```java
package edu.it.disk;

public interface ByteArrayPair {
  /**
   *@return  bytes
   */
  byte[] first();


  /**
   *@return  rest of valuestream.
   */
  ValueStream rest();
}
```



```java
package edu.it.disk;

import java.util.*;

/**
 *  CachedDisk.java Created: Fri Jun 28 17:59:25 2002
 *
 *@author  <a href="mailto:kasperp@it-c.dk">Kasper Bøgebjerg Pedersen</a>
 */

public class CachedDisk implements Disk {
  private Disk disk;
  private HashMap cache;
  private int size, maxSize;


  public CachedDisk( Disk disk ) {
    this( disk, 1000 );
  }


  public CachedDisk( Disk disk, int maxSize ) {
    this.disk = disk;
    this.maxSize = maxSize;
    this.cache = new HashMap();
  }


  public String getName() {
    return disk.getName();
  }


  public ValueStream load( ValueReference vsr ) throws DiskException {
    ValueStream vs = (ValueStream) cache.get( vsr.toByteArray() );
    if ( vs == null ) {
      // vs not in cache
      vs = disk.load( vsr );
    }
    return vs;
  }


  private void cache( ValueReference vsr, ValueStream vs ) {
    if ( size >= maxSize ) {
      // empty cache
      cache.clear();
    }
    cache.put( vsr.toByteArray(), vs );
  }


  public ValueReference save( ValueStream vs ) throws DiskException {
    ValueReference vsr = disk.save( vs );
    cache( vsr, vs );
    return vsr;
  }


  /**
   *  Cells cannot easily be cached, as they may change
   */
```

```
  public CellReference saveCell( ValueStream vs ) throws DiskException {
    return saveCell( vs );
  }
}
```



```
package edu.it.disk;

interface CellLocator extends Locator {
  void set( ValueReference vsr ) throws DiskException;
}
```

```
package edu.it.disk;

public interface CellReference extends ValueReference {
  /**
   *  Updates the cell to reference a new valuestream.
   */
  void set( ValueReference vsr ) throws DiskException;
}
```



```
package edu.it.disk;

public interface CellReferencePair {

  /**
   *@return  cell
   */
  CellReference first();


  /**
   *@return  rest of valuestream.
   */
  ValueStream rest();
}
```

```java
package edu.it.disk;

/**
 *  Disk objects reprecent the lowest level of a xmlstore, a Disk has two
 *  methods load and save. Where load loads a ValueStream from a Locator
 *  and save saves a ValueStream and returns a Locator. The actual
 *  implementation of how ValueStreams are represented on physical as well
 *  as the virtual is hidden from the user (client).
 */
public interface Disk {
  ValueReference save( ValueStream valueStream ) throws DiskException;


  CellReference saveCell( ValueStream valueStream ) throws DiskException;


  ValueStream load( ValueReference vsr ) throws DiskException;


  String getName();
}
```



```java
package edu.it.disk;

public class DiskException extends Exception {
  public final static byte UN_KNOWN = 0x00;
  public final static byte INIT = 0x01;
  public final static byte TYPE = 0x02;
  public final static byte IO = 0x03;
  public final static byte LOOKUP = 0x04;
  public final static byte INSTANTIATION = 0x05;
  public final static byte CLASSNOTFOUND = 0x06;
  public final static byte ILLEGAL_ACCESS = 0x07;
  public final static byte PROPS = 0x08;
  public final static byte UNKNOWN_HOST = 0x09;

  private byte type;


  public DiskException( byte type, Exception e ) {
    super( e );
    this.type = type;
  }


  public DiskException( String msg, byte type ) {
    super( msg );
    this.type = type;
  }


  public DiskException( String msg ) {
    this( msg, (byte) 0x00 );
  }


  public DiskException() {
    this( "", (byte) 0x00 );
  }


  public byte getType() {
    return type;
  }
}
```

```
package edu.it.disk;


import edu.it.properties.*;

public class DiskFactory {
  public final static String LOCALDISK = "LocalDisk";
  public final static String GLOBALDISK = "GlobalDisk";
  public final static String BUFFEREDDISK = "BufferedDisk";
  private String propsId = null;
  private String diskName = null;


  public DiskFactory() { }


  protected Disk createDisk( String diskDesc, Disk innerDisk ) throws DiskExcept
ion {
    String[] infos = diskDesc.split( " " );
    if ( infos.length == 2 && infos[0].equals( GLOBALDISK ) ) {
      return new GlobalDisk( innerDisk, Integer.parseInt( infos[1] ) );
    }
    if ( infos.length == 1 && infos[0].equals( BUFFEREDDISK ) ) {
      return new BufferedDisk( innerDisk );
    }
    error();
    return null;
  }


  protected Disk createDisk( String diskDesc ) throws DiskException {
    String[] infos = diskDesc.split( " " );
    if ( infos.length == 1 && infos[0].equals( LOCALDISK ) ) {
      return new LocalDisk( diskName + ".disk" );
    }
    error();
    return null;
  }


  public Disk createDisk( String name, String propsId ) throws DiskException {
    try {
      diskName = name;
      this.propsId = propsId;
      String diskDesc = XMLStoreProperties.getInstance( propsId ).getDisk();
      String[] descriptions = diskDesc.split( ";" );
      Disk disk = createDisk( descriptions[descriptions.length - 1].trim() );

      for ( int i = descriptions.length - 2; i >= 0; i-- ) {
        disk = createDisk( descriptions[i], disk );
      }
      return disk;
    } catch ( PropertyException e ) {
      throw new DiskException( DiskException.PROPS, e );
    }
  }


  private void error() {
    String msg = "disk description is not a valid/known description";
    throw new IllegalArgumentException( msg );
  }
```



```

  public static DiskFactory create( String propsId ) throws DiskException {
    try {
      String classN = XMLStoreProperties.getInstance( propsId ).getDiskFactory()
;
      return (DiskFactory) ( Class.forName( classN ).newInstance() );
    } catch ( InstantiationException e ) {
      throw new DiskException( DiskException.INSTANTIATION, e );
    } catch ( ClassNotFoundException e ) {
      throw new DiskException( DiskException.CLASSNOTFOUND, e );
    } catch ( IllegalAccessException e ) {
      throw new DiskException( DiskException.ILLEGAL_ACCESS, e );
    } catch ( PropertyException e ) {
      throw new DiskException( DiskException.PROPS, e );
    }
  }
}
// DiskFactory
```

```
package edu.it.disk;

import java.io.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

class DiskUtil {
  public static byte[] value2ByteArray( ValueStream vs ) throws DiskException {
    try {
      ByteArrayOutputStream out = new ByteArrayOutputStream();
      while ( vs.getValueType() != ValueType.EMPTY ) {
        switch ( vs.getValueType() ) {
            case ValueType.BYTES:
              ByteArrayPair barr = vs.getBytes();
              out.write( barr.first() );
              vs = barr.rest();
              break;
            case ValueType.V_REF:
              ValueReferencePair vsrPair = vs.getValueReference();
              out.write( ValueType.V_REF );
              //delimiter
              out.write( vsrPair.first().toByteArray() );
              vs = vsrPair.rest();
              break;
            case ValueType.C_REF:
              CellReferencePair cellPair = vs.getCell();
              out.write( ValueType.C_REF );
              //delimiter
              out.write( cellPair.first().toByteArray() );
              vs = cellPair.rest();
              break;
        }
      }
      return out.toByteArray();
    } catch ( IOException e ) {
      throw new DiskException( e.getMessage(), DiskException.IO );
    }
  }


  static byte[] getValueRef( byte[] valueStream ) {
    MessageDigest md = null;
    try {
      md = MessageDigest.getInstance( "MD5" );
    } catch ( NoSuchAlgorithmException e ) {}
    md.update( valueStream );
    return md.digest();
  }

}
```



```
package edu.it.disk;

import java.io.*;

/**
 *  DiskValueStream.java Created: Fri May 31 11:02:31 2002
 *
 *@author  <a href="mailto:kasperp@it-c.dk">Kasper Bøgebjerg Pedersen</a>
 */

class DiskValueStream implements ValueStream {

  private final int BUFSIZE;
  protected byte type;
  protected String diskname;
  protected int off, len;
  //memorization
  private ValueStream rest;
  private byte[] bytes;
  private ValueReferencePair vsrPair;
  private CellReferencePair crPair;
  protected ReferenceFactory refFactory;


  DiskValueStream( String diskname, int off, int len ) throws DiskException {
    this( diskname, off, len, ValueType.BYTES );
  }


  DiskValueStream( String diskname, int off, int len, byte type ) throws DiskExc
eption {
    this.diskname = diskname;
    this.off = off;
    this.len = len;
    this.type = type;

    this.vsrPair = null;
    this.crPair = null;
    String bufsize = System.getProperty( "Byte_buf_size" );
    int bufs;
    try {
      bufs = Integer.parseInt( bufsize );
    } catch ( Exception e ) {
      bufs = 1000;
    }
    //property not set or not set properly
    BUFSIZE = bufs;
    refFactory = SimpleReferenceFactory.getInstance();
  }


  public byte getValueType() throws DiskException {
    return type;
  }


  //MEMORAIZATION STILL NEED TO BE IMPLEMENTED
  public ByteArrayPair getBytes() throws DiskException {
    return getBytes( -1 );
  }
```

```
  public ByteArrayPair getBytes( int num ) throws DiskException {
    if ( type != ValueType.BYTES ) {
      throw new DiskException( "Wrong Value type: is " + type + " should be "
          + ValueType.BYTES,
          DiskException.TYPE );
    }
    InputStream in = getInputStream();
    //read from RAF stream not knowing number of bytes, a BufferList is used.
    //BufferList is a list of byte arrays
    byte nexttype = ValueType.EMPTY;
    byte[] byteBuffer = new byte[BUFSIZE];
    BufferList inputChunks = null;
    int bytesRead = 0;
    int currRead = 0;
    int delimiter = 0;
    try {
      int currByte;
      while ( ( ( currByte = in.read() ) != -1 ) ) {
        // -1 == EOF
        //max number is specified and has been read (i.e. there are bytes left i
n stream)
        if ( num != -1 && currRead >= num ) {
          // -1 == read all bytes;
          nexttype = ValueType.BYTES;
          break;
        }
        // no more bytes, next value in stream is a reference
        if ( currByte == ValueType.V_REF || currByte == ValueType.C_REF ) {
          nexttype = (byte) currByte;
          delimiter = 1;
          break;
        }
        //buffer is full make new chunk
        if ( currRead >= byteBuffer.length ) {
          inputChunks = new BufferList( byteBuffer, currRead, inputChunks );
          currRead = 0;
          byteBuffer = new byte[BUFSIZE];
        }
        //add read byte to buffer
        byteBuffer[currRead++] = (byte) currByte;
        bytesRead++;
      }
      in.close();
    } catch ( IOException e ) {
      throw new DiskException( e.getMessage(), DiskException.IO );
    }
    byte[] result = null;
    inputChunks = new BufferList( byteBuffer, currRead, inputChunks );
    result = toArray( inputChunks, bytesRead );
    return new SimpleByteArrayPair( result,
        getRest( off + bytesRead + delimiter, len - bytesRead, nexttype ) );
  }


  public ValueReferencePair getValueReference() throws DiskException {
    if ( type != ValueType.V_REF ) {
      throw new DiskException( "Wrong Value type is " + type + " should be " + ValueType.
V_REF,
          DiskException.TYPE );
    }
    if ( vsrPair == null ) {
      InputStream in = getInputStream();
```



```
      byte[] ref = new byte[ValueReference.LENGTH];
      try {
        in.read( ref );
        in.close();
      } catch ( IOException e ) {
        throw new DiskException( e.getMessage(), DiskException.IO );
      }
      rest = getRest( off + ValueReference.LENGTH, len - ValueReference.LENGTH -
 1 );
      vsrPair = new SimpleValueReferencePair( refFactory.createValueReference( r
ef ), rest );
    }
    return vsrPair;
  }


  public CellReferencePair getCell() throws DiskException {
    if ( type != ValueType.C_REF ) {
      throw new DiskException( "Wrong Value type is " + type + " should be " + ValueType.
C_REF,
          DiskException.TYPE );
    }
    if ( crPair == null ) {
      InputStream in = getInputStream();
      byte[] ref = new byte[ValueReference.LENGTH];
      try {
        in.read( ref );
        in.close();
      } catch ( IOException e ) {
        throw new DiskException( e.getMessage(), DiskException.IO );
      }
      rest = getRest( off + ValueReference.LENGTH, len - ValueReference.LENGTH -
 1 );
      crPair = new SimpleCellReferencePair( refFactory.createCellReference( ref
), rest );
    }
    return crPair;
  }


  public ValueStream skip() throws DiskException {
    if ( rest == null ) {
      switch ( type ) {
          case ValueType.BYTES:
            rest = getBytes().rest();
            break;
          case ValueType.V_REF:
            getValueReference();
            break;
          case ValueType.C_REF:
            getCell();
            break;
      }
    }
    return rest;
  }


  protected InputStream getInputStream() throws DiskException {
    try {
      //return new RAFInputStream( diskname, off, len );
      return new BufferedInputStream( new RAFInputStream( diskname, off, len ) )
```

```java
;
    } catch ( IOException e ) {
      throw new DiskException( e.getMessage(), DiskException.IO );
    }
  }


  protected ValueStream getRest( int off, int len, byte type ) throws DiskExcept
ion {
    return new DiskValueStream( diskname, off, len, type );
  }


  protected ValueStream getRest( int off, int len ) throws DiskException {
    return getRest( off, len, ValueType.BYTES );
  }


  /**
   *  Class for representing lists of byte buffers
   */
  private class BufferList {
    public byte[] byteBuffer;
    public int length;
    public BufferList next;


    public BufferList( byte[] byteBuffer, int length, BufferList next ) {
      this.byteBuffer = byteBuffer;
      this.length = length;
      this.next = next;
    }
  }


  private static byte[] toArray( BufferList inputChunks, int size ) {
    //put byte Buffer in one byte[]
    byte[] resultArr = new byte[size];

    while ( inputChunks != null ) {
      for ( int i = inputChunks.length; i > 0;  ) {
        resultArr[--size] = inputChunks.byteBuffer[--i];
      }
      inputChunks = inputChunks.next;
    }
    return resultArr;
  }
}
```



```java
package edu.it.disk;

import java.io.IOException;
import java.io.RandomAccessFile;

import java.util.HashMap;
import java.util.Vector;

public class FilePool {
  private static FilePool instance;
  private HashMap poolDictionary;


  private FilePool() {
    poolDictionary = new HashMap();
  }


  public RandomAccessFile acquireFile( String filename ) throws IOException {
    Vector pool = (Vector) poolDictionary.get( filename );
    if ( pool != null ) {
      int size = pool.size();
      if ( size > 0 ) {
        return (RandomAccessFile) pool.remove( size - 1 );
      }
    }
    // if null
    return new RandomAccessFileProxy( filename, "r" );
  }


  public void releaseFile( RandomAccessFileProxy file ) {
    String fileName = file.getName();
    Vector pool = (Vector) poolDictionary.get( fileName );
    if ( pool == null ) {
      pool = new Vector();
      if ( pool.size() < 10 ) {
        poolDictionary.put( fileName, pool );
        pool.addElement( file );
      }
      // maybe a check on the number of objects allready in pool
      else {
        try {
          file.realclose();
        } catch ( Exception e ) {}
      }
    }
  }


  public static FilePool getInstance() {
    if ( instance == null ) {
      instance = new FilePool();
    }
    return instance;
  }
}
```

```java
package edu.it.disk;


import java.io.*;
import java.net.*;
import java.util.Arrays;

public class GlobalCellLocator extends GlobalLocator implements CellLocator {

  public GlobalCellLocator( InetAddress ip, int port, String diskname, int off,
int len ) throws DiskException {
    super( ip, port, diskname, off, len );
  }


  GlobalCellLocator( byte[] ip, int port, String diskname, int off, int len ) th
rows DiskException {
    super( ip, port, diskname, off, len );
  }


  public void set( ValueReference vr ) throws DiskException {
    if ( ip.equals( LOCAL_IP ) ) {
      //local locator
      new LocalCellLocator( diskname, off, len ).set( vr );
    }
    try {
      Socket socket = new Socket( ip, port );
      DataOutputStream out = new DataOutputStream( new BufferedOutputStream( soc
ket.getOutputStream() ) );
      out.write( GlobalDisk.SET_CELL );
      out.writeUTF( diskname );
      out.writeInt( off );
      out.writeInt( len );
      out.write( vr.toByteArray() );
      out.close();
      socket.close();
    } catch ( IOException e ) {
      throw new DiskException( e.getMessage() + " Error: setting cell",
          DiskException.IO );
    }
  }


  public String toString() {
    try {
      return Byte.toString( Locator.CELL ) + " " + ip.getHostAddress() + " " + po
rt + " " + diskname + " " + off + " " + len;
    } catch ( Exception e ) {
      return "";
    }
  }
}
```



```java
package edu.it.disk;


import edu.it.disk.refserver.ReferenceServer;
import java.io.*;
import java.net.*;

public class GlobalDisk implements Disk {
  final static byte READ = 0x00;
  final static byte READ_LEN = 0x01;
  final static byte SET_CELL = 0x02;

  private Disk disk;
  private BufferedOutputStream out;
  private ServerSocket serverSock;


  public GlobalDisk( Disk disk, int port ) throws DiskException {
    ReferenceServer.mkGlobal();
    this.disk = disk;
    try {
      this.serverSock = new ServerSocket( port );
    } catch ( IOException e ) {
      throw new DiskException( DiskException.IO, e );
    }
    LocatorFactory.getInstance( disk.getName() ).mkGlobal( port );
      new Thread() {
        public void run() {
          while ( true ) {
            //something to stop needs to be implemented
            try {
              handleRequest( serverSock.accept() );
            } catch ( IOException e ) {
              Thread.currentThread().interrupt();
            }
          }
        }
      }.start();
  }


  private void handleRequest( final Socket sock ) {
      new Thread() {
        public void run() {
          try {
            DataInputStream in = new DataInputStream( new BufferedInputStream( s
ock.getInputStream() ) );
            out = new BufferedOutputStream( sock.getOutputStream() );
            int bulk = -1;
            switch ( in.read() ) {
                case READ_LEN:
                  bulk = in.readInt();
                //fallthough
                case READ:
                  String diskname = in.readUTF();
                  int off = in.readInt();
                  int len = in.readInt();
                  handleRead( diskname, off, len, bulk );
                  break;
                case SET_CELL:
                  String diskname1 = in.readUTF();
                  int off1 = in.readInt();
```

```
                  int len1 = in.readInt();
                  byte[] vsr = new byte[ValueReference.LENGTH];
                  in.read( vsr );
                  try {
                    new LocalCellLocator( diskname1, off1, len1 ).set( new Simpl
eValueReference( vsr ) );
                  } catch ( DiskException e ) {}
            }
          } catch ( IOException e ) {
            Thread.currentThread().interrupt();
          }
        }
      }.start();
  }


  private void handleRead( String diskname, int off, int len, int bulk ) throws
IOException {
    //InputStream in = new RAFInputStream( diskname, off, len );
    InputStream in = new BufferedInputStream( new RAFInputStream( diskname, off,
 len ) );
    if ( bulk == -1 ) {
      //just read one byte
      out.write( in.read() );
      out.flush();
      return;
    }
    byte[] b = new byte[bulk];
    in.read( b, 0, bulk );
    in.close();
    out.write( b );
    out.flush();
  }


  public ValueReference save( ValueStream vs ) throws DiskException {
    return disk.save( vs );
  }


  public CellReference saveCell( ValueStream vs ) throws DiskException {
    return disk.saveCell( vs );
  }


  public ValueStream load( ValueReference vsr ) throws DiskException {
    return disk.load( vsr );
  }


  public String getName() {
    return disk.getName();
  }
}
```



```
package edu.it.disk;

import java.net.InetAddress;
import java.net.UnknownHostException;


import java.util.Arrays;

public class GlobalLocator extends LocalLocator {
  protected static InetAddress LOCAL_IP;
  static {
    try {
      LOCAL_IP = InetAddress.getLocalHost();
      System.out.println( LOCAL_IP.getHostAddress() );
      //System.out.println(InetAddress.getByAddress(LOCAL_IP));
    } catch ( UnknownHostException e ) {
      System.out.println( "error getting localhost" );
    }
  }
  protected InetAddress ip;
  protected int port;


  public GlobalLocator( InetAddress ip, int port, String diskname, int off, int
len ) throws DiskException {
    super( diskname, off, len );
    this.ip = ip;
    this.port = port;
  }


  public GlobalLocator( byte[] ip, int port, String diskname, int off, int len )
 throws DiskException {
    super( diskname, off, len );
    try {
      this.ip = InetAddress.getByAddress( ip );
    } catch ( UnknownHostException e ) {
      throw new DiskException( DiskException.UNKNOWN_HOST, e );
    }
    this.port = port;
  }


  public GlobalLocator( byte[] ip, int port, LocalLocator loc ) throws DiskExcep
tion {
    this( ip, port, loc.diskname, loc.off, loc.len );
  }


  public ValueStream getValueStream() throws DiskException {
    if ( ip.equals( LOCAL_IP ) ) {
      //local locator
      return super.getValueStream();
    }
    return new GlobalValueStream( ip.getAddress(), port, diskname, off, len );
  }


  public String toString() {
    try {
      return Byte.toString( Locator.BOX ) + " " + ip.getHostAddress() + " " + por
t + " " + diskname + " " + off + " " + len;
```

```
    } catch ( Exception e ) {
      return "";
    }
  }
}
```



```
package edu.it.disk;


import java.io.*;

public class GlobalValueStream extends DiskValueStream {
  private byte[] ip;
  private int port;


  GlobalValueStream( byte[] ip, int port, String diskname, int off, int len, byt
e type ) throws DiskException {
    super( diskname, off, len, type );
    this.ip = ip;
    this.port = port;
  }


  GlobalValueStream( byte[] ip, int port, String diskname, int off, int len ) th
rows DiskException {
    this( ip, port, diskname, off, len, ValueType.BYTES );
  }


  protected InputStream getInputStream() throws DiskException {
    try {
      return new BufferedInputStream( new SocketInputStream( ip, port, diskname,
 off, len ) );
    } catch ( IOException e ) {
      throw new DiskException( e.getMessage(), DiskException.IO );
    }
  }


  protected ValueStream getRest( int off, int len, byte type ) throws DiskExcept
ion {
    return new GlobalValueStream( ip, port, diskname, off, len, type );
  }
}
```

```java
package edu.it.disk;


import java.io.*;

public class LocalCellLocator extends LocalLocator implements CellLocator {
  LocalCellLocator( String diskname, int off, int len ) {
    super( diskname, off, len );
  }


  public void set( ValueReference vsr ) throws DiskException {
    try {
      RandomAccessFile raf = new RandomAccessFile( diskname, "rw" );
      raf.seek( off + 1 );
      raf.write( vsr.toByteArray() );
      raf.close();
    } catch ( IOException e ) {
      throw new DiskException( e.getMessage() + " Error: setting cell",
          DiskException.IO );
    }
  }


  public String toString() {
    return Byte.toString( Locator.CELL ) + " " + diskname + " " + off + " " + len
;
  }
}
```



```java
package edu.it.disk;

import edu.it.disk.refserver.*;


import java.io.*;

public class LocalDisk implements Disk {
  private RandomAccessFile file;
  private final String name;
  protected ReferenceServer referenceServer;
  private LocatorFactory locFactory;
  private ReferenceFactory refFactory;


  public LocalDisk( String name ) throws DiskException {
    this.name = name;
    locFactory = LocatorFactory.getInstance( name );
    refFactory = SimpleReferenceFactory.getInstance();
    try {
      file = new RandomAccessFile( name, "rw" );
      file.seek( file.length() );
      //allways append
    } catch ( IOException e ) {
      throw new DiskException( e.toString(), DiskException.INIT );
    }
    referenceServer = LocalReferenceServer.getInstance();
  }


  /**
   *  Saves the the given valueStream on disk. The following process is
   *  followed
   *  <ul>
   *    <li> Read ValueStream fully (if valuestreams are large this may
   *    crash the system), compute valueRef (MD5 sum)</li>
   *    <li> check for macthing ValueRefs (i.e the value allready exists on
   *    some disk), (nameServer)</li>
   *    <li> if ValueStream exists just return ValueReference else, write
   *    ValueStream to disk, bind ValueReference to Locator and return
   *    ValueReference</li>
   *  </ul>
   *  method holds the following invariant, v.equals(d.load(d.save(v))) ==
   *  true ValueStreams are saved in a logstructured fashion.
   *
   *@return  ValueReference that locates valueStream
   */
  public ValueReference save( ValueStream vs ) throws DiskException {
    byte[] value = DiskUtil.value2ByteArray( vs );
    //compute md5 sum from serialized object
    byte[] ref = DiskUtil.getValueRef( value );

    ValueReference vsr = refFactory.createValueReference( ref );
    Locator loc = referenceServer.lookup( vsr );

    if ( loc == null ) {
      // value doesn't exists write value to disk
      try {
        int off = (int) file.getFilePointer();
        file.write( value );
        Locator locator = locFactory.createLocator( off, value.length );
        vsr = refFactory.createValueReference( ref, locator );
```

```java
        // make new vsr that cache locator
        referenceServer.bind( vsr, locator );
      } catch ( IOException e ) {
        throw new DiskException( e.toString() );
      }
    }
    return vsr;
  }


  public CellReference saveCell( ValueStream vs ) throws DiskException {
    ValueReference vsr = save( vs );
    ValueStreamFactory factory = SimpleValueStreamFactory.getInstance();
    ValueStream refvs = factory.mkValRefValueStream( vsr, factory.mkEmpty() );
    CellLocator locator = null;
    byte[] value = DiskUtil.value2ByteArray( refvs );
    //compute md5 sum from serialized object
    CellReference cell = refFactory.createCellReference( DiskUtil.getValueRef( v
alue ) );
    Locator loc = referenceServer.lookup( cell );
    if ( loc == null ) {
      // value doesn't exists write value to disk
      try {
        int off = (int) file.getFilePointer();
        file.write( value );
        Locator loc1 = locFactory.createCellLocator( off, value.length );
        referenceServer.bind( cell, loc1 );
      } catch ( IOException e ) {
        throw new DiskException( e.toString() );
      }
    }
    return cell;
  }


  /**
   *  load a ValueStream from Disk. method holds the following invariant,
   *  v.equals(d.load(d.save(v)) == true
   *
   *@return  ValueStream "pointed to" by locator
   */
  public ValueStream load( ValueReference vr ) throws DiskException {
    return vr.get();
  }


  public String getName() {
    return name;
  }


  public void finalize() {
    try {
      file.close();
    } catch ( IOException e ) {}
  }
}
```



```java
package edu.it.disk;


import java.io.*;

public class LocalLocator implements Locator {
  protected int off;
  protected int len;
  protected String diskname;


  public LocalLocator( String diskname, int off, int len ) {
    this.diskname = diskname;
    this.off = off;
    this.len = len;
  }


  public ValueStream getValueStream() throws DiskException {
    return new DiskValueStream( diskname, off, len );
  }


  public String toString() {
    return Byte.toString( Locator.BOX ) + " " + diskname + " " + off + " " + len;
  }
}
```

```java
package edu.it.disk;


import java.io.*;

public interface Locator {
  public final static byte BOX = 0x01;
  public final static byte CELL = 0x02;


  ValueStream getValueStream() throws DiskException;
}
```



```java
package edu.it.disk;

import java.util.HashMap;

public class LocatorFactory {
  private static HashMap factories = new HashMap();
  private int port;
  private String diskname;


  private LocatorFactory( String diskname ) {
    this.diskname = diskname;
    port = -1;
  }


  static LocatorFactory getInstance( String diskname ) {
    LocatorFactory instance = (LocatorFactory) factories.get( diskname );
    if ( instance == null ) {
      instance = new LocatorFactory( diskname );
      factories.put( diskname, instance );
    }
    return instance;
  }


  public void mkGlobal( int port ) {
    this.port = port;
  }


  public Locator createLocator( int off, int len ) throws DiskException {
    if ( port == -1 ) {
      return new LocalLocator( diskname, off, len );
    }
    return new GlobalLocator( GlobalLocator.LOCAL_IP, port, diskname, off, len )
;
  }


  public Locator createCellLocator( int off, int len ) throws DiskException {
    if ( port == -1 ) {
      return new LocalCellLocator( diskname, off, len );
    }
    return new GlobalCellLocator( GlobalLocator.LOCAL_IP, port, diskname, off, l
en );
  }


  public static Locator createLocalLocator( String diskname, int off, int len )
       throws DiskException {
    return new LocalLocator( diskname, off, len );
  }


  public static Locator createGlobalLocator( byte[] ip, int port,
      String diskname, int off, int len )
       throws DiskException {
    return new GlobalLocator( ip, port, diskname, off, len );
  }
```

```java
  public static Locator createLocalCellLocator( String diskname, int off,
      int len )
       throws DiskException {
    return new LocalCellLocator( diskname, off, len );
  }


  public static Locator createGlobalCellLocator( byte[] ip, int port,
      String diskname, int off,
      int len ) throws DiskException {
    return new GlobalCellLocator( ip, port, diskname, off, len );
  }
}
```



```java
package edu.it.disk;

import java.io.*;

/**
 *  RAFStream.java Created: Mon May 27 12:47:28 2002
 *
 *@author  <a href="mailto:kasperp@it-c.dk">Kasper Bøgebjerg Pedersen</a>
 */
class RAFInputStream extends InputStream {
  private RandomAccessFile raf;
  private int off, len, bytesRead;


  RAFInputStream( String filename, int off, int len ) throws IOException {
    this.off = off;
    this.len = len;
    this.raf = FilePool.getInstance().acquireFile( filename );
    this.bytesRead = 0;
  }


  public int read() throws IOException {
    if ( bytesRead == -1 || bytesRead >= len ) {
      return -1;
    }
    raf.seek( off++ );
    bytesRead++;
    return raf.read();
  }


  public int read( byte[] b, int off, int len ) throws IOException {
    if ( bytesRead == -1 || bytesRead >= this.len ) {
      //EOF
      return -1;
    }
    if ( len > this.len ) {
      //trying to read more bytes than are in stream
      len = this.len;
    }
    raf.seek( this.off );
    // move filepointer to current
    bytesRead = raf.read( b, off, len );
    if ( bytesRead != -1 ) {
      this.off += bytesRead;
    }
    return bytesRead;
  }


  public void close() throws IOException {
    raf.close();
  }
}
```

```java
package edu.it.disk;

import java.io.File;
import java.io.IOException;

import java.io.RandomAccessFile;

public class RandomAccessFileProxy extends RandomAccessFile {
  private String name;


  public RandomAccessFileProxy( File file, String mode ) throws IOException {
    super( file, mode );
  }


  public RandomAccessFileProxy( String name, String mode ) throws IOException {
    super( name, mode );
    this.name = name;
  }


  public String getName() {
    return name;
  }


  /*
   *  public void close() throws IOException{
   *  FilePool pool = FilePool.getInstance();
   *  pool.releaseFile(this);
   *  }
   */
  public void realclose() throws IOException {
    super.close();
  }
  /*
   *  protected void finalize(){
   *  try{
   *  super.close();
   *  }catch(Exception e){}
   *  }
   */
}
```



```java
package edu.it.disk;

public interface ReferenceFactory {
  ValueReference createValueReference( byte[] ref );


  ValueReference createValueReference( byte[] ref, Locator loc );


  CellReference createCellReference( byte[] ref );


  CellReference createCellReference( byte[] ref, CellLocator loc );
}
```

```java
package edu.it.disk;

/**
 *  SimpleByteArrayPair.java Created: Thu May 30 22:07:23 2002
 *
 *@author  <a href="mailto:kasperp@it-c.dk">Kasper Bøgebjerg Pedersen</a>
 */

public class SimpleByteArrayPair implements ByteArrayPair {
  private byte[] first;
  private ValueStream rest;


  SimpleByteArrayPair( byte[] first, ValueStream rest ) {
    this.first = first;
    this.rest = rest;
  }


  public byte[] first() {
    return first;
  }


  public ValueStream rest() {
    return rest;
  }

}
```



```java
package edu.it.disk;


import edu.it.disk.refserver.ReferenceServer;

class SimpleCellReference extends SimpleValueReference implements CellReference {

  protected CellLocator cellLocator;


  SimpleCellReference( byte[] ref ) {
    this( ref, null );
  }


  SimpleCellReference( byte[] ref, CellLocator loc ) {
    super( ref, loc );
    this.cellLocator = loc;
  }


  public ValueStream get() throws DiskException {
    if ( cellLocator == null ) {
      cellLocator = (CellLocator) refServer.lookup( this );
    }
    if ( cellLocator == null ) {
      throw new DiskException( "get Lookup1 failed", DiskException.LOOKUP );
    }
    ValueStream vs = cellLocator.getValueStream();
    ValueReference vsr = vs.skip().getValueReference().first();
    Locator l = refServer.lookup( vsr );
    if ( l == null ) {
      throw new DiskException( "get Lookup2 failed", DiskException.LOOKUP );
    }
    return l.getValueStream();
  }


  public void set( ValueReference vsr ) throws DiskException {
    if ( cellLocator == null ) {
      cellLocator = (CellLocator) refServer.lookup( this );
    }
    if ( cellLocator == null ) {
      throw new DiskException( "set Lookup failed", DiskException.LOOKUP );
    }
    cellLocator.set( vsr );
  }


  public int hashCode() {
    return super.hashCode();
  }


  public boolean equals( Object o ) {
    if ( this == o ) {
      return true;
    }
    if ( !( o instanceof SimpleCellReference ) ) {
      return false;
    }
    return super.equals( o );
```

```
  }
}
```



```
package edu.it.disk;

/**
 *  SimpleCellReferencePair.java Created: Fri May 31 09:46:35 2002
 *
 *@author  <a href="mailto:kasperp@it-c.dk">Kasper Bøgebjerg Pedersen</a>
 */

public class SimpleCellReferencePair implements CellReferencePair {
  private CellReference cell;
  private ValueStream rest;


  public SimpleCellReferencePair( CellReference cell, ValueStream rest ) {
    this.cell = cell;
    this.rest = rest;
  }


  public CellReference first() {
    return cell;
  }


  public ValueStream rest() {
    return rest;
  }

}
```

```
package edu.it.disk;

/**
 *  SimpleReferenceFactory.java Created: Mon Jun 24 14:00:55 2002
 *
 *@author  <a href="mailto:kasperp@it-c.dk">Kasper Bøgebjerg Pedersen</a>
 */

public class SimpleReferenceFactory implements ReferenceFactory {
  private static ReferenceFactory instance = null;


  private SimpleReferenceFactory() { }


  public ValueReference createValueReference( byte[] ref ) {
    return new SimpleValueReference( ref );
  }


  public ValueReference createValueReference( byte[] ref, Locator loc ) {
    return new SimpleValueReference( ref, loc );
  }


  public CellReference createCellReference( byte[] ref ) {
    return new SimpleCellReference( ref );
  }


  public CellReference createCellReference( byte[] ref, CellLocator loc ) {
    return new SimpleCellReference( ref, loc );
  }


  public static ReferenceFactory getInstance() {
    if ( instance == null ) {
      instance = new SimpleReferenceFactory();
    }
    return instance;
  }
}
```



```
package edu.it.disk;

import edu.it.disk.refserver.*;

import java.util.Arrays;

public class SimpleValueReference implements ValueReference {

  protected byte[] ref;
  protected Locator locator;
  protected ReferenceServer refServer;


  public SimpleValueReference( byte[] ref ) {
    this( ref, null );
  }


  SimpleValueReference( byte[] ref, Locator loc ) {
    if ( ref.length != ValueReference.LENGTH ) {
      throw new IllegalArgumentException( "invalid MD5 sum" );
    }
    this.ref = ref;
    this.locator = loc;
    refServer = ReferenceServer.getInstance();
  }


  public ValueStream get() throws DiskException {
    if ( locator == null ) {
      locator = refServer.lookup( this );
    }
    if ( locator == null ) {
      throw new DiskException( "SimpleValueReference Lookup failed :", DiskException.LOOK
UP );
    }
    ValueStream vs = locator.getValueStream();
    return vs;
  }


  public byte[] toByteArray() {
    return ref;
  }


  public static ValueReference create( byte[] ref ) {
    return new SimpleValueReference( ref );
  }


  public int hashCode() {
    return new String( ref ).hashCode();
  }


  public boolean equals( Object o ) {
    if ( o == null ) {
      return false;
    }
    if ( !( o instanceof SimpleValueReference ) ) {
      return false;
```

```
    }
    SimpleValueReference vsr = (SimpleValueReference) o;
    return Arrays.equals( ref, vsr.ref );
  }
}
```



```
package edu.it.disk;

/**
 *  SimpleValueReferencePair.java Created: Fri May 31 09:39:52 2002
 *
 *@author  <a href="mailto:kasperp@it-c.dk">Kasper Bøgebjerg Pedersen</a>
 */

public class SimpleValueReferencePair implements ValueReferencePair {
  private ValueReference vsr;
  private ValueStream rest;


  public SimpleValueReferencePair( ValueReference vsr, ValueStream rest ) {
    this.vsr = vsr;
    this.rest = rest;
  }


  public ValueReference first() {
    return vsr;
  }


  public ValueStream rest() {
    return rest;
  }
}
```

```
package edu.it.disk;

/**
 *  SimpleValueStream.java Created: Thu May 30 20:38:33 2002
 *
 *@author  <a href="mailto:kasperp@it-c.dk">Kasper Bøgebjerg Pedersen</a>
 */

public class SimpleValueStream implements ValueStream {
  private ValueStream rest;
  private byte[] bytes;
  private ValueReference vsr;
  private CellReference cell;
  private byte type;


  SimpleValueStream() {
    this.bytes = null;
    this.vsr = null;
    this.cell = null;
    this.rest = null;
    type = ValueType.EMPTY;
  }


  SimpleValueStream( byte[] first, ValueStream rest ) throws DiskException {
    if ( rest.getValueType() == ValueType.BYTES ) {
      throw new DiskException( "Init error: Can't create valuestream bytes, bytes",
          DiskException.INIT );
    }
    this.bytes = first;
    this.vsr = null;
    this.cell = cell;
    this.rest = rest;
    type = ValueType.BYTES;
  }


  SimpleValueStream( ValueReference first, ValueStream rest ) throws DiskExcepti
on {
    if ( rest.getValueType() == ValueType.V_REF || rest.getValueType() == ValueT
ype.C_REF ) {
      throw new DiskException( "Init error: Can't create valuestream reference, reference",
          DiskException.INIT );
    }
    this.bytes = null;
    this.vsr = first;
    this.cell = null;
    this.rest = rest;
    type = ValueType.V_REF;
  }


  SimpleValueStream( CellReference first, ValueStream rest ) throws DiskExceptio
n {
    if ( rest.getValueType() == ValueType.V_REF || rest.getValueType() == ValueT
ype.C_REF ) {
      throw new DiskException( "Init error: Can't create valuestream reference, reference",
          DiskException.INIT );
    }
    this.bytes = null;
    this.vsr = null;
```



```
    this.cell = first;
    this.rest = rest;
    type = ValueType.C_REF;
  }


  public byte getValueType() throws DiskException {
    return type;
  }


  public ByteArrayPair getBytes() throws DiskException {
    if ( type != ValueType.BYTES ) {
      throw new DiskException( "Wrong Value type is " + type + " should be " + ValueType.
BYTES,
          DiskException.TYPE );
    }
    return getBytes( bytes.length );
  }


  public ByteArrayPair getBytes( int num ) throws DiskException {
    if ( type != ValueType.BYTES ) {
      throw new DiskException( "Wrong Value type: is " + type + " should be " + ValueType.
BYTES,
          DiskException.TYPE );
    }
    byte[] first = bytes;
    ValueStream tmprest = rest;
    if ( num < bytes.length ) {
      first = new byte[num];
      System.arraycopy( bytes, 0, first, 0, num );

      int restlen = bytes.length - num;
      byte[] brest = new byte[restlen];
      System.arraycopy( bytes, num, brest, 0, restlen );

      tmprest = new SimpleValueStream( brest, this.rest );
    }
    return new SimpleByteArrayPair( first, tmprest );
  }


  public ValueReferencePair getValueReference() throws DiskException {
    if ( type != ValueType.V_REF ) {
      throw new DiskException( "Wrong Value type is " + type + " should be " + ValueType.
V_REF,
          DiskException.TYPE );
    }
    return new SimpleValueReferencePair( vsr, rest );
  }


  public CellReferencePair getCell() throws DiskException {
    if ( type != ValueType.C_REF ) {
      throw new DiskException( "Wrong Value type is " + type + " should be " + ValueType.
C_REF,
          DiskException.TYPE );
    }
    return new SimpleCellReferencePair( cell, rest );
  }
```

```
  public ValueStream skip() throws DiskException {
    return rest;
  }
}
```



```
package edu.it.disk;

/**
 *  SimpleValueStreamFactory.java Created: Thu May 30 20:27:45 2002
 *
 *@author  <a href="mailto:kasperp@it-c.dk">Kasper Bøgebjerg Pedersen</a>
 */

public class SimpleValueStreamFactory implements ValueStreamFactory {
  private static SimpleValueStreamFactory instance;


  private SimpleValueStreamFactory() { }


  public static SimpleValueStreamFactory getInstance() {
    if ( instance == null ) {
      instance = new SimpleValueStreamFactory();
    }
    return instance;
  }


  public ValueStream mkEmpty() {
    return new SimpleValueStream();
  }


  public ValueStream mkByteValueStream( byte[] first, ValueStream rest ) throws
DiskException {
    return new SimpleValueStream( first, rest );
  }


  public ValueStream mkValRefValueStream( ValueReference first, ValueStream rest
 ) throws DiskException {
    return new SimpleValueStream( first, rest );
  }


  public ValueStream mkCellRefValueStream( CellReference cell, ValueStream rest
) throws DiskException {
    return new SimpleValueStream( cell, rest );
  }
}
```

```java
package edu.it.disk;


import java.io.*;
import java.net.*;

public class SocketInputStream extends InputStream {
  private Socket socket;
  private InputStream in;
  private DataOutputStream out;
  private String diskname;
  private int off, len, bytesRead;
  private final byte[] ip;
  private final int port;


  public SocketInputStream( byte[] ip, int port, String diskname, int off, int l
en ) throws IOException {
    this.ip = ip;
    this.port = port;
    this.diskname = diskname;
    this.off = off;
    this.len = len;
  }


  private void connect() throws IOException {
    socket = new Socket( InetAddress.getByAddress( ip ), port );
    in = new BufferedInputStream( socket.getInputStream() );
    out = new DataOutputStream( new BufferedOutputStream( socket.getOutputStream
() ) );
  }


  private void disconnect() throws IOException {
    if ( out != null ) {
      out.close();
    }
    if ( in != null ) {
      in.close();
    }
    if ( socket != null ) {
      socket.close();
    }
  }


  public int read() throws IOException {
    if ( bytesRead == -1 || bytesRead >= len ) {
      return -1;
    }
    connect();
    out.write( GlobalDisk.READ );
    out.writeUTF( diskname );
    out.writeInt( off++ );
    out.writeInt( len );
    out.flush();
    int currByte = in.read();
    disconnect();
    bytesRead++;
    return currByte;
  }
```



```java

  public int read( byte[] b, int off, int len ) throws IOException {
    if ( bytesRead == -1 || bytesRead >= this.len ) {
      //EOF
      return -1;
    }
    if ( len > this.len ) {
      //trying to read more bytes than are in stream
      len = this.len;
    }
    connect();
    out.write( GlobalDisk.READ_LEN );
    out.writeInt( len );
    out.writeUTF( diskname );
    out.writeInt( this.off );
    // move filepointer to current
    out.writeInt( len );
    out.flush();
    bytesRead = in.read( b, off, len );
    disconnect();
    if ( bytesRead != -1 ) {
      this.off += bytesRead;
    }
    return bytesRead;
  }


  public void close() throws IOException {
    disconnect();
  }
}
```

```java
package edu.it.disk;

public interface Value {
  byte[] toByteArray();
}
```



```java
package edu.it.disk;

public interface ValueReference {
  public final static byte LENGTH = 16;


  /**
   *@return  the referenced valuestream
   */
  ValueStream get() throws DiskException;


  byte[] toByteArray();
}
```

```java
package edu.it.disk;

public interface ValueReferencePair {

  /**
   *@return  reference
   */
  ValueReference first();


  /**
   *@return  rest of valuestream.
   */
  ValueStream rest();
}
```



```java
package edu.it.disk;

/**
 *  Values are either a streams of bytes or 16 byte references to other
 *  valuestreams. Cells and valuereferences could have been removed from
 *  interfaces, by returning valuestreams instead of those - e.g. interface
 *  must be modified to interface ValueStream{ byte getType();
 *  ByteArrayPair getBytes(); ByteArrayPair getBytes(int num);
 *  ValueReferencePair getValueReference(); CellPair getCell(); } interface
 *  ValueReferencePair{ ValueStream first(); ValueStream rest(); }
 *  interface CellPair{ UpdateableValueStream first() ValueStream rest(); }
 *  interface UpdateableValueStream extends ValueStream{ ValueStream
 *  set(ValueStream); } As seen the ValueReferencePair interface seems
 *  confusing, as first() and rest bothe return the same type, and as
 *  valuestreamreferences (and thereby their existens) is hidden from the
 *  user of the interface. The CellPair interface implies the need of an
 *  UpdateableValueStream interface, in order to be able to update
 *  valuestream (actually the cell) Thus working with
 *  valuestreamsreferences and cells implies one additionally interface,
 *  but it is much more understandable, what is actually happening. With
 *  the chosen interfaces byte tags can not be used on disk. This is
 *  explained by the bulk operation for getting bytes, e.g. the next value
 *  is a byte sequence with length > 100 and ValueStream.get(100) is called
 *  to get the next 100 bytes. Afterwoods ByteArrayPair.rest() is called in
 *  order to get the rest the valuestream. The "pointer" on disk for this
 *  valuestream will now point to an arbitrary position in a byte sequence,
 *  e.g. it will not point to a byte tag. Thus tags can only be applied to
 *  ValueReferences and Cells (and inlined valuestreams, which are shorter
 *  than their corresponding ValueReference).
 */
public interface ValueStream {

  /**
   *@return  the type of the first value in stream. Types defined in
   *      interface ValueType.
   */
  byte getValueType() throws DiskException;


  /**
   *@throws  Exception if(getType() != BYTES)
   *@return  all bytes until next valuereference (Possibly large, use
   *      method with care) and remainder of valuetream
   */
  ByteArrayPair getBytes() throws DiskException;


  /**
   *@throws  Exception, if(getType() != BYTES)
   *@return  max num bytes and the remainer of the valuestream
   */
  ByteArrayPair getBytes( int num ) throws DiskException;


  /**
   *@throws  Exception, if( !(getType() == VAL_REF || getType() == CELL))
   *@return  ValueReference and remainder of stream
   */
  ValueReferencePair getValueReference() throws DiskException;


  /**
```

```java
   *@throws  Exception, if(getType() != CELL)
   *@return  Cell and remainer of stream
   */
  CellReferencePair getCell() throws DiskException;


  /**
   *  throws away first value either valuereference or all bytes
   *
   *@return  rest of valuestreamf
   */
  ValueStream skip() throws DiskException;
}
```



```java
package edu.it.disk;

public interface ValueStreamFactory {
  ValueStream mkEmpty();


  ValueStream mkByteValueStream( byte[] first, ValueStream rest ) throws DiskExc
eption;


  ValueStream mkValRefValueStream( ValueReference first, ValueStream rest ) thro
ws DiskException;


  ValueStream mkCellRefValueStream( CellReference first, ValueStream rest ) thro
ws DiskException;
}
```

```
package edu.it.disk;

public interface ValueType {
  public final static byte EMPTY = 0x01;
  public final static byte BYTES = 0x02;
  public final static byte V_REF = 0x03;
  public final static byte C_REF = 0x04;
  public final static byte INLIN = 0x05;
}
```



```
package edu.it.disk.refserver;

import edu.it.disk.*;

import edu.it.properties.*;
import edu.it.util.*;
import java.io.IOException;
import java.net.*;

public class GlobalReferenceServer extends ReferenceServer {
  private ReferenceServer localServer;
  private Communicator com;
  private static ReferenceServer instance;


  GlobalReferenceServer( ReferenceServer localServer ) throws IOException, Prope
rtyException {
    this.localServer = localServer;
    XMLStoreProperties props = XMLStoreProperties.getInstance( "xmlstore" );
    this.com = new MulticastCommunicator( InetAddress.getByName( props.getServer
IP() ), props.getValRefServerPort() );
  }


  public Locator lookup( ValueReference vr ) throws DiskException {
    Locator loc = localServer.lookup( vr );
    if ( loc == null ) {
      // lookup globally
      byte[] msg = new byte[ValueReference.LENGTH + 2];
      msg[0] = MapAction.LOOKUP;
      msg[1] = ValueReference.LENGTH;
      // buggy if length > 128
      System.arraycopy( vr.toByteArray(), 0, msg, 2, ValueReference.LENGTH );
      byte[] resp = new byte[Communicator.PAC_SIZE];
      int len = -1;
      try {
        len = com.sendReceive( msg, resp );
      } catch ( IOException e ) {
        throw new DiskException( DiskException.IO, e );
      }
      if ( len != MapAction.NOT_FOUND ) {
        loc = Converter.string2locator( new String( resp, 0, len ) );
        localServer.bind( vr, loc );
        // replicate binding
      }
    }
    return loc;
  }


  public void bind( ValueReference vr, Locator loc ) throws DiskException {
    localServer.bind( vr, loc );
    //bind glocally needed for central server
    /*
     *  byte[] bvr = vr.toByteArray();
     *  byte[] bloc = loc.toString().getBytes();
     *  byte[] msg = new byte[ValueReference.LENGTH + bloc.length + 3];
     *  msg[0] = MapAction.BIND;
     *  msg[1] = ValueReference.LENGTH;
     *  System.arraycopy(bvr, 0, msg, 2, ValueReference.LENGTH);
     *  msg[2 + ValueReference.LENGTH] = (byte)bloc.length;
     *  System.arraycopy(bloc, 0, msg, 3+ValueReference.LENGTH, bloc.length);
```

```java
     *  try{
     *  com.send( msg );
     *  }catch(IOException e){ throw new DiskException(DiskException.IO, e); }
     */
  }


  public static ReferenceServer getInstance() {
    if ( instance == null ) {
      try {
        instance = new GlobalReferenceServer( LocalReferenceServer.getInstance()
 );
      } catch ( IOException e ) {
        System.err.println( "creating global reference server failed " + e );
      } catch ( PropertyException e ) {
        System.err.println( "creating global reference server failed " + e );
      }
    }
    return instance;
  }


  public static void deleteInstance() {
    instance = null;
  }
}
```



```java
package edu.it.disk.refserver;


import edu.it.disk.*;
import edu.it.util.*;
import java.io.IOException;

public class LocalReferenceServer extends ReferenceServer {
  private static LocalReferenceServer instance;
  private PHashtable table;


  private LocalReferenceServer() throws IOException {
    this.table = new PHashtable( "referenceserver.hash" );
  }


  public Locator lookup( ValueReference vsr ) throws DiskException {
    byte[] loc = table.get( vsr.toByteArray() );
    if ( loc == null ) {
      return null;
    }
    return Converter.string2locator( new String( loc ) );
  }


  public void bind( ValueReference vsr, Locator loc ) throws DiskException {
    try {
      table.put( vsr.toByteArray(), loc.toString().getBytes() );
    } catch ( IOException e ) {
      throw new DiskException( DiskException.IO, e );
    }
  }


  public static ReferenceServer getInstance() {
    if ( instance == null ) {
      try {
        instance = new LocalReferenceServer();
      } catch ( IOException e ) {
        System.err.println( "Creating local reference server failed " + e );
      }
    }
    return instance;
  }


  public static LocalReferenceServer getLocalInstance() {
    if ( instance == null ) {
      try {
        instance = new LocalReferenceServer();
      } catch ( IOException e ) {
        System.err.println( "Creating local reference server failed " + e );
      }
    }
    return instance;
  }


  public PHashtable getTable() {
    return table;
  }
```

```java

  public static void deleteInstance() {
    instance = null;
  }

}
```



```java
package edu.it.disk.refserver;

import edu.it.disk.*;

public abstract class ReferenceServer {
  private static ReferenceServer instance;
  private static boolean global;


  public abstract Locator lookup( ValueReference vr ) throws DiskException;


  public abstract void bind( ValueReference vr, Locator locator ) throws DiskExc
eption;


  public static void mkGlobal() {
    global = true;
    instance = null;
  }


  public static ReferenceServer getInstance() {
    if ( instance == null ) {
      if ( global ) {
        instance = GlobalReferenceServer.getInstance();
      } else {
        instance = LocalReferenceServer.getInstance();
      }
    }
    return instance;
  }


  public static void deleteInstance() {
    instance = null;
  }
}
```

```java
package edu.it.dvm;

/**
 *  The Attribute interface represents an attribute in a XML element. This
 *  interface is only used when nodes are initialized or modified. <p>
 *
 *  Created: Wed May 1 15:10:31 2002 </p>
 *
 *@author  <a href="mailto:tejl@it-c.dk">Jesper Tejlgaard Pedersen</a>
 */

public interface Attribute {

  /**
   *  The method returns the value of the attribute
   *
   *@return  the value of the attribute.
   */
  public String getValue();


  /**
   *  The method returns the name of the attribute
   *
   *@return  the name of the attribute
   */
  public String getName();
}
```



```java
package edu.it.dvm;

/**
 *  The ChildNodes interface provides the abstraction of an ordered
 *  collection of nodes. The nodes in the ChildNodes are accessible via an
 *  integral index, starting from 0. <p>
 *
 *  Created: Tue Mar 12 13:58:13 2002</pC
 *
 *@author  <a href="mailto:tejl@it-c.dk">Jesper Tejlgaard Pedersen</a>
 */

public interface ChildNodes {

  /**
   *  The method returns the number of nodes in the list.
   *
   *@return  the number of nodes in the list.
   */
  public int getLength();


  /**
   *  Returns the node from given by the index in the argument.
   *
   *@param  index into the list
   *@return  node at the given index
   *@throws  ArrayIndexOutOfBoundsException
   */
  public Node getNode( int index );
}
// ChildNodes
```

```
package edu.it.dvm;

import java.io.File;
import java.io.IOException;


import javax.xml.parsers.*;
import org.xml.sax.*;

public class DOMUtil {
  //public static org.w3c.dom.Element removeEmptyTextNodes(org.w3c.dom.Element n
ode){return node;}

  public static org.w3c.dom.Node removeEmptyTextNodes( org.w3c.dom.Node node ) {
    if ( node.getNodeType() == org.w3c.dom.Node.ELEMENT_NODE ) {
      int length = node.getChildNodes().getLength();
      for ( int i = 0; i < length; i++ ) {
        org.w3c.dom.Node child = node.getFirstChild();
        switch ( child.getNodeType() ) {
            case org.w3c.dom.Node.ELEMENT_NODE:
            {
              node.removeChild( child );
              node.appendChild( removeEmptyTextNodes( child ) );
              break;
            }
            case org.w3c.dom.Node.TEXT_NODE:
            {
              if ( child.getNodeValue().trim().equals( "" ) ) {
                node.removeChild( child );
              }
              break;
            }
        }
      }
    }
    return node;
  }


  public static org.w3c.dom.Document parseFile( String filename )
       throws DVMException {
    try {
      DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
      factory.setIgnoringElementContentWhitespace( true );
      DocumentBuilder builder = factory.newDocumentBuilder();
      return builder.parse( new File( filename ) );
    } catch ( SAXException e ) {
      throw new DVMException( DVMException.DVM, e );
    } catch ( ParserConfigurationException e ) {
      throw new DVMException( DVMException.DVM, e );
    } catch ( IOException e ) {
      throw new DVMException( DVMException.IO, e );
    }
  }
}
```



```
package edu.it.dvm;

/**
 *  DVMAttribute.java Created: Wed May 1 15:12:11 2002
 *
 *@author  <a href="mailto:tejl@it-c.dk">Jesper Tejlgaard Pedersen</a>
 */


public class DVMAttribute implements Attribute {
  private String name;
  private String value;


  public DVMAttribute( String name, String value ) {
    this.name = name;
    this.value = value;
  }


  public String getValue() {
    return value;
  }


  public String getName() {
    return name;
  }


  public boolean getSpecified() {
    return false;
  }
}
```

```java
package edu.it.dvm;


import edu.it.disk.ValueReference;
import java.io.*;
import java.util.Vector;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

public class DVMBuilder {
  private static SAXParser parser;
  private XMLStore xmlstore = null;


  public DVMBuilder( XMLStore xmlstore ) throws DVMException {
    this.xmlstore = xmlstore;
    System.setProperty( "javax.xml.parsers.SAXParserFactory",
        "org.apache.xerces.jaxp.SAXParserFactoryImpl" );
    try {
      SAXParserFactory factory = SAXParserFactory.newInstance();
      parser = factory.newSAXParser();
    } catch ( ParserConfigurationException e ) {
      throw new DVMException( DVMException.SAX, e );
    } catch ( SAXException e ) {
      throw new DVMException( DVMException.SAX, e );
    }
  }



  /**
   *  Reads an XML document in serialized form and builds a DVM
   *  representation of the document. The return value is the root node of
   *  the DVM representation. The DVM document consists of immutable nodes
   *  and is not saved.
   *
   *@param  in serialized XML document.
   *@return  root node of the DVM representation.
   */
  public static Node parse( InputSource in ) throws DVMException {
    try {
      System.setProperty( "javax.xml.parsers.DocumentBuilderFactory",
          "org.apache.xerces.jaxp.DocumentBuilderFactoryImpl" );
      DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
      factory.setIgnoringElementContentWhitespace( true );
      DocumentBuilder builder = factory.newDocumentBuilder();

      org.w3c.dom.Document doc = builder.parse( in );
      org.w3c.dom.Node root = doc.getDocumentElement();
      root = DOMUtil.removeEmptyTextNodes( root );
      return DVMUtil.unparse( root );
    } catch ( SAXException e ) {
      throw new DVMException( DVMException.SAX, e );
    } catch ( ParserConfigurationException e ) {
      throw new DVMException( DVMException.PARSERCONFIGURATION, e );
    } catch ( IOException e ) {
      throw new DVMException( DVMException.IO, e );
    }


    //String keyword = nInsert.getFirstChild().getFirstChild().getNodeValue();
```



```java
    /*
     *  DOMUtil.parseFile(filename);
     *  org.w3c.dom.Node root = doc.getDocumentElement();
     *  root = DOMUtil.removeEmptyTextNodes(root);
     *  return unparse(root);
     */
    //return null;
  }



  /**
   *  Reads an XML document in serialized form and builds a DVM
   *  representation of the document. The return value is the root node of
   *  the DVM representation. The DVM document consists of immutable nodes
   *  and is not saved.
   *
   *@param  fileName file name denoting the serialized XML document.
   *@return  root node of the DVM representation.
   */
  public static Node parse( String fileName ) throws DVMException {
    org.w3c.dom.Document doc = DOMUtil.parseFile( fileName );
    org.w3c.dom.Node root = doc.getDocumentElement();
    root = DOMUtil.removeEmptyTextNodes( root );
    return DVMUtil.unparse( root );
  }



  /**
   *  Unparses an XML document in serialized format to a persistent
   *  representation in the Document Value Model (DVM). The DVM document
   *  consists of immutable nodes and is saved during unparsing. A
   *  reference to the document's root node is returned.
   *
   *@param  in serialized XML document
   *@return  value reference to the root node of the DVM representation .
   */
  public ValueReference unparse( InputSource in ) throws DVMException {
    XMLHandler handler = new XMLHandler( xmlstore );
    try {
      parser.parse( in, handler );
    } catch ( SAXException e ) {
      throw new DVMException( DVMException.SAX, e );
    } catch ( IOException e ) {
      throw new DVMException( DVMException.IO, e );
    }
    return handler.getReference();
  }



  /**
   *  Unparses an XML document in serialized format to a persistent
   *  representation in the Document Value Model (DVM). The DVM document
   *  consists of immutable nodes and is saved during unparsing. A
   *  reference to the document's root node is returned.
   *
   *@param  fileName serialized XML document
   *@return  value reference to the root node of the DVM representation .
   */
  public ValueReference unparse( String fileName ) throws DVMException {
    XMLHandler handler = new XMLHandler( xmlstore );
    try {
      parser.parse( fileName, handler );
```

```
    } catch ( SAXException e ) {
      throw new DVMException( DVMException.SAX, e );
    } catch ( IOException e ) {
      throw new DVMException( DVMException.IO, e );
    }
    return handler.getReference();
  }


  private static class XMLHandler extends DefaultHandler {
    public static int DEFAULT_STACKSIZE = 20;
    private XMLStore xmlstore;
    private XMLStoreFactory factory;
    private int elements;
    private Element[] elementStack;


    public XMLHandler( XMLStore xmlstore ) throws DVMException {
      this( xmlstore, "DVM", DEFAULT_STACKSIZE );
    }


    public XMLHandler( XMLStore xmlstore, String propsId, int stackSize )
         throws DVMException {
      this.xmlstore = xmlstore;
      factory = XMLStoreFactory.getInstance( propsId );
      elementStack = new Element[stackSize];
      elements = 0;
      elementStack[elements++] = new Element( "document" );
      // creating document
    }


    public void startElement( String uri, String localName, String qName,
        Attributes attributes ) {
      if ( qName.trim().length() > 0 ) {
        Attribute[] attrs = unparseAttributes( attributes );
        elementStack[elements++] = new Element( qName, attrs );
      }
    }


    public void endElement( String uri, String localName, String qName ) {
      try {
        if ( qName.trim().length() == 0 ) {
          return;
        }
        ValueReference ref = null;
        // pop element from stack
        Element element = elementStack[--elements];
        try {
          // create XMLStore node
          ChildNodes children = createProxies( element.references );
          Node n = factory.createElementNode( element.name, children,
              element.attributes );
          // save node
          ref = xmlstore.save( n );
        } catch ( DVMException e ) {
          System.out.println( "DVMException in endElement " + e.getMessage() );
        }
        // put reference in top element
        elementStack[elements - 1].references.add( ref );
```



```
      } catch ( RuntimeException e ) {
        System.out.println( "caught exception in endElement" );
        System.out.println( "element name " + qName );
        System.out.println( "exception message = " + e.getMessage() );
        /*
         *  System.out.println(elementStack[elements-1].name);
         *  Vector refs = elementStack[elements-1].references;
         *  try{
         *  Node n = xmlstore.load((ValueReference)refs.get(refs.size()-2));
         *  System.out.println(n.getNodeValue());
         *  System.out.println(n.getChildNodes().getNode(0).getNodeValue());
         *  System.out.println(n.getChildNodes().getNode(0).getChildNodes().getN
ode(0).getNodeValue());
         *  }catch(Exception ed){
         *  System.out.println("fucking exceptions");
         *  }
         */
        e.printStackTrace();
        throw e;
      }
    }


    public void characters( char[] ch, int start, int length ) {
      // create CharDataNode
      try {
        String chardata = new String( ch, start, length );
        if ( chardata.trim().equals( "" ) ) {
          return;
        }
        Node n = factory.createCharDataNode( chardata );
        // save node
        ValueReference ref = null;
        try {
          ref = xmlstore.save( n );
        } catch ( DVMException e ) {
          System.out.println( "DVMExecption in characters " + e.getMessage() );
        }
        // add reference to owning element
        elementStack[elements - 1].references.add( ref );
      } catch ( Exception e ) {
        System.out.println( new String( ch, start, length ) );
        e.getMessage();
      }
    }


    public ValueReference getReference() {
      Object o = elementStack[elements - 1].references.firstElement();
      return (ValueReference) o;
    }


    private Attribute[] unparseAttributes( Attributes attributes ) {
      Attribute[] attrs = new Attribute[attributes.getLength()];
      for ( int i = 0; i < attrs.length; i++ ) {
        attrs[i] = factory.createAttribute( attributes.getQName( i ),
            attributes.getValue( i ) );
      }
      return attrs;
    }
```

```java
    private ChildNodes createProxies( Vector references ) throws DVMException {
      Node[] children = new Node[references.size()];
      for ( int i = 0; i < children.length; i++ ) {
        children[i] = xmlstore.load( (ValueReference) references.get( i ) );
      }
      return factory.createChildNodes( children );
    }


    private static class Element {
      public String name;
      public Attribute[] attributes;
      public Vector references;


      public Element( String name ) {
        this( name, null );
      }


      public Element( String name, Attribute[] attrs ) {
        this.name = name;
        this.attributes = attrs;
        this.references = new Vector();
      }
    }

  }
}
// XMLFileReader
```



```java
package edu.it.dvm;

/**
 *  DVMChildNodes.java Created: Tue Mar 12 22:39:43 2002
 *
 *@author  <a href="mailto:tejl@it-c.dk">Jesper Tejlgaard Pedersen</a> 1.0
 */

public class DVMChildNodes implements ChildNodes {
  private Node[] nodeList;


  /**
   *  The construktor takes an array containing the nodes, neded to create
   *  at list. The sequence of the nodes will be the same in the list as in
   *  the array. <p>
   *
   *  A NullPointerException is throwned in two cases.
   *  <ul>
   *    <li> If the param is null.</li>
   *    <li> if one or more spaces in the array contains a null value.</li>
   *
   *  </ul>
   *  </p>
   *
   *@param  nodes
   */
  DVMChildNodes( Node[] nodes ) {
    // copying array in order to make ChildNodes immutable
    nodeList = new Node[nodes.length];
    for ( int i = 0; i < nodeList.length; i++ ) {
      if ( nodes[i] == null ) {
        throw new NullPointerException();
      }
      nodeList[i] = nodes[i];
    }
  }


  /**
   *@return  Length of list
   */
  public int getLength() {
    return nodeList.length;
  }


  /**
   *@return  The Node in the list with the given index.
   */
  public Node getNode( int index ) {
    try {
      return nodeList[index];
    } catch ( ArrayIndexOutOfBoundsException e ) {
      throw new ArrayIndexOutOfBoundsException( "index=" + index );
    }
  }

}
// DVMChildNodes
```

```java
package edu.it.dvm;

/**
 *  DVMException.java Created: Tue May 14 18:11:06 2002
 *
 *@author  <a href="mailto:tejl@it-c.dk">Jesper Tejlgaard Pedersen</a>
 */

public class DVMException extends Exception {
  private byte type;

  public static byte DVM = 0x00;
  public static byte IO = 0x01;
  public static byte INSTANTIATION = 0x02;
  public static byte DISK = 0x03;
  public static byte NET = 0x04;
  public static byte LOOKUP = 0x05;
  public static byte EXT_NOT_ROOT = 0x06;
  public static byte CLASSNOTFOUND = 0x07;
  public static byte ILLEGAL_ACCESS = 0x08;
  public static byte SAX = 0x09;
  public static byte PARSERCONFIGURATION = 0x0A;
  public static byte PROPS = 0x0B;


  public DVMException( byte type, String msg ) {
    super( msg );
    init( type );
  }


  public DVMException( byte type, String msg, Exception e ) {
    super( msg, e );
    init( type );
  }


  public DVMException( byte type, Exception e ) {
    super( e );
    init( type );
  }


  public DVMException( byte type ) {
    init( type );
  }


  private void init( byte type ) {
    this.type = type;
  }


  public byte getType() {
    return type;
  }
}
```



```java
package edu.it.dvm;


import edu.it.disk.DiskException;
import java.util.Hashtable;

public class DVMImmNode implements Node {
  private byte type;
  private String value;
  private ChildNodes children;
  private Hashtable attributes;


  public DVMImmNode( byte type, String value, ChildNodes children,
      Attribute[] attrs ) {
    if ( value == "" ) {
      throw new IllegalArgumentException( "Empty String not a valid argument according to xml st
andards" );
    }
    this.type = type;
    this.value = value;
    this.children = children;

    if ( type != Node.CHARDATA ) {
      attributes = new Hashtable();
      for ( int i = 0; i < attrs.length; i++ ) {
        attributes.put( attrs[i].getName(), attrs[i].getValue() );
      }
    }
  }


  public boolean isMutable() throws DVMException {
    return false;
  }


  public byte getType() throws DVMException {
    return type;
  }


  public String getNodeValue() throws DVMException {
    return value;
  }


  public ChildNodes getChildNodes() throws DVMException {
    return children;
  }


  public String getAttribute( String attrName ) throws DVMException {
    if ( getType() == Node.CHARDATA ) {
      return null;
    }
    return (String) attributes.get( attrName );
  }


  public String[] getAttributeNames() throws DVMException {
    if ( getType() == Node.CHARDATA ) {
```

```
      return null;
    }
    Object[] oArr = attributes.keySet().toArray();
    String[] names = new String[oArr.length];
    for ( int i = 0; i < oArr.length; i++ ) {
      names[i] = (String) oArr[i];
    }
    return names;
  }


  public void accept( Visitor v ) throws DVMException {
    v.visit( this );
  }


  public String toString() {
    try {
      String result = "";
      if ( getType() == Node.ELEMENT ) {
        result += "<" + getNodeValue();
        String[] attrs = getAttributeNames();
        for ( int i = 0; i < attrs.length; i++ ) {
          result += " " + attrs[i] + "=\"" + getAttribute( attrs[i] ) + "\"";
        }
        ChildNodes children = getChildNodes();
        if ( children.getLength() > 0 ) {
          result += ">\n";
          for ( int i = 0; i < children.getLength(); i++ ) {
            result += children.getNode( i ) + "\n";
          }
          result += "</" + getNodeValue() + ">";
        } else {
          result += "/>";
        }
      }
      if ( getType() == Node.CHARDATA ) {
        result += getNodeValue();
      }
      return result;
    } catch ( DVMException e ) {
      String msg = "caught DVMException with type = " + e.getType();
      msg += " and message = " + e.getMessage();
      return msg;
    }
  }
}
```



```
package edu.it.dvm;


import edu.it.disk.*;
import java.util.Hashtable;
import java.util.Vector;

public class DVMImmNodeProxy implements Node {
  protected ValueReference vStreamRef;
  protected ValueStream vStream;
  protected Disk disk;
  protected Node node;

  protected String value;
  protected byte[] types;
  // 0 mutable/immtable , 1 chardata /element
  protected ChildNodes children;
  protected Hashtable attrs;

  private static ProxyQueue queue = new ProxyQueue();


  public DVMImmNodeProxy( Disk disk, ValueReference vStreamRef )
       throws DVMException {
    this.disk = disk;
    this.node = null;
    this.vStreamRef = vStreamRef;
    try {
      this.vStream = disk.load( vStreamRef );
    } catch ( DiskException e ) {
      throw new DVMException( DVMException.DISK, e );
    }
  }


  public boolean isMutable() throws DVMException {
    return false;
  }


  public byte getType() throws DVMException {
    return getTypes()[1];
  }


  public String getNodeValue() throws DVMException {
    if ( value == null ) {
      activate();
      value = DVMNodeLoader.getValue( vStream );
    }
    return value;
  }


  public ChildNodes getChildNodes() throws DVMException {
    if ( children == null ) {
      activate();
      children = DVMNodeLoader.getChildren( disk, vStream );
    }
    return children;
  }
```

```java
public String getAttribute( String attrName ) throws DVMException {
  if ( attrs == null ) {
    activate();
    attrs = DVMNodeLoader.getAttributes( disk, vStream );
  }
  return (String) attrs.get( attrName );
}


public String[] getAttributeNames() throws DVMException {
  if ( getType() == Node.CHARDATA ) {
    return null;
  }
  // load attributes:
  getAttribute( "" );
  // extract and return attribute names
  Object[] oArr = attrs.keySet().toArray();
  String[] names = new String[oArr.length];
  for ( int i = 0; i < oArr.length; i++ ) {
    names[i] = (String) oArr[i];
  }
  return names;
}


public byte[] getTypes() throws DVMException {
  if ( types == null ) {
    types = DVMNodeLoader.getTypes( vStream );
  }
  return types;
}


public String toString() {
  try {
    return getNode().toString();
  } catch ( DVMException e ) {
    return "DVMException caught in DVMNodeProxy";
  }
}


public void accept( Visitor visitor ) throws DVMException {
  visitor.visit( this );
}


private Node getNode() throws DVMException {
  if ( node == null ) {
    XMLStoreFactory factory = XMLStoreFactory.getInstance();
    byte type = getType();
    if ( type == Node.CHARDATA ) {
      node = factory.createCharDataNode( getNodeValue() );
    } else if ( type == Node.ELEMENT ) {
      node = factory.createElementNode( getNodeValue(), getChildNodes(),
          DVMUtil.getAttributes( this ) );
    }
  }
  return node;
}
```



```java
  private void activate() {
    if ( children == null && attrs == null && value == null ) {
      queue.add( this );
    }
  }


  private void inactivate() {
    children = null;
    value = null;
    attrs = null;
    node = null;
  }


  ValueReference getValueReference() {
    return vStreamRef;
  }


  static class ProxyQueue {
    private DVMImmNodeProxy[] queue;
    private int size;


    ProxyQueue() {
      this.queue = new DVMImmNodeProxy[1000];
      size = 0;
    }


    public void add( DVMImmNodeProxy element ) {
      if ( size >= queue.length ) {
        size = 0;
      }
      if ( queue[size] != null ) {
        //System.out.println("inactivate()");
        queue[size].inactivate();
      }
      queue[size++] = element;
    }
  }
}
```

```java
package edu.it.dvm;

import edu.it.disk.CellReference;


import edu.it.disk.Disk;
import edu.it.disk.DiskException;
import edu.it.disk.ValueReference;
import java.util.Hashtable;

public class DVMMutNode implements MutNode {
  private CellReference cell;
  private Disk disk;
  private Node node;


  public DVMMutNode( Node node ) {
    this( node, null, null );
  }


  public DVMMutNode( Disk disk, CellReference cell ) {
    this( null, disk, cell );
  }


  private DVMMutNode( Node node, Disk disk, CellReference cell ) {
    this.node = node;
    this.cell = cell;
    this.disk = disk;
  }


  public boolean isMutable() throws DVMException {
    return true;
  }


  public byte getType() throws DVMException {
    return getNodeState().getType();
  }


  public String getNodeValue() throws DVMException {
    return getNodeState().getNodeValue();
  }


  public ChildNodes getChildNodes() throws DVMException {
    return getNodeState().getChildNodes();
  }


  public String getAttribute( String attrName ) throws DVMException {
    return getNodeState().getAttribute( attrName );
  }


  public String[] getAttributeNames() throws DVMException {
    return getNodeState().getAttributeNames();
  }
```



```java
  public Node getNodeState() throws DVMException {
    if ( getCellReference() != null ) {
      return DVMNodeLoader.loadNode( disk, cell );
    }
    return node;
  }


  public void setNodeState( Node node ) throws DVMException {
    if ( getCellReference() != null ) {
      SaveVisitor visitor = new SaveVisitor( disk );
      node.accept( visitor );
      try {
        cell.set( visitor.getValueReference() );
      } catch ( DiskException e ) {
        throw new DVMException( DVMException.DISK, e );
      }
      return;
    }
    this.node = node;
  }


  public void accept( Visitor v ) throws DVMException {
    v.visit( this );
  }


  CellReference getCellReference() {
    return cell;
  }


  void setCell( CellReference cell ) {
    this.cell = cell;
  }


  void setDisk( Disk disk ) {
    this.disk = disk;
  }


  public String toString() {
    try {
      return getNodeState().toString();
    } catch ( DVMException e ) {
      String msg = "caught DVMException with type = " + e.getType();
      msg += " and message = " + e.getMessage();
      return msg;
    }
  }
}
```

```java
package edu.it.dvm;


import edu.it.disk.*;
import java.util.Hashtable;
import java.util.Vector;

public class DVMNodeLoader {
  public final static byte IMMUTABLE = 0x00;
  public final static byte MUTABLE = 0x01;


  public static Node loadNode( Disk disk, ValueReference vStreamRef )
       throws DVMException {
    try {
      XMLStoreFactory factory = XMLStoreFactory.getInstance( "DVM" );
      ValueStream vStream = disk.load( vStreamRef );

      if ( vStream.getValueType() == ValueType.V_REF
           || vStream.getValueType() == ValueType.C_REF
           || vStream.getValueType() == ValueType.INLIN ) {
        String msg = "Miss typed first element in initialized ValueStream – first element is a value reference.
";
        throw new DVMException( DVMException.DISK, msg );
      }

      if ( vStream.getValueType() == ValueType.EMPTY ) {
        String msg = "ValueStream is empty";
        throw new DVMException( DVMException.DISK, msg );
      }

      // First element is always a BYTESTREAM in order to follow the
      // definition of ValueStream as being VS::= BS(L|BS)*. The first
      // byte tells the type of the node being loaded.
      ByteArrayPair bArrPair = vStream.getBytes( 2 );

      boolean immutable = ( bArrPair.first()[0] == IMMUTABLE );

      if ( bArrPair.first()[1] == Node.CHARDATA ) {
        vStream = bArrPair.rest();
        bArrPair = vStream.getBytes();
        if ( immutable ) {
          return factory.createCharDataNode( new String( bArrPair.first() ) );
        }
        return factory.createMutCharDataNode( new String( bArrPair.first() ) );
      }
      vStream = bArrPair.rest();
      return loadElement( disk, vStream, immutable );
    } catch ( DiskException e ) {
      throw new DVMException( DVMException.DISK, e );
    }
  }


  public static byte[] getTypes( ValueStream vStream ) throws DVMException {
    try {
      ByteArrayPair bArrPair = vStream.getBytes( 2 );
      return bArrPair.first();
    } catch ( DiskException e ) {
      throw new DVMException( DVMException.DISK, e );
    }
  }
```



```java
  public static String getValue( ValueStream vStream ) throws DVMException {
    try {
      ByteArrayPair bArrPair = vStream.getBytes( 2 );
      if ( bArrPair.first()[1] == Node.CHARDATA ) {
        vStream = bArrPair.rest();
        bArrPair = vStream.getBytes();
        return new String( bArrPair.first() );
      }
      vStream = bArrPair.rest();
      bArrPair = vStream.getBytes();
      // name is always first element (attributes are the remaining elements)
      return ( new String( bArrPair.first() ) ).split( " " )[0];
    } catch ( DiskException e ) {
      throw new DVMException( DVMException.DISK, e );
    }
  }


  public static Hashtable getAttributes( Disk disk, ValueStream vStream )
       throws DVMException {
    try {
      ByteArrayPair bArrPair = vStream.getBytes( 2 );
      if ( bArrPair.first()[1] == Node.CHARDATA ) {
        return new Hashtable();
      }
      vStream = bArrPair.rest();
      // get element name and attributes
      bArrPair = vStream.getBytes();
      String sValue = new String( bArrPair.first() );
      String[] sArr = sValue.split( " " );
      return loadAttributes( sArr );
    } catch ( DiskException e ) {
      throw new DVMException( DVMException.DISK, e );
    }
  }


  public static ChildNodes getChildren( Disk disk, ValueStream vStream ) throws
DVMException {
    try {
      ByteArrayPair bArrPair = vStream.getBytes( 2 );
      if ( bArrPair.first()[1] == Node.CHARDATA ) {
        return XMLStoreFactory.getInstance( "DVM" ).createChildNodes( new Node[
0] );
      }
      vStream = bArrPair.rest();
      vStream = vStream.skip();
      // skib bytes (element name)
      return loadChildren( disk, vStream );
    } catch ( DiskException e ) {
      throw new DVMException( DVMException.DISK, e );
    }
  }


  protected static ChildNodes loadChildren( Disk disk, ValueStream vStream )
       throws DVMException {
    try {
      Vector children = new Vector();
```

```java
    while ( vStream.getValueType() != ValueType.EMPTY ) {
      if ( vStream.getValueType() == ValueType.V_REF ) {
        ValueReferencePair rPair = vStream.getValueReference();
        children.add( new DVMImmNodeProxy( disk, rPair.first() ) );
        vStream = rPair.rest();
      } else if ( vStream.getValueType() == ValueType.C_REF ) {
        CellReferencePair cPair = vStream.getCell();
        children.add( new DVMMutNode( disk, cPair.first() ) );
        vStream = cPair.rest();
      } else if ( vStream.getValueType() == ValueType.INLIN ) {
        ByteArrayPair bArrPair = vStream.getBytes();
        children.add( loadInlined( bArrPair ) );
        vStream = bArrPair.rest();
      } else if ( vStream.getValueType() == ValueType.BYTES ) {
        ByteArrayPair bArr = vStream.getBytes();
        if ( bArr.first().length != 0 ) {
          String msg = "Miss formed ValueStream.";
          msg += " Bytes can not occur in between valuestream references.";
          throw new DVMException( DVMException.DISK, msg );
        }
        vStream = bArr.rest();
      }
    }

    Node[] nArr = new Node[children.size()];
    for ( int i = 0; i < nArr.length; i++ ) {
      nArr[i] = (Node) children.get( i );
    }
    return XMLStoreFactory.getInstance( "DVM" ).createChildNodes( nArr );
  } catch ( DiskException e ) {
    throw new DVMException( DVMException.DISK, e );
  }
}


protected static Node loadInlined( ByteArrayPair bArrPair ) {
  return null;
}


protected static Node loadElement( Disk disk, ValueStream vStream,
    boolean immutable ) throws DVMException {
  try {
    ByteArrayPair bArrPair = vStream.getBytes();
    String sValue = new String( bArrPair.first() );

    String[] sArr = sValue.split( " " );
    Attribute[] attrs = extractAttributes( sArr );
    XMLStoreFactory factory = XMLStoreFactory.getInstance( "DVM" );
    ChildNodes list = loadChildren( disk, bArrPair.rest() );
    if ( immutable ) {
      return factory.createElementNode( sArr[0], list, attrs );
    }
    return factory.createMutElementNode( sArr[0], list, attrs );
  } catch ( DiskException e ) {
    throw new DVMException( DVMException.DISK, e );
  }
}


/*
 *  protected static Node loadMutableElement(Disk disk, ValueStream vStream){
```



```java
   *  try{
   *  ByteArrayPair bArrPair = vStream.getBytes();
   *  String sValue = new String(bArrPair.first());
   *  String[] sArr = sValue.split(" ");
   *  Attribute[] attrs = extractAttributes(sArr);
   *  XMLStoreFactory factory = XMLStoreFactory.getInstance("DVM");
   *  ChildNodes list = loadChildren(disk, bArrPair.rest());
   *  return factory.createMutElementNode(sArr[0], list, attrs);
   *  }catch(DiskException e){
   *  throw new DVMException(DVMException.DISK, e);
   *  }
   *  }
   */
  protected static Attribute[] extractAttributes( String[] sArr )
      throws DVMException {

    Attribute[] attrs = new Attribute[sArr.length - 1];
    XMLStoreFactory factory = XMLStoreFactory.getInstance( "DVM" );
    for ( int i = 0; i < attrs.length; i++ ) {
      String[] attr = sArr[i + 1].split( "=" );
      if ( attr.length != 2 ) {
        throw new DVMException( DVMException.DISK,
            "wrong number of attribute arguments" );
      }
      attrs[i] = factory.createAttribute( attr[0], attr[1] );
    }
    return attrs;
  }


  protected static Hashtable loadAttributes( String[] sArr )
      throws DVMException {

    Hashtable attrs = new Hashtable();
    for ( int i = 0; i < ( sArr.length - 1 ); i++ ) {
      String[] attr = sArr[i + 1].split( "=" );
      if ( attr.length != 2 ) {
        throw new DVMException( DVMException.DISK,
            "wrong number of attribute arguments" );
      }
      attrs.put( attr[0], attr[1] );
    }
    return attrs;
  }

}
```

```java
package edu.it.dvm;


import java.util.Arrays;

public class DVMUtil {
  private static XMLStoreFactory factory;

  static {
    try {
      initialize( "DVM" );
    } catch ( DVMException e ) {}
  }


  /**
   *  This method is used for specifying which type of XMLStoreFactory
   *  DVMUtil is to use. If the method is not invoked the default property
   *  file id is used.
   *
   *@param  propsId property file id
   */
  public static void initialize( String propsId ) throws DVMException {
    factory = XMLStoreFactory.getInstance( propsId );
  }


  /**
   *  The method is used for creating new immutable element nodes. It
   *  extends the XMLStoreFactory functionality for creating nodes.
   *
   *@param  tagName tagName of new element
   *@param  children children of new element
   *@return  New element node.
   */
  public static Node createElement( String tagName, Node[] children ) {
    ChildNodes childNodes = factory.createChildNodes( children );
    return factory.createElementNode( tagName, childNodes );
  }


  /**
   *  The method extends the XMLStoreFactory functionality for creating
   *  nodes. It is used for creating a new immutable element nodes and a
   *  new immutable node representing character data. The last is made a
   *  child of the element node.
   *
   *@param  tagName tagName of new element
   *@param  data character data of the elements child node.
   *@return  New element node.
   */
  public static Node createElement( String tagName, String data ) {
    return factory.createElementNode( tagName,
        factory.createCharDataNode( data ) );
  }


  /**
   *  The method returns a child of an element node. The child is specified
   *  by its index.
   *
   *@param  node Node
```



```java
   *@param  index child index
   *@return  Child node.
   */
  public static Node getChild( Node node, int index ) throws DVMException {
    return node.getChildNodes().getNode( index );
  }


  /**
   *  The method is used for replacing a child of an element node. The
   *  replacement is done by constructing and returning a new element node,
   *  which only differs from the original element node in the replaced
   *  child.
   *
   *@param  node element node.
   *@param  index index of the child to replace.
   *@param  child new child node.
   *@return  new element node.
   */
  public static Node replaceChild( Node node, int index, Node child )
       throws DVMException {
    ChildNodes childNodes = node.getChildNodes();
    Node[] children = new Node[childNodes.getLength()];
    for ( int i = 0; i < children.length; i++ ) {
      children[i] = childNodes.getNode( i );
    }
    children[index] = child;

    ChildNodes newChildNodes = factory.createChildNodes( children );
    return factory.createElementNode( node.getNodeValue(), newChildNodes,
        getAttributes( node ) );
  }


  /**
   *  The method is used for replacing a child in an instance of
   *  ChildNodes. The replacement is done by constructing and returning a
   *  new ChildNodes instance node, which only differs from the original
   *  instance in the replaced child.
   *
   *@param  childnodes an instance of ChildNodes
   *@param  index index of the child to replace.
   *@param  child new child node.
   *@return  new instance of ChildNodes
   */
  public static ChildNodes replaceChild( ChildNodes childNodes, int index,
      Node child ) {
    Node[] children = new Node[childNodes.getLength()];
    for ( int i = 0; i < children.length; i++ ) {
      children[i] = childNodes.getNode( i );
    }
    children[index] = child;
    return factory.createChildNodes( children );
  }


  /**
   *  The method is used for adding a child to an element node. This is
   *  done by constructing and returning a new element node, which only
   *  differs from the original element node in the added child.
   *
   *@param  node element node.
```

```java
   *@param  index index at which new child is inserted.
   *@param  child new child node.
   *@return  new element node.
   */
  public static Node insertChild( Node n, int index, Node child )
       throws DVMException {
    ChildNodes childNodes = n.getChildNodes();
    Node[] children = new Node[childNodes.getLength() + 1];
    for ( int i = 0; i < index; i++ ) {
      children[i] = childNodes.getNode( i );
    }
    children[index] = child;
    for ( int i = index + 1; i < children.length; i++ ) {
      children[i] = childNodes.getNode( i - 1 );
    }
    ChildNodes newChildNodes = factory.createChildNodes( children );
    return factory.createElementNode( n.getNodeValue(), newChildNodes,
        getAttributes( n ) );
  }


  /**
   *  The method is used for removing a child from an element node. This is
   *  done by constructing and returning a new element node, which only
   *  differs from the original element node in the removed child.
   *
   *@param  node element node.
   *@param  index index specifying the child to remove
   *@return  new element node.
   */
  public static Node removeChild( Node n, int index )
       throws DVMException {
    ChildNodes childNodes = n.getChildNodes();
    Node[] children = new Node[childNodes.getLength() - 1];
    for ( int i = 0; i < index; i++ ) {
      children[i] = childNodes.getNode( i );
    }
    for ( int i = index; i < children.length; i++ ) {
      children[i] = childNodes.getNode( i + 1 );
    }
    ChildNodes newChildNodes = factory.createChildNodes( children );
    return factory.createElementNode( n.getNodeValue(), newChildNodes,
        getAttributes( n ) );
  }


  /**
   *  The method is used fore creating a new instance of ChildNodes, which
   *  is a sub range of existing ChildNodes instance. The new child nodes
   *  in the new instance is specified by a start index in the original
   *  ChildNodes instance and the number of child nodes to use.
   *
   *@param  childnodes an instance of ChildNodes to create sub range from
   *@param  start index of the first child to.
   *@param  length number of child nodes to share.
   *@return  new instance of ChildNodes
   */
  public static ChildNodes subChildNodes( ChildNodes childNodes, int start,
      int length ) {
    Node[] children = new Node[length];
    for ( int i = 0; i < length; i++ ) {
      children[i] = childNodes.getNode( i + start );
```



```java
    }
    return factory.createChildNodes( children );
  }


  public static Attribute getAttribute( Node n, String attr ) throws DVMExceptio
n {
    return factory.createAttribute( attr, n.getAttribute( attr ) );
  }


  public static Attribute[] getAttributes( Node n ) throws DVMException {
    String[] names = n.getAttributeNames();
    Attribute[] attrs = new Attribute[names.length];
    for ( int i = 0; i < names.length; i++ ) {
      attrs[i] = factory.createAttribute( names[i], n.getAttribute( names[i] ) )
;
    }
    return attrs;
  }


  public static Attribute[] extractAttributes( org.w3c.dom.Node domNode )
       throws DVMException {
    org.w3c.dom.NamedNodeMap attrMap = domNode.getAttributes();
    Attribute[] attrs = new Attribute[attrMap.getLength()];
    for ( int i = 0; i < attrs.length; i++ ) {
      org.w3c.dom.Node attr = attrMap.item( i );
      attrs[i] = factory.createAttribute( attr.getNodeName(),
          attr.getNodeValue() );
    }
    return attrs;
  }


  /**
   *  Converts a DOM representation of an XML tree to a DVM representation.
   *  The DVM representation only contains immutable nodes.
   *
   *@param  domNode the DOM node representing an XML tree.
   *@return  root node of created DVM representation.
   *@throws  DVMException
   */
  public static Node unparse( org.w3c.dom.Node domNode ) throws DVMException {
    switch ( domNode.getNodeType() ) {
        case org.w3c.dom.Node.TEXT_NODE:
        {
          return factory.createCharDataNode( domNode.getNodeValue() );
        }
        case org.w3c.dom.Node.ELEMENT_NODE:
        {
          org.w3c.dom.NamedNodeMap domAttrs = domNode.getAttributes();
          Attribute[] attr = new Attribute[domAttrs.getLength()];
          for ( int i = 0; i < attr.length; i++ ) {
            org.w3c.dom.Node att = domAttrs.item( i );
            attr[i] = factory.createAttribute( att.getNodeName(), att.getNodeVal
ue() );
          }

          org.w3c.dom.NodeList domChildren = domNode.getChildNodes();
          Node[] dvmChildren = new Node[domChildren.getLength()];
          for ( int i = 0; i < dvmChildren.length; i++ ) {
```

```
            dvmChildren[i] = unparse( domChildren.item( i ) );
          }
          ChildNodes childNodes = factory.createChildNodes( dvmChildren );
          return factory.createElementNode( domNode.getNodeName(), childNodes, a
ttr );
        }
        default:
        {
          throw new DVMException( DVMException.DVM,
              "type " + domNode.getNodeType() + " not supported" );
        }
      }
    }


  /**
   *  The method checks equality of two nodes. In order to be equal two
   *  nodes must have the same type. For character data nodes, the
   *  character data's (i.e. node values) must also be equal. For element
   *  nodes equality is checked by
   *  <ul>
   *    <li> equality of the tag names (i.e. node values).</li>
   *    <li> equality of the attributes. The same attributes must occur
   *    with in the elements, and each attribute must have the same value
   *    within the elements. </li>
   *    <li> equality of child nodes. Child nodes must occur with the same
   *    order in the elements. Their equality is checked recursively. </li>
   *
   *  </ul>
   *
   *
   *@param  node1 first node to compare
   *@param  node2 second node to compare
   *@return  true if equal, false otherwise
   */
  public static boolean equals( Node node1, Node node2 ) throws DVMException {
    // making sure that state of mutable nodes is consistent
    //if(node1.isMutable()) node1 = ((MutNode)node1).getNodeState();
    //if(node2.isMutable()) node2 = ((MutNode)node2).getNodeState();

    boolean result = ( node1.getType() == node2.getType() );
    if ( !result ) {
      return false;
    }
    if ( !node1.getNodeValue().equals( node2.getNodeValue() ) ) {
      System.out.println( "node1 = " + node1.getNodeValue() );
      System.out.println( "node2 = " + node2.getNodeValue() );

      return false;
    }
    if ( node1.getType() == Node.ELEMENT ) {
      String[] attr1 = node1.getAttributeNames();
      for ( int i = 0; i < attr1.length; i++ ) {
        // check that the same attributes exists
        String attr2val = node2.getAttribute( attr1[i] );
        if ( attr2val == null ) {
          return false;
        }
        // check equality of attribute values
        if ( !node1.getAttribute( attr1[i] ).equals( attr2val ) ) {
          return false;
        }
```



```
      }
      ChildNodes children1 = node1.getChildNodes();
      ChildNodes children2 = node2.getChildNodes();
      if ( children1.getLength() != children2.getLength() ) {
        return false;
      }
      for ( int i = 0; i < children1.getLength(); i++ ) {
        result &= equals( children1.getNode( i ), children2.getNode( i ) );
      }
    }
    return result;
  }
}
```

```java
package edu.it.dvm;

import edu.it.disk.CellReference;


import edu.it.disk.Disk;
import edu.it.disk.LocalDisk;
import edu.it.disk.ValueReference;
import edu.it.dvm.nameserver.NameServer;

public class DVMXMLStore implements XMLStore {
  protected NameServer nameserver;
  protected Disk disk;


  protected DVMXMLStore() { }


  DVMXMLStore( Disk disk, String props ) throws DVMException {
    this( disk, props, "" );
  }


  DVMXMLStore( Disk disk, String props, String path ) throws DVMException {
    this.disk = disk;
    nameserver = NameServer.getInstance();
  }


  public ValueReference save( Node node ) throws DVMException {
    SaveVisitor visitor = new SaveVisitor( disk );
    node.accept( visitor );
    return visitor.getValueReference();
  }


  public Node load( ValueReference vStreamRef ) throws DVMException {
    if ( vStreamRef instanceof CellReference ) {
      return new DVMMutNode( disk, (CellReference) vStreamRef );
    }
    return new DVMImmNodeProxy( disk, vStreamRef );
  }


  public ValueReference lookup( String name ) throws DVMException {
    return nameserver.lookup( name );
  }


  public void bind( String name, ValueReference vRef )
       throws DVMException {
    nameserver.bind( name, vRef );
  }


  public void rebind( String name, ValueReference vRef )
       throws DVMException {
    nameserver.rebind( name, vRef );
  }
}
```



```java
package edu.it.dvm;


import edu.it.disk.Disk;
import edu.it.disk.DiskException;
import edu.it.disk.DiskFactory;
import edu.it.properties.XMLStoreProperties;
import java.io.*;

public class DVMXMLStoreFactory extends XMLStoreFactory {
  DVMXMLStoreFactory() { }


  public Node createElementNode( String tagName ) {
    ChildNodes children = createChildNodes( new Node[0] );
    return createElementNode( tagName, children, new Attribute[0] );
  }


  public Node createElementNode( String tagName, Attribute[] attrs ) {
    ChildNodes children = createChildNodes( new Node[0] );
    return createElementNode( tagName, children, attrs );
  }


  public Node createElementNode( String tagName, ChildNodes children ) {
    return createElementNode( tagName, children, new Attribute[0] );
  }


  public Node createElementNode( String tagName, Node child ) {
    return new DVMImmNode( Node.ELEMENT, tagName,
        createChildNodes( new Node[]{child} ),
        new Attribute[0] );
  }


  public Node createElementNode( String tagName, ChildNodes children,
      Attribute[] attrs ) {
    return new DVMImmNode( Node.ELEMENT, tagName, children, attrs );
  }


  public Node createElementNode( String tagName, Node child, Attribute[] attrs )
 {
    return new DVMImmNode( Node.ELEMENT, tagName,
        createChildNodes( new Node[]{child} ), attrs );
  }


  public Node createCharDataNode( String data ) {
    return new DVMImmNode( Node.CHARDATA, data,
        createChildNodes( new Node[]{} ), new Attribute[0] );
  }


  public MutNode createMutElementNode( String tagName, ChildNodes children,
      Attribute[] attrs ) {
    return new DVMMutNode( createElementNode( tagName, children, attrs ) );
  }
```

```
  public MutNode createMutCharDataNode( String data ) {
    return new DVMMutNode( createCharDataNode( data ) );
  }


  public ChildNodes createChildNodes( Node[] nodes ) {
    return new DVMChildNodes( nodes );
  }


  public Attribute createAttribute( String name, String value ) {
    return new DVMAttribute( name, value );
  }


  public XMLStore createXMLStore( String name ) throws DVMException {
    try {
      String path = extractPath( name );
      Disk disk = DiskFactory.create( propertiesId ).createDisk( name, propertie
sId );
      return new DVMXMLStore( disk, propertiesId, path );
    } catch ( DiskException e ) {
      throw new DVMException( DVMException.DISK, e );
    }
  }


  private String extractPath( String storeName ) {
    int index = storeName.lastIndexOf( File.separator );
    if ( index >= 0 ) {
      return storeName.substring( 0, index );
    }
    return "";
  }
}
```



```
package edu.it.dvm;

/**
 *  MutNode.java Created: Thu Jul 25 21:14:46 2002
 *
 *@author  <a href="mailto:tejl@it-c.dk">Jesper Tejlgaard Pedersen</a>
 */

public interface MutNode extends Node {

  /**
   *  The method sets the state of the mutable node. This method provides
   *  the only functionality to modify mutable nodes.
   *
   *@param  node an immutable node representing the new state of the
   *      mutable node.
   *@throws  DVMException
   */
  public void setNodeState( Node node ) throws DVMException;


  /**
   *  The method returns the current state of a mutable node. Working with
   *  mutable nodes, this method should always be prefered to getNodeValue,
   *  getAttribute, getAttributeNames and getChildNodes since the use of
   *  these can provide an inconsistent view of the nodes state, if the
   *  node is update by another user/process between to getX calls.
   *
   *@throws  DVMException
   */
  public Node getNodeState() throws DVMException;
}
```

```java
package edu.it.dvm;


import edu.it.disk.DiskException;

public interface Node {
  /**
   *  Constant used when a node is an element Node
   */
  public final static byte ELEMENT = 0x00;
  /**
   *  Constant used when a node is a character data Node
   */
  public final static byte CHARDATA = 0x01;


  /**
   *  In case the node is immutable false is returned otherwise true.
   *
   *@return  true if mutable, false if immutable
   *@throws  DVMException
   */
  public boolean isMutable() throws DVMException;


  /**
   *  In case Node represents a element the constant ELEMENT is returned
   *  else CHARDATA is returned.
   *
   *@return  The node type
   *@throws  DVMException
   */
  public byte getType() throws DVMException;


  /**
   *  The method returns the value of the node. When the node represents a
   *  character data Node the returned value will be the character data,
   *  and when the node represents a element Node the returned value will
   *  by the tag name of the element.
   *
   *@return  The node value
   *@throws  DVMException
   */
  public String getNodeValue() throws DVMException;


  /**
   *  The method returns the attribute with the given name. In case the
   *  node represents a character data Node the returned value is null.
   *
   *@return  The attribute named by the argument.
   *@throws  DVMException
   */
  public String getAttribute( String attrName ) throws DVMException;


  /**
   *  The method returns an array containing the names of the nodes
   *  attributes. In case the node represents a character data Node the
   *  returned array is null.
   *
```



```java
   *@return  The attribute names of the node
   *@throws  DVMException
   */
  public String[] getAttributeNames() throws DVMException;


  /**
   *  For element Nodes the children is returned. If the node represents
   *  character data the returned ChildNodes will be empty.
   *
   *@return  ChildNodes containing the children of the node.
   *@throws  DVMException
   */
  public ChildNodes getChildNodes() throws DVMException;


  /**
   *  This method is part of the visitor pattern for extending the node
   *  with additional functionality. Traversement of the node structure
   *  must be done in Visitor's visit method.
   *
   *@throws  DVMException
   */
  public void accept( Visitor v ) throws DVMException;

  //public Node addNode(int index, Node child);
  //public Node replaceNode(int index , Node newNode);
  //public Node removeNode(int index);

  //public Node getNodeSet(String expr);
}
```

```java
package edu.it.dvm;


import edu.it.disk.*;

public class SaveVisitor implements Visitor {
  protected Disk disk;
  protected ValueReference vStreamRef;


  protected SaveVisitor( Disk disk ) {
    this.disk = disk;
  }


  ValueReference getValueReference() {
    return vStreamRef;
  }


  public void visit( DVMImmNode node ) throws DVMException {
    try {
      vStreamRef = disk.save( makeValue( node ) );
    } catch ( DiskException e ) {
      throw new DVMException( DVMException.DISK, e );
    }
  }


  public void visit( DVMMutNode node ) throws DVMException {
    try {
      if ( node.getCellReference() == null ) {
        CellReference cell = disk.saveCell( makeValue( node.getNodeState() ) );
        node.setCell( cell );
        node.setDisk( disk );
      }
      vStreamRef = node.getCellReference();
    } catch ( DiskException e ) {
      throw new DVMException( DVMException.DISK, e );
    }
  }


  public void visit( DVMImmNodeProxy np ) throws DVMException {
    vStreamRef = np.getValueReference();
  }


  private ValueStream makeValue( Node node ) throws DVMException {
    byte immutable = node.isMutable() ? DVMNodeLoader.MUTABLE
        : DVMNodeLoader.IMMUTABLE;
    String sValue = new String( new byte[]{immutable, node.getType()} );
    sValue += node.getNodeValue();
    // an elementnode does not necessarily have a name.
    if ( node.getType() == Node.ELEMENT ) {
      String[] names = node.getAttributeNames();
      for ( int i = 0; i < names.length; i++ ) {
        sValue += " " + names[i] + "=" + node.getAttribute( names[i] );
      }
    }

    ChildNodes children = node.getChildNodes();
```



```java
    ValueStream vStream = null;
    try {
      ValueStreamFactory factory = SimpleValueStreamFactory.getInstance();
      vStream = factory.mkEmpty();
      if ( children.getLength() > 0 ) {
        Node child = children.getNode( children.getLength() - 1 );
        child.accept( this );
        if ( child.isMutable() ) {
          vStream = factory.mkCellRefValueStream( (CellReference) this.getValueR
eference(),
              vStream );
        } else {
          vStream = factory.mkValRefValueStream( this.getValueReference(),
              vStream );
        }
        for ( int i = children.getLength() - 2; i >= 0; i-- ) {
          // byte sequences and valuestream references is alternating
          // in value streams.
          // Thus vs := bytes (valuestream reference | bytes)*
          // must be followed. Therefore empty byte sequences are used.
          vStream = factory.mkByteValueStream( new byte[]{}, vStream );
          child = children.getNode( i );
          child.accept( this );
          if ( child.isMutable() ) {
            vStream = factory.mkCellRefValueStream( (CellReference) this.getValu
eReference(),
                vStream );
          } else {
            vStream = factory.mkValRefValueStream( this.getValueReference(),
                vStream );
          }
        }
        vStream = factory.mkByteValueStream( sValue.getBytes(), vStream );
      } else {
        vStream = factory.mkByteValueStream( sValue.getBytes(), vStream );
      }
    } catch ( DiskException e ) {
      throw new DVMException( DVMException.DISK, e );
    }
    return vStream;
  }

}
```

```java
package edu.it.dvm;

/**
 *  Visitor.java Created: Thu Mar 14 17:08:53 2002
 *
 *@author  <a href="mailto:tejl@it-c.dk">Jesper Tejlgaard Pedersen</a> 2.0
 */

public interface Visitor {
  public void visit( DVMImmNode n ) throws DVMException;


  public void visit( DVMMutNode n ) throws DVMException;


  public void visit( DVMImmNodeProxy np ) throws DVMException;
}
```



```java
package edu.it.dvm;


import edu.it.disk.ValueReference;

public interface XMLStore {

  /**
   *  The method is used for persisting XML data. The data is given the
   *  node param. The method returns a unique location independent
   *  reference, which can be used to load in the node, when needed.
   *
   *@param  node representing XML data in the form of semistructured tree
   *      data.
   *@return  unique location independent reference to the saved node/tree.
   *@throws  DVMException
   */
  public ValueReference save( Node node ) throws DVMException;


  /**
   *  The method is used for loading persisted XML data. The data is given
   *  by the param, which is a unique location independent reference. The
   *  returned value is semistructured tree data, represented by a "root
   *  node" having 0 or more subtrees themself represented by nodes (see
   *  Node).
   *
   *@param  vref a unique location independent reference to the data.
   *@return  node, representing the loaded XML data.
   *@throws  DVMException
   */
  public Node load( ValueReference vref ) throws DVMException;


  /**
   *  The method returns a unique location independent reference named by
   *  the methods param.
   *
   *@param  name of unique location independent reference.
   *@return  unique location independent reference.
   *@throws  DVMException
   */
  public ValueReference lookup( String name ) throws DVMException;


  /**
   *  The method binds a unique location independent reference to a name.
   *  In case a unique location independent reference is already bound to
   *  the given name, an exception is thrown
   *
   *@param  name name for location independent reference
   *@param  vref location independent reference.
   *@throws  DVMException
   */
  public void bind( String name, ValueReference vref )
       throws DVMException;


  /**
   *@param  name name for location independent reference
   *@param  vref location independent reference.
   *@throws  DVMException
```

```java
   */
  public void rebind( String name, ValueReference vRef ) throws DVMException;
}
```



```java
package edu.it.dvm;


import edu.it.properties.*;
import java.io.IOException;
import java.util.HashMap;

public abstract class XMLStoreFactory {
  private static HashMap factories = new HashMap();
  protected String propertiesId;


  /**
   *  The method creates an immutable node representing the character data
   *  provided in the argument
   *
   *@param  data character data to be represent by a node.
   *@return  Node reprensenting character data
   */
  public abstract Node createCharDataNode( String data );


  /**
   *  The method creates and returns a immutable element node, with the
   *  given tag name and children. The returned node does not have
   *  attributes.
   *
   *@param  tagName name of element node
   *@param  children children of element node.
   *@return  element node.
   */
  public abstract Node createElementNode( String tagName, ChildNodes children );


  /**
   *  The method creates and returns a immutable element node, with the
   *  given tag name and child. The returned node does not have attributes.
   *
   *@param  tagName name of element node
   *@param  child child of element node.
   *@return  element node.
   */
  public abstract Node createElementNode( String tagName, Node child );


  /**
   *  The method creates and returns a immutable element node, with the
   *  given tag name, children and attributes.
   *
   *@param  tagName name of new element node
   *@param  attrs attributes of new element node
   *@param  children children of new element node.
   *@return  element node.
   */
  public abstract Node createElementNode( String tagName, ChildNodes children,
      Attribute[] attrs );


  /**
   *  The method creates and returns a immutable element node, with the
   *  given tag name, child and attributes.
   *
```

```java
 *@param  tagName name of new element node
 *@param  attrs attributes of new element node
 *@param  child child of new element node.
 *@return  element node.
 */
public abstract Node createElementNode( String tagName, Node child,
    Attribute[] attrs );


/**
 *  The method creates and returns a immutable element node, with the
 *  given tag name and attributes. The returned node does not have
 *  children.
 *
 *@param  tagName name of new element node
 *@param  attrs attributes of new element node
 *@return  element node.
 */
public abstract Node createElementNode( String tagName, Attribute[] attrs );



/**
 *  The method creates and returns a immutable element node, with the
 *  given tag name. The node does not have any attributes nor children.
 *
 *@param  tagName name of new element node
 *@return  element node.
 */
public abstract Node createElementNode( String tagName );


/**
 *  Similar to createElementNode(String value,Attribute[]
 *  attrs,ChildNodes children) only the returned node is mutable.
 *
 *@param  tagName name of new element node
 *@param  attrs attributes of new element node
 *@param  children children of new element node.
 *@return  mutable element node.
 */
public abstract MutNode createMutElementNode( String tagName,
    ChildNodes children,
    Attribute[] attrs );


/**
 *  Similar to createCharDataNode(String value), only the returned node
 *  is mutable.
 *
 *@param  data character data to be represent by a node.
 *@return  Mutable node reprensenting character data
 */
public abstract MutNode createMutCharDataNode( String data );


/**
 *  Similar to createCharDataNode(String value), only the returned node
 *  is mutable.
 *
 *@param  name name of attribute
 *@param  value value of attribute
```



```java
 *@return  instance of Attribute representing an attribute of an element.
 */
public abstract Attribute createAttribute( String name, String value );


/**
 *  Creates an instance of ChildNodes.
 *
 *@param  nodes the child nodes to be represented by an instance of
 *      ChildNodes.
 *@return  instance of ChildNodes.
 */
public abstract ChildNodes createChildNodes( Node[] nodes );


public abstract XMLStore createXMLStore( String name ) throws DVMException;


/**
 *  The method returns the default instance of a XMLStoreFactory.
 *
 *@return  Instance of a XMLStoreFactory.
 *@throws  DVMException.
 */
public static XMLStoreFactory getInstance() throws DVMException {
  return getInstance( "xmlstore" );
}


/**
 *  The method returns an instance of a XMLStoreFactory. The param is the
 *  name of a property file, in which the type of the instantiated
 *  XMLStoreFactory is written.
 *
 *@return  Instance of a XMLStoreFactory.
 *@throws  DVMException.
 */
public static XMLStoreFactory getInstance( String propsId )
     throws DVMException {
  XMLStoreFactory factory = (XMLStoreFactory) factories.get( propsId );
  XMLStoreProperties properties;
  if ( factory == null ) {
    try {
      properties = XMLStoreProperties.getInstance( propsId );
      String name = properties.getFactoryName();
      factory = (XMLStoreFactory) Class.forName( name ).newInstance();
      factory.propertiesId = propsId;
      factories.put( propsId, factory );
    } catch ( ClassNotFoundException e ) {
      System.out.println( "ClassNotFound caught" );
      throw new DVMException( DVMException.CLASSNOTFOUND, e );
    } catch ( IllegalAccessException e ) {
      System.out.println( "IllegalAccess caught" );
      throw new DVMException( DVMException.ILLEGAL_ACCESS, e );
    } catch ( InstantiationException e ) {
      System.out.println( "Instantiation caught" );
      throw new DVMException( DVMException.INSTANTIATION, e );
    } catch ( PropertyException e ) {
      throw new DVMException( DVMException.PROPS, e );
    }
  }
  return factory;
```

```
  }
}
```



```
package edu.it.dvm.nameserver;

import edu.it.disk.*;
import edu.it.dvm.*;

import edu.it.properties.*;
import edu.it.util.*;
import java.io.IOException;
import java.net.*;

public class GlobalNameServer extends NameServer {
  private static NameServer instance;
  private NameServer localServer;
  private ReferenceFactory refFac;
  private Communicator com;


  public GlobalNameServer() throws IOException, PropertyException, DVMException
{
    XMLStoreProperties props = XMLStoreProperties.getInstance( "xmlstore" );
    refFac = SimpleReferenceFactory.getInstance();
    localServer = LocalNameServer.getInstance();
    this.com = new CentralCommunicator( InetAddress.getByName( props.getNameServ
erIP() ), props.getNameServerPort() );
  }


  public ValueReference lookup( String name ) throws DVMException {
    ValueReference vr = localServer.lookup( name );
    if ( vr == null ) {
      byte[] bname = name.getBytes();
      byte[] msg = new byte[bname.length + 2];
      msg[0] = MapAction.LOOKUP;
      msg[1] = (byte) bname.length;
      // buggy if length > 128
      System.arraycopy( bname, 0, msg, 2, bname.length );
      byte[] resp = new byte[Communicator.PAC_SIZE];
      int len = -1;
      try {
        len = com.sendReceive( msg, resp );
      } catch ( IOException e ) {
        throw new DVMException( DVMException.IO, e );
      }
      if ( len != MapAction.NOT_FOUND ) {
        byte[] ref = new byte[ValueReference.LENGTH];
        System.arraycopy( resp, 1, ref, 0, len - 1 );
        if ( resp[0] == ValueType.V_REF ) {
          vr = refFac.createValueReference( ref );
        } else if ( resp[0] == ValueType.C_REF ) {
          vr = refFac.createCellReference( ref );
        }
      }
    }
    return vr;
  }


  public void bind( String name, ValueReference vr ) throws DVMException {
    localServer.bind( name, vr );
    //bind glocally needed for central server
    byte[] bname = name.getBytes();
    byte[] bvr = vr.toByteArray();
```

```java
    byte[] msg = new byte[bname.length + bvr.length + 4];
    msg[0] = MapAction.BIND;
    msg[1] = (byte) bname.length;
    //buggy if bname.length > 128
    System.arraycopy( bname, 0, msg, 2, bname.length );
    msg[2 + bname.length] = (byte) ( bvr.length + 1 );
    msg[2 + bname.length + 1] = ( vr instanceof CellReference ) ? ValueType.C_RE
F : ValueType.V_REF;
    System.arraycopy( bvr, 0, msg, 4 + bname.length, bvr.length );
    try {
      com.send( msg );
    } catch ( IOException e ) {
      throw new DVMException( DVMException.IO, e );
    }
  }


  public void rebind( String name, ValueReference vr ) throws DVMException {
    bind( name, vr );
  }


  public static NameServer getInstance() throws DVMException {
    if ( instance == null ) {
      try {
        instance = new GlobalNameServer();
      } catch ( IOException e ) {
        throw new DVMException( DVMException.IO, e );
      } catch ( PropertyException e ) {
        throw new DVMException( DVMException.PROPS, e );
      }
    }
    return instance;
  }
}
```



```java
package edu.it.dvm.nameserver;


import edu.it.disk.*;
import edu.it.dvm.*;
import edu.it.util.*;
import java.io.*;

public class LocalNameServer extends NameServer {
  private PHashtable table;
  private ReferenceFactory refFac;
  private static LocalNameServer instance;


  public LocalNameServer() throws DVMException {
    try {
      this.table = new PHashtable( "nameserver.hash", true );
    } catch ( IOException e ) {
      throw new DVMException( DVMException.IO, e );
    }
    refFac = SimpleReferenceFactory.getInstance();
  }


  public ValueReference lookup( String docName ) {
    byte[] ref = table.get( docName.getBytes() );
    if ( ref == null ) {
      return null;
    }
    byte[] res = new byte[ref.length - 1];
    System.arraycopy( ref, 1, res, 0, res.length );
    if ( ref[0] == ValueType.V_REF ) {
      return refFac.createValueReference( res );
    }
    return refFac.createCellReference( res );
  }


  public void bind( String docName, ValueReference vRef ) throws DVMException {
    try {
      byte[] ref = new byte[ValueReference.LENGTH + 1];
      ref[0] = vRef instanceof CellReference ? ValueType.C_REF : ValueType.V_REF
;
      System.arraycopy( vRef.toByteArray(), 0, ref, 1, ValueReference.LENGTH );
      table.put( docName.getBytes(), ref );
    } catch ( IOException e ) {
      throw new DVMException( DVMException.IO, e );
    }
  }


  public void rebind( String docName, ValueReference ref ) throws DVMException {
    bind( docName, ref );
  }


  public static NameServer getInstance() {
    return getLocalInstance();
  }


  public static LocalNameServer getLocalInstance() {
```

```
    if ( instance == null ) {
      try {
        instance = new LocalNameServer();
      } catch ( DVMException e ) {
        System.err.println( "Creating local reference server failed " + e );
      }
    }
    return instance;
  }


  public PHashtable getTable() {
    return table;
  }
}
```



```
package edu.it.dvm.nameserver;

import edu.it.disk.SimpleValueReference;

import edu.it.disk.ValueReference;

public class NSConverter {

  public String createKey( Object o ) {
    return (String) o;
  }


  public String createSValue( Object o ) {
    String valref = new String( ( (ValueReference) o ).toByteArray() );
    return valref.replaceAll( "<v>", "" ).replaceAll( "</v>", "" );
  }


  public Object createValue( String s ) {
    return SimpleValueReference.create( s.getBytes() );
  }
}
```

```java
package edu.it.dvm.nameserver;


import edu.it.disk.ValueReference;
import edu.it.dvm.DVMException;
import edu.it.properties.*;

public abstract class NameServer {
  private static NameServer instance;


  public abstract ValueReference lookup( String name ) throws DVMException;


  public abstract void rebind( String name, ValueReference vRef )
       throws DVMException;


  public abstract void bind( String name, ValueReference ref )
       throws DVMException;


  /*
   *  protected abstract void initialize(String path) throws DVMException;
   *  public static NameServer create(String propsId, String path)
   *  throws DVMException{
   *  try{
   *  String className = XMLStoreProperties.getInstance(propsId).getNameServer()
;
   *  NameServer ns = (NameServer)(Class.forName(className).newInstance());
   *  ns.initialize(path);
   *  return ns;
   *  }catch(InstantiationException e){
   *  throw new DVMException(DVMException.INSTANTIATION,e);
   *  }catch(ClassNotFoundException e){
   *  throw new DVMException(DVMException.CLASSNOTFOUND,e);
   *  }catch(IllegalAccessException e){
   *  throw new DVMException(DVMException.ILLEGAL_ACCESS,e);
   *  }catch(PropertyException e){
   *  throw new DVMException(DVMException.PROPS,e);
   *  }
   *  }
   */
  public static NameServer getInstance() throws DVMException {
    if ( instance == null ) {
      //do configuration here
      instance = GlobalNameServer.getInstance();
      //instance = new LocalNameServer();

    }
    return instance;
  }
}
```



```java
package edu.it.util;

import java.io.*;


import java.net.*;

public class CentralCommunicator implements Communicator {
  private Socket socket;
  private OutputStream out;
  private InputStream in;


  public CentralCommunicator( InetAddress serverIP, int serverPort ) throws IOEx
ception {
    connect( serverIP, serverPort );
  }


  public void send( byte[] msg ) throws IOException {
    out.write( msg );
    out.flush();
  }


  public int sendReceive( byte[] msg, byte[] respons ) throws IOException {
    out.write( msg );
    out.flush();
    int length = in.read( respons );
    if ( length == 1 && respons[0] == MapAction.NOT_FOUND ) {
      return MapAction.NOT_FOUND;
    }
    return length;
  }


  private void connect( InetAddress serverIP, int serverPort ) throws IOExceptio
n {
    socket = new Socket( serverIP, serverPort );
    out = new BufferedOutputStream( socket.getOutputStream() );
    out.flush();
    in = new BufferedInputStream( socket.getInputStream() );
  }


  public void disconnect() {
    try {
      socket.close();
    } catch ( IOException e ) {

    }
    socket = null;
  }
}
```

```java
package edu.it.util;

import java.io.*;
import java.net.*;

import java.util.*;

public class CentralMap {
  private int port;
  private Thread serverThread;
  private ServerSocket serverSocket = null;
  private PHashtable map = null;
  private List requestHandlers = new LinkedList();


  public CentralMap( int serverport, String fileName ) throws IOException {
    this( serverport, fileName, false );
  }


  public CentralMap( int serverport, String fileName, boolean nameserver ) throw
s IOException {
    this.port = serverport;
    map = new PHashtable( fileName, nameserver );

    serverThread =
      new Thread() {
        public void run() {
          try {
            serverSocket = new ServerSocket( port );
            System.out.println( "Server started" );
            while ( !serverThread.isInterrupted() ) {
              Socket socket = serverSocket.accept();
              System.out.println( "Server accepted connection" );
              requestHandlers.add( new RequestHandler( socket ) );
            }
          } catch ( IOException e ) {
            if ( !serverThread.isInterrupted() ) {
              System.out.println( "Exception during initialization of serverThread" );
              e.printStackTrace();
              Thread.currentThread().interrupt();
              System.exit( -1 );
            }
          }
        }
      };
    //serverThread.setPriority(10);
  }


  public void start() {
    serverThread.start();
  }


  public void stop() throws IOException {
    serverThread.interrupt();
    serverSocket.close();
    ListIterator it = requestHandlers.listIterator();
    while ( it.hasNext() ) {
      ( (Thread) it.next() ).interrupt();
    }
```



```java
  }


  private class RequestHandler extends Thread {
    private Socket socket;


    public RequestHandler( Socket socket ) {
      this.socket = socket;
      start();
    }


    public void run() {
      OutputStream out = null;
      InputStream in = null;
      try {
        out = new BufferedOutputStream( socket.getOutputStream() );
        out.flush();
        in = new BufferedInputStream( socket.getInputStream() );
      } catch ( IOException e ) {
        try {
          String msg = "IOException while connecting to :";
          msg += socket.getInetAddress();
          System.out.println( msg );
          socket.close();
          interrupt();
        } catch ( IOException ex ) {
        } catch ( SecurityException ex ) {}
      }
      while ( !isInterrupted() ) {
        try {
          byte type = (byte) in.read();
          switch ( type ) {
              case MapAction.LOOKUP:
              {
                System.out.println( "map Lookup" );
                // reads length of key and initialize byte array
                byte[] key = new byte[in.read()];
                // reads key
                in.read( key );
                byte[] result = map.get( key );

                if ( result == null ) {
                  result = new byte[]{MapAction.NOT_FOUND};
                }
                out.write( result );
                out.flush();
                break;
              }
              case MapAction.BIND:
              {
                System.out.println( "map bind" );
                // reads length of key and initialize byte array
                byte[] key = new byte[in.read()];
                // reads key
                in.read( key );
                // reads length of value and initializes byte array
                byte[] value = new byte[in.read()];
                // reads value
                in.read( value );
                // binds
```

```java
            map.put( key, value );
            break;
          }
          case MapAction.REBIND:
          {
            // reads length of key and initialize byte array
            byte[] key = new byte[in.read()];
            // reads key
            in.read( key );
            // reads length of value and initializes byte array
            byte[] value = new byte[in.read()];
            // reads value
            in.read( value );
            // binds
            map.put( key, value );
            break;
          }
        }
      } catch ( Exception e ) {
        if ( !socket.isClosed() ) {
          try {
            socket.close();
          } catch ( IOException ex ) {
            /*
             *  tjajajajaja
             */
          }
        }
      }
    }
  }
}


  public static void main( String[] args ) throws IOException {
    CentralMap server = null;
    if ( args.length > 0 && args[0].equals( "refserver" ) ) {
      server = new CentralMap( 2224, "centralmaprs.hash" );
    } else if ( args.length > 0 && args[0].equals( "nameserver" ) ) {
      server = new CentralMap( 2222, "centralmapns.hash", true );
    } else {
      System.out.println( "wrong arguments:" );
      System.exit( -1 );
    }
    server.start();
  }
}
```



```java
package edu.it.util;

import java.io.*;

public interface Communicator {
  public final static int PAC_SIZE = 256;


  public void send( byte[] msg ) throws IOException;


  public int sendReceive( byte[] msg, byte[] respons ) throws IOException;
}
```

```java
package edu.it.util;

import edu.it.disk.*;
import java.net.InetAddress;
import java.net.UnknownHostException;

public class Converter {

  /**
   *  Converts a string representation of a locator to an object
   */
  public static Locator string2locator( String value ) throws DiskException {
    try {
      String[] sArr = value.split( " " );
      if ( sArr == null || !( sArr.length == 4 || sArr.length == 6 ) ) {
        String msg = "Argument must be a string of the form:\n";
        msg += "'type diskname, offset length')\n";
        msg += "'type ip port diskname, offset length'";
        throw new IllegalArgumentException( msg );
      }

      if ( sArr.length == 4 ) {
        int off = Integer.parseInt( sArr[2] );
        int len = Integer.parseInt( sArr[3] );
        if ( Byte.parseByte( sArr[0] ) == Locator.CELL ) {
          return LocatorFactory.createLocalCellLocator( sArr[1], off, len );
        }
        return LocatorFactory.createLocalLocator( sArr[1], off, len );
      }

      byte[] ip = InetAddress.getByName( sArr[1] ).getAddress();
      int port = Integer.parseInt( sArr[2] );
      int off = Integer.parseInt( sArr[4] );
      int len = Integer.parseInt( sArr[5] );

      if ( Byte.parseByte( sArr[0] ) == Locator.CELL ) {
        return LocatorFactory.createGlobalCellLocator( ip, port, sArr[3], off, l
en );
      }
      return LocatorFactory.createGlobalLocator( ip, port, sArr[3], off, len );
    } catch ( UnknownHostException e ) {
      throw new DiskException( DiskException.UN_KNOWN, e );
    }
  }
}
```



```java
package edu.it.util;

public interface MapAction {
  public final static byte LOOKUP = 0x00;
  public final static byte BIND = 0x01;
  public final static byte REBIND = 0x02;
  public final static byte NOT_FOUND = -1;
}
```

```java
package edu.it.util;

import edu.it.disk.refserver.*;
import edu.it.dvm.nameserver.*;
import edu.it.properties.*;

import java.io.*;
import java.net.*;
import java.text.SimpleDateFormat;
import java.util.*;

public class MulticastCommunicator implements Communicator {
  private final static int RID_LEN = 12;
  private final static int RPORT_LEN = 4;
  private RequestMonitor monitor;
  private MulticastSocket socket;
  private InetAddress group;
  private int port;
  private boolean connected;
  private MulticastRequestReceiver rr;
  private int timeout;


  public MulticastCommunicator( InetAddress group, final int port ) throws Prope
rtyException, IOException {
    this.group = group;
    this.port = port;
    this.socket = new MulticastSocket( port );
    socket.joinGroup( group );
    this.rr = new MulticastRequestReceiver();
    rr.start();
    this.connected = true;
    timeout = XMLStoreProperties.getInstance( "xmlstore" ).getNetworkTimeOut();

    monitor = new RequestMonitor();
      new Thread() {
        public void run() {
          DatagramSocket dsocket = null;
          try {
            dsocket = new DatagramSocket( ( port + 1 ) );
          } catch ( SocketException e ) {
            System.err.println( "Error creating socket in multicast"
                + " communicator" + e );
          }
          while ( connected ) {
            DatagramPacket dg = new DatagramPacket( new byte[Communicator.PAC_SI
ZE], Communicator.PAC_SIZE );
            try {
              dsocket.receive( dg );
            } catch ( IOException e ) {
              System.err.println( "Error receiving respons in multicast"
                  + " communicator" + e );
            }
            // should be handled by a thread
            byte[] reqid = new byte[RID_LEN];
            byte[] answer = new byte[dg.getLength() - RID_LEN];
            byte[] msg = dg.getData();
            System.arraycopy( msg, 0, reqid, 0, RID_LEN );
            System.arraycopy( msg, RID_LEN, answer, 0, answer.length );
            monitor.setRequest( reqid, answer );
          }
          dsocket.close();
```



```java
        }
      }.start();
  }


  public int sendReceive( byte[] msg, byte[] repons ) throws IOException {
    byte[] bport = Integer.toString( ( port + 1 ) ).getBytes();
    byte[] id = getNextRequestId();
    byte[] msg1 = new byte[msg.length + bport.length + id.length];

    System.arraycopy( msg, 0, msg1, 0, msg.length );
    System.arraycopy( id, 0, msg1, msg.length, id.length );
    System.arraycopy( bport, 0, msg1, msg.length + id.length, bport.length );

    socket.send( new DatagramPacket( msg1, msg1.length, group, port ) );
    byte[] req = monitor.getRequest( id, timeout );
    if ( req == null || Arrays.equals( new byte[]{MapAction.NOT_FOUND}, req ) )
{
      return MapAction.NOT_FOUND;
    }
    System.arraycopy( req, 0, repons, 0, req.length );
    return req.length;
  }


  public void send( byte[] msg ) throws IOException {
    socket.send( new DatagramPacket( msg, msg.length, group, port ) );
  }


  public void disconnect() {
    rr.disconnect();
    connected = false;
    socket.close();
  }


  private byte[] getNextRequestId() {
    Date date = new Date( System.currentTimeMillis() );
    SimpleDateFormat timeFormat = new SimpleDateFormat( "hh:mm:ss:SSS" );
    return timeFormat.format( date ).getBytes();
  }


  private class MulticastRequestReceiver extends Thread {

    private boolean connected;
    private DatagramSocket dsocket;
    private PHashtable table;


    public MulticastRequestReceiver() throws IOException {
      this.table = LocalReferenceServer.getLocalInstance().getTable();
      this.connected = true;
      this.dsocket = new DatagramSocket();
      // used for sending
    }


    public void run() {
      while ( connected ) {
        DatagramPacket dg = new DatagramPacket( new byte[Communicator.PAC_SIZE],
```

```
Communicator.PAC_SIZE );
        try {
          socket.receive( dg );
          new RequestHandler( dg ).start();
        } catch ( IOException e ) {
          System.err.println( "error receiving in MulticastReceiver" + e );
        }
      }
    }


    public class RequestHandler extends Thread {
      private DatagramPacket dg;


      public RequestHandler( DatagramPacket dg ) {
        this.dg = dg;
      }


      public void run() {
        byte[] data = dg.getData();
        int length = dg.getLength();
        switch ( data[0] ) {
            case MapAction.LOOKUP:
              // reads length of key and initialize byte array
              byte[] key = new byte[data[1]];
              // reads key
              System.arraycopy( data, 2, key, 0, key.length );
              byte[] result = table.get( key );
              if ( result == null ) {
                //not found do nothing
                return;
              }
              int port = Integer.parseInt( new String( data, 2 + key.length + RI
D_LEN, RPORT_LEN ) );
              SocketAddress sa = new InetSocketAddress( dg.getAddress(), port );

              byte[] msg = new byte[RID_LEN + result.length];

              System.arraycopy( data, 2 + key.length, msg, 0, RID_LEN );
              System.arraycopy( result, 0, msg, RID_LEN, result.length );
              try {
                DatagramPacket answerdg = new DatagramPacket( msg, msg.length, s
a );
                dsocket.send( answerdg );
              } catch ( SocketException e ) {} catch ( IOException e ) {}
              break;
            case MapAction.BIND:
              byte[] bkey = new byte[data[1]];
              System.arraycopy( data, 2, bkey, 0, bkey.length );
              byte[] bvalue = new byte[data[1 + bkey.length]];
              System.arraycopy( data, 2 + bkey.length, bvalue, 0, bvalue.length
);
              try {
                table.put( bkey, bvalue );
              } catch ( IOException e ) {
                System.err.println( "Exception caught during bind "
                    + e.getMessage() );
              }
              break;
        }
```



```
      }
    }


    public void disconnect() {
      connected = false;
    }
  }
}
```

```java
package edu.it.util;

import edu.it.disk.ValueReference;
import java.io.*;

import java.util.HashMap;

public class PHashtable {
  private HashMap hashtable;
  private BufferedOutputStream out;
  private boolean nameserver;


  public PHashtable( String filename ) throws IOException {
    this( filename, false );
  }


  public PHashtable( String filename, boolean nameserver ) throws IOException {
    this.hashtable = new HashMap();
    this.out = new BufferedOutputStream( new FileOutputStream( filename, true )
);
    this.nameserver = nameserver;
    BufferedInputStream in = new BufferedInputStream( new FileInputStream( filen
ame ), 10000 );
    int reflen = ValueReference.LENGTH;
    if ( nameserver ) {
      reflen = ValueReference.LENGTH + 1;
    }
    byte[] ref = new byte[reflen];
    byte[] loc = new byte[100];
    while ( in.read( ref ) != -1 ) {
      int c;
      int i = 0;
      while ( ( c = in.read() ) != '\n' ) {
        loc[i++] = (byte) c;
      }
      if ( nameserver ) {
        hashtable.put( new String( loc, 0, i ), ref );
      } else {
        hashtable.put( new String( ref ), new String( loc, 0, i ).getBytes() );
      }
      ref = new byte[reflen];
      /*
       *  chould be avoided by reimplementing Arrays.equals
       */
    }
  }


  public synchronized byte[] get( byte[] key ) {
    return (byte[]) hashtable.get( new String( key ) );
  }


  public synchronized void put( byte[] key, byte[] val ) throws IOException {
    hashtable.put( new String( key ), val );
    if ( nameserver ) {
      out.write( val );
      out.write( key );
      out.write( "\n".getBytes() );
    } else {
```



```java
      out.write( key );
      out.write( val );
      out.write( "\n".getBytes() );
    }
    out.flush();
  }


  public String toString() {
    return hashtable.toString();
  }
}
```

```java
package edu.it.util;


import java.util.*;

public class RequestMonitor {
  private final static byte[] WAITING = new byte[]{'W'};
  private HashMap map;


  public RequestMonitor() {
    map = new HashMap();
  }


  public synchronized byte[] getRequest( final byte[] id, final int timeout ) {
    final String sId = new String( id );
    byte[] answer = null;
    if ( ( answer = (byte[]) map.remove( sId ) ) != null ) {
      //request has allready been set
      return answer;
    }
    map.put( sId, WAITING );
    //starting timer thread;
      new Thread() {
        public void run() {
          try {
            Thread.sleep( timeout );
          } catch ( InterruptedException e ) {}
          ;
          setTimeOut( sId );
        }
      }.start();

    while ( Arrays.equals( ( (byte[]) map.get( sId ) ), WAITING ) ) {
      try {
        wait();
      } catch ( InterruptedException e ) {
        System.err.println( "Thread interrupted in RequestMonitor" );
      }
    }
    return (byte[]) map.remove( sId );
  }


  public synchronized void setRequest( byte[] id, byte[] answer ) {
    String sId = new String( id );
    if ( !map.containsKey( sId ) || Arrays.equals( (byte[]) map.get( sId ), WAIT
ING ) ) {
      map.put( sId, answer );
      notifyAll();
    }
  }


  private synchronized void setTimeOut( String sId ) {
    byte[] answer = (byte[]) map.get( sId );
    if ( answer != null && Arrays.equals( answer, WAITING ) ) {
      map.put( sId, new byte[]{MapAction.NOT_FOUND} );
      notifyAll();
    }
  }
```



```java
}
```