated for sake of brevity. See the appendix for more details.

```
... imports ...
```

```
public class SingularLink {

    ... class variables ...

    /**
     * @since 1.0
     */

    public synchronized boolean start( String fParameters ) {
        // System.out.println( "SingularLink.start" );

        if ( mProcess == null ) {
            try {
                /*
                 * Start Singular with the given command.
                 */

                mProcess = Runtime.getRuntime().exec( fParameters );

                /*
                 * We need the process output-stream to write the request to
                 * the process, and the process input-stream to read the
                 * response.
                 */

                mProcessInput = mProcess.getOutputStream();
                mProcessOutput = mProcess.getInputStream();

                /*
                 * Wait till started, and read out some junk.
                 */

                execute(";");
            }
            catch( IOException tException ) {
                tException.printStackTrace();
                return false;
            }
        }
    }
}
```

```
    return true;
}

... removed stop, same implementation as Gap ...

/**
 * @since 1.0
 */

public synchronized String execute( String fCommand ) {
    // System.out.println( "SingularLink.execute: '" + fCommand + "'");

    if ( mProcess != null ) {
        try {
            String cNastyTrick = "singular-link: end of output.";
            String cNastyTrickCmd = "print(\"" + cNastyTrick + "\");";
            String cNastyTrickCheck =
                cNastyTrick + (new String(new byte[] {10}));

            /*
             * Write the command.
             */

            PrintStream tPrintStream = new PrintStream( mProcessInput );

            if ( !fCommand.endsWith( ";" ) )
                fCommand = fCommand + ";";

            fCommand = fCommand + cNastyTrickCmd;

            tPrintStream.println( fCommand );
            tPrintStream.flush();

            /*
             * Reads the response.
             */

            StringBuffer tStringBuffer = new StringBuffer();
            char tChar;

            while( true ) {
```

```
tChar = (char) mProcessOutput.read();
tStringBuffer.append (tChar);

if ( ( (int) tChar == 10 ) &&
    ( tStringBuffer.length() >= cNastyTrickCheck.length() ) ) {
    if ( tStringBuffer.substring( tStringBuffer.length() -
        cNastyTrickCheck.length()).equals( cNastyTrickCheck ) ) {
        tStringBuffer.setLength( tStringBuffer.length() -
            cNastyTrickCheck.length());
        break;
    }
}

if ( (int) tChar > 255 ) {
    break;
}

String tResult = tStringBuffer.toString().trim();
return tResult;
}
catch( IOException tException ) {
    tException.printStackTrace();
}
}

return null;
}
}
```

Note that this Link implementation uses a nasty trick (according to the author) to check the end of the output of the legacy program. Such implementations will be frequent as most legacy programs do not have a clear interface on which one can build.

Chapter 8

Services

This section shows you how to implement Services. A Service is a program that wraps around a Link to make it possible to deliver a TCP/IP based link to a legacy program. At least that is our current implementation. One could implement a service using RMI; this shows you that the Link and the Communication are in a sense abstracted.

8.1 Design of the Service classes

This section describes the design of the phrasebook. Note that these classes do not need to be modified by a programmer that wishes to implement his own service. See an example for a socket-based GAP service later in the trail.

The Service class

This class is an abstract class specifying what an implementation of a service needs to have to be a service.

The Server class

This class is an implementation of a service which allows you to bootstrap other services. It is called server as it is started up the main process for all our services.

The Socket Service class

This class is an implementation of a service which allows you to connect to it by using TCP/IP sockets. It has a default implementation when a client connects which should be extended by subclasses. This is the class that most service implementors will use to create a TCP/IP based service connecting to a legacy application.

8.1.1 The Service class

This is the abstract base class for every service. It has 2 methods that subclasses need to implement:

- start, this will start the service
- stop, this will stop the service

Other than that a service is pretty much free to do what you want. The abstract base class also allows you to set some properties by using `setProperty`s, or to get the properties using `getProperty`s. In most of our implementations we first set up its properties and then we call the start method.

The code snippet below shows you the basis of the implementation of the abstract Service class (omitting the comments).

```
package nl.tue.win.riaca.service;

import java.util.*;

public abstract class Service {

    protected Properties mProperties = new Properties();

    public Properties getProperty() {
        // System.out.println( "Service.getProperty" );

        return mProperties;
    }

    public void setProperty( Properties fProperties ) {
        // System.out.println( "Service.setProperty" );

        mProperties = fProperties;
    }

    public abstract boolean start();

    public abstract boolean stop();
}
```

8.1.2 The Server class

The Server class is a specialized server that works as a super-service for services that you can start using it. Think of it as a way to start multiple services integrated into one process.

The class implements the abstract methods start and stop as specified by the abstract class. It also has an init method that constructs the services that will be started by means of the start method. In code that amounts to the following:

```
private boolean init() {
    // System.out.println( "Server.init" );

    Server.mLogger.info( "Initializing server" );

    String tServices = mProperties.getProperty( "order" );

    if ( tServices == null )
        tServices = "";

    StringTokenizer tServiceTokenizer =
        new StringTokenizer( tServices );

    for( ; tServiceTokenizer.hasMoreTokens(); )
        initService( tServiceTokenizer.nextToken() );

    return true;
}

public boolean start() {
    // System.out.println( "Server.start" );

    Server.mLogger.info( "Starting server" );

    if ( init() ) {
        Enumeration tEnum = mOrder.elements();

        for( ; tEnum.hasMoreElements(); ) {
            String tName = (String) tEnum.nextElement();
            Service tService = (Service) mServices.get( tName );
        }
    }
}
```

```

        try {
            tService.start();
        }
        catch( Exception tException ) {
            tException.printStackTrace();
        }
    }

    mThread = new Thread( this );
    mThread.start();

    return true;
}

return false;
}

public boolean stop() {
    // System.out.println( "Server.stop" );

    Server.mLogger.info( "Stopping server" );

    for( int i=mOrder.size() - 1; i >= 0; i-- ) {
        String tName    = (String) mOrder.elementAt( i );
        Service tService = (Service) mServices.get( tName );

        tService.stop();
    }

    mShutdown = true;
    mThread.interrupt();

    return true;
}

```

Note that the `init` method uses another method called `initService` which has not been included above. It is used to construct an instance of the service you want to start within the `Server`.

The rest of the code is for review in the appendix. Note that the methods above could be implemented a bit different in the actual source code.

8.1.3 The Socket Service class

The Socket Service class implements the basic skeleton for a socket based service. A socket based service allows you to bind a TCP/IP socket and a legacy program together so that you can access the legacy program from another computer.

Because the Socket Service is a service it implements the 2 abstract methods and in this case also some additional support classes that are used to export/bind a socket. The Socket Service consists of the following classes:

- EndPoint
- Handler
- Host
- SocketHandler
- SocketHost
- SocketService

The SocketService class is the main point for the service. It uses to SocketHost class to bind a specified socket to an EndPoint. So the SocketHost uses EndPoint. An EndPoint uses a Handler to handle an incoming request. In general this would be Handler, but in the specialized case of the SocketHost this is SocketHandler. And to complete the picture the SocketHost extends the behavior of the Host class.

8.2 An example implementation

An 'Gap Socket' Service

This example implements a service that exports GAP on a socket.

8.2.1 A Gap Socket Service

The code below is a part of the service that makes up the entire GapSocketService class. The entire code is for sake of completeness added as an appendix to the book.

```
public class GapSocketService
    extends SocketService {

    protected GapLink mLink;

    public boolean start() {
        if ( super.start() == false ) return false;

        // System.out.println( "GapSocketService.start" );

        try {
            String tCommand = mProperties.getProperty( "command" );

            mLink = new GapLink();
            mLink.start( tCommand );
        }
        catch( Exception tException ) {
            tException.printStackTrace();

            return false;
        }

        return true;
    }

    public boolean stop() {
        // System.out.println( "GapSocketService.stop" );

        if ( super.stop() == false ) return false;

        mLink.stop();
        mLink = null;

        return true;
    }
}
```

Note that in the above code snippet the `GapLink` class is used to access GAP. This is step 1 of the customization that is needed to support a different legacy program. Step 2 involves writing your own `SocketHandler`. See the code snippet below for an example.

```
public synchronized void handle() {
    // System.out.println( "GapSocketHandler.handle" );

    boolean          tShutdown = false;
    GapSocketService tService   = (GapSocketService) mHost.getService();
    GapLink          tLink      = tService.getLink();

    if ( tLink != null ) {
        try {
            /*
             * We need the socket input-stream to read the actual request,
             * and the socket output-stream to write the actual response.
             */

            BufferedReader tSocketInput =
                new BufferedReader(
                    new InputStreamReader( mSocket.getInputStream() ) );

            PrintWriter tSocketOutput =
                new PrintWriter( mSocket.getOutputStream() );

            String tInput  = "";
            String tOutput = "";

            while( !tSocketInput.ready() ) {
                try {
                    Thread.sleep( 500 );
                }
                catch( InterruptedException tException ) {}
            }

            tInput = tSocketInput.readLine();

            if ( !tInput.startsWith( "quit" ) ) {
                tOutput = tLink.execute( tInput );

                tSocketOutput.println( tOutput );
                tSocketOutput.flush();
            }
        }
        catch( Exception tException ) {
```

```
        tException.printStackTrace();
    }
}

try {
    mSocket.close();
}
catch( Exception tException ) {}
}
```

Chapter 9

Shells

The chapter shows you how to use Shells and to do OpenMath.

Using GAP explains you how to setup a shell for communicating with GAP and how to start the GAP application needed for this purpose. The communication can take place either in GAP syntax or in OpenMath.

Note: this portion of the book assumes that you are sitting behind a computer to try out what is mentioned. While you can read through it without the use of a computer, it is a more exciting experience to actually do the exercises.

9.1 Using GAP

This section gives a gentle introduction on how to use GAP in an OpenMath context. It describes what you need to setup and how you send a query to GAP.

Setting up

Before you can actually query GAP you will need to set it up so that you can. The necessary steps are outlined here.

Querying GAP

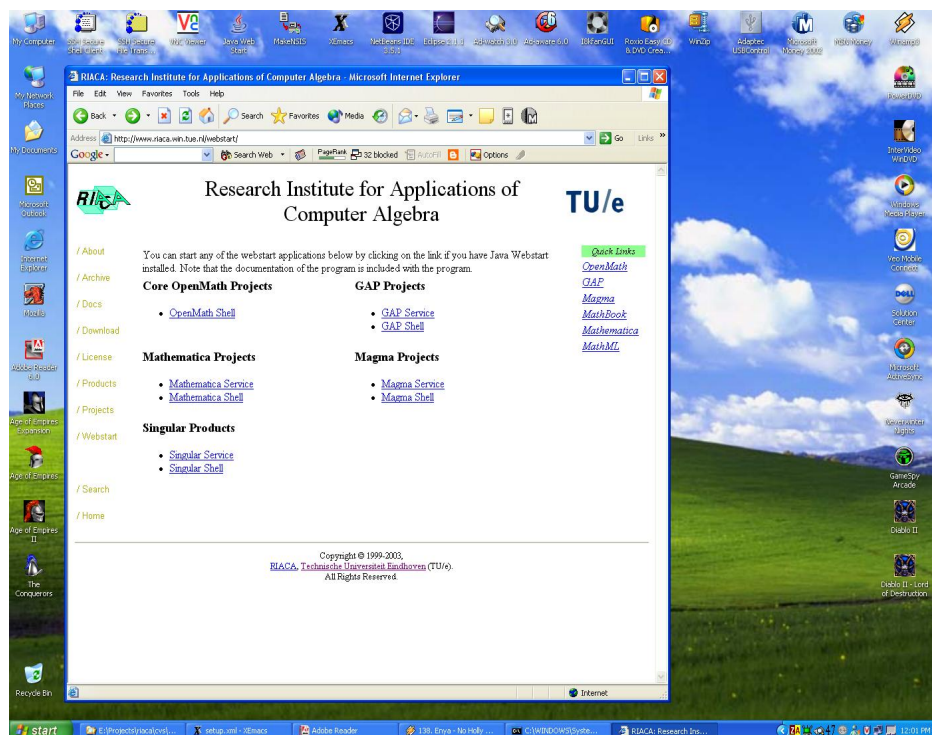
After your successful setup you are ready to query GAP. We'll outline some examples you can use as a basis for your own experiments.

9.1.1 Setting up

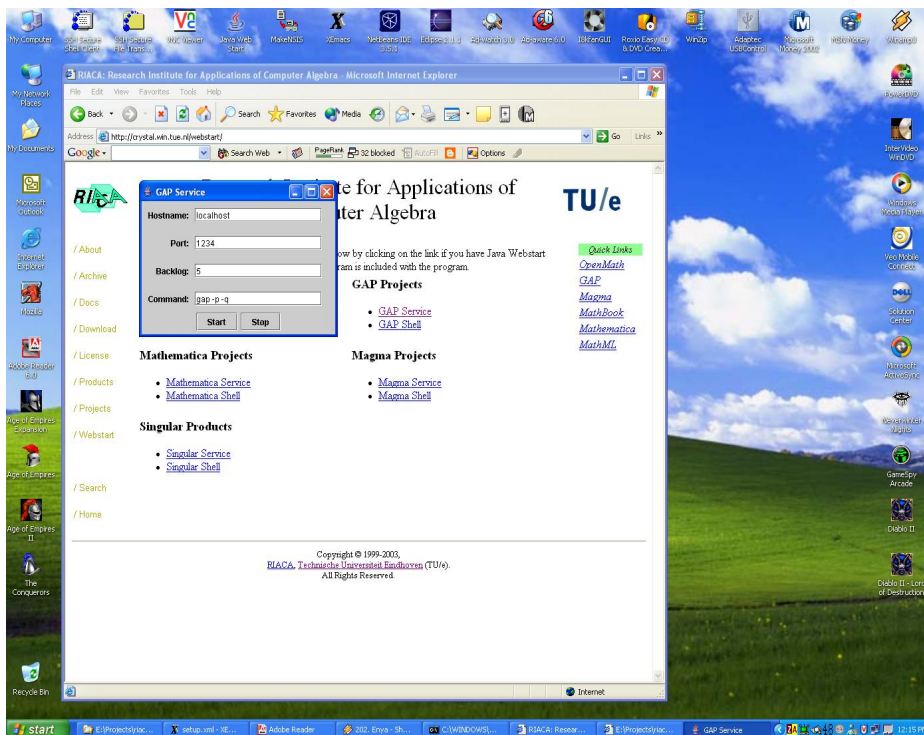
To facilitate an easy setup for users we have deployed several Java Webstart applications. If you already have Java Webstart installed you can start at

point 2 below, otherwise walk through the list from the start.

1. To install Java Webstart you need to get Java2 1.4+ installed. Please goto <http://java.sun.com/getjava> and click on the "Get It Now" button on the right side of the screen. Please follow the directions given there and return here after installing.
2. We assume you have Java Webstart installed now! Please go to the RIACA webstart web page at <http://www.riaca.win.tue.nl/webstart/>. You will see a screen similar to the following screen (click on the screen for its full size)



3. Click on the link "GAP Service" to start the GAP service. Note this action is necessary to create a running GAP service to connect to later on. If you already started your GAP service before, you can safely ignore this step. After the click the screen below will appear.



4. Please fill in the location of the GAP application. Note that the path needs to be absolute if you don't have the GAP 'bin' directory in your PATH environment variable.

Eg. for Microsoft Windows systems

`C:/Gap4r3/bin/gap.bat -p -q`

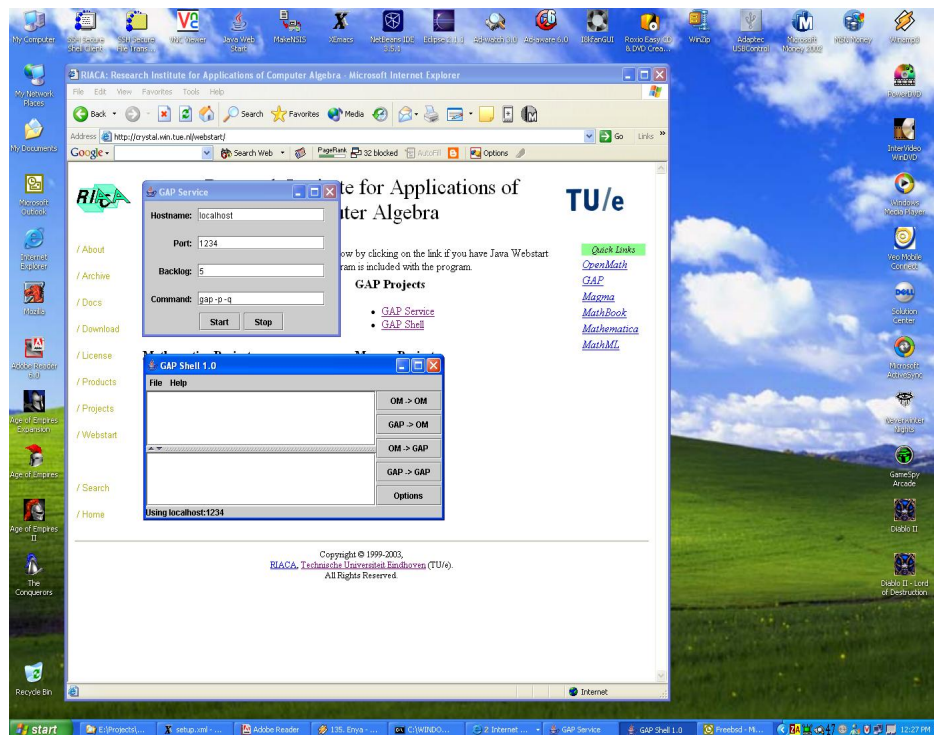
Eg. for Unix alike systems

`/usr/local/bin/gap.sh -p -q`

Note: please do NOT forget the `-p -q` at the end. The GAP service will not start correctly if these are omitted.

Now press the "Start" button. You will notice that the click on the button blocks the interface for a little while; this is due to the time it takes for GAP to initialize. Just wait until it unblocks and then continue on with the next step.

5. Now that the GAP service is running we are ready to start up the GAP Shell. Click on the "GAP Shell" in the webpage (you did keep that page open?).

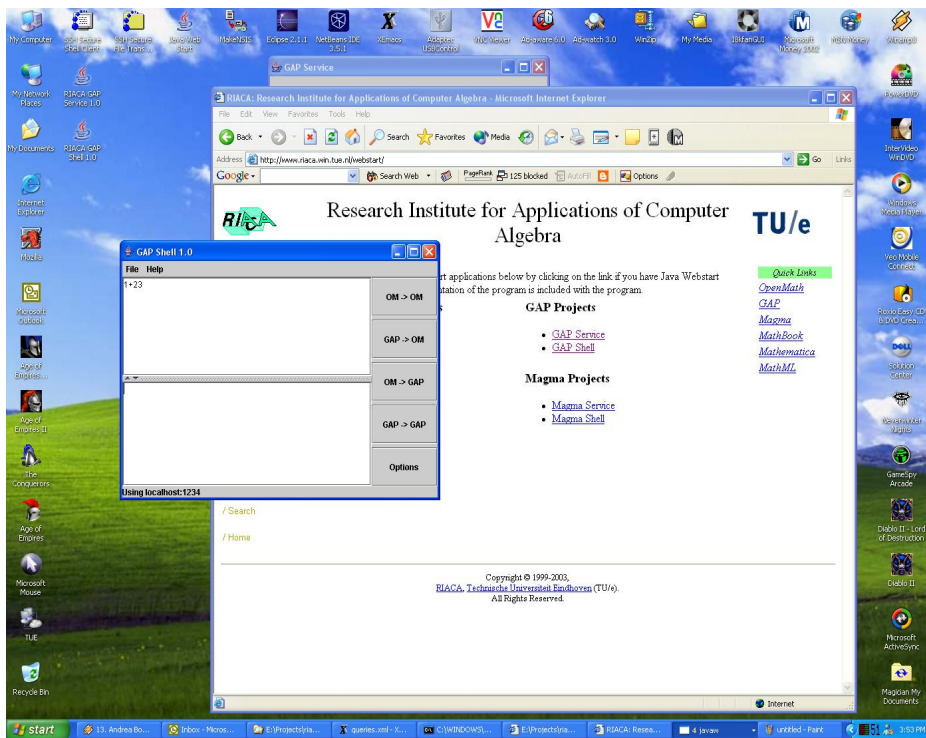


You will see a screen similar to the one above. This is all for the setup. You are ready to continue with Querying GAP.

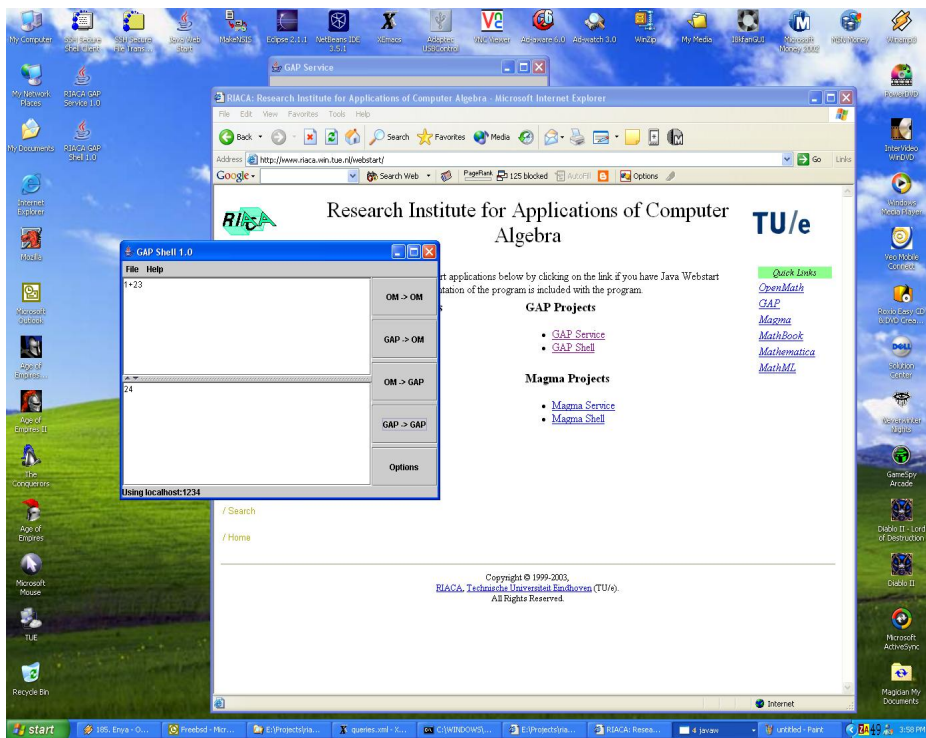
9.1.2 Querying GAP

If you don't have GAP configured and started please go back to Setting up to setup your GAP to work in an OpenMath context

Before we enter into OpenMath mode we will first send some basic GAP to the GAP we have running. Enter $1+23$ in the top edit field. See the picture below for an explanation.

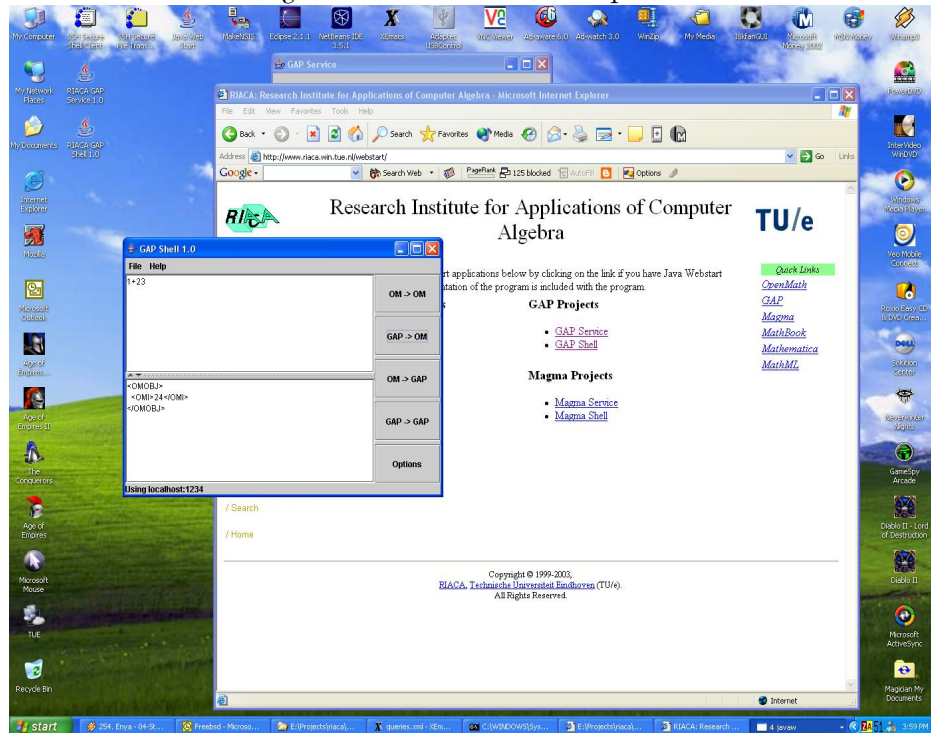


Now it is time to send it off to GAP. Press the Button that is labelled "GAP -j GAP". You should see the result given in the picture below.



We have just communicated with GAP, but we are still not using OpenMath. Well, the next step will tell you how you can input a piece of GAP, then send it to GAP, and get the result as an OpenMath object.

Instead of pressing the button labelled "GAP -j GAP", we will press the button labelled "GAP -j OM". You will see output similar to the one below.

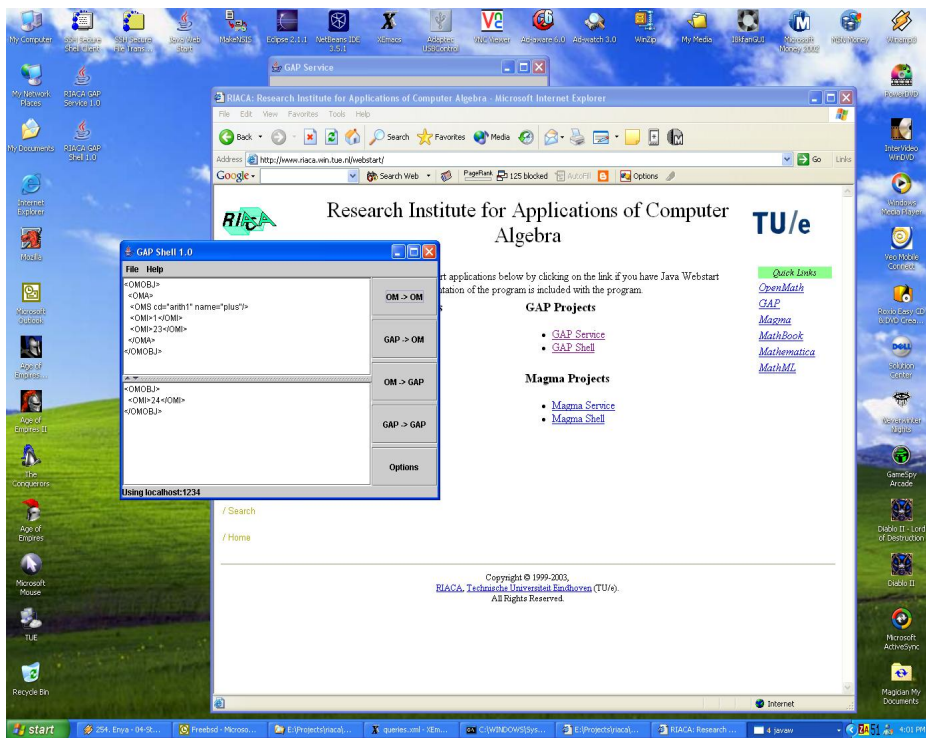


So you have now seen a basic OpenMath object returned in XML. Note that there is still one step left to take. And that is sending an OpenMath object to GAP and getting back an OpenMath object.

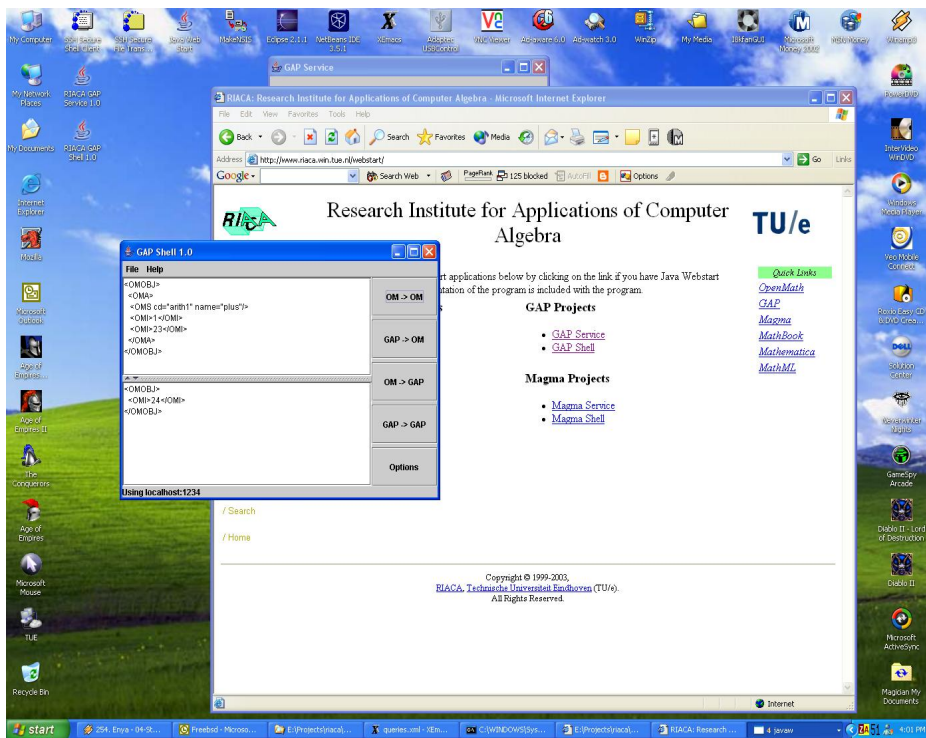
Type the following into the input field.

```
<OMOBJ>
<OMA>
  <OMS cd="arith1" name="plus"/>
  <OMI>1</OMI>
  <OMI>23</OMI>
</OMA>
</OMOBJ>
```

You should have entered something similar to the image below



And now it is time to press the button labelled "OM -i OM". As a result you should see output similar to the one below.



Now you have the basic skills needed to do OpenMath with GAP. Note that we have omitted the button labelled "OM -i GAP", this one is used if you want to send some OpenMath and get a GAP result back. The button on the bottom is for configuring the connection and is needed when you are not using the basic setup.