Athens University of Economics and Business

Department of Management Science and Technology

XINS
XML Interface for Network Services

Advanced topics in software engineering

Georgios Veioglanis
gveiog@dmst.aueb.gr

Contents

1. Introduction

The scope of this document is to describe the contribution to the open source project XINS (XML Interface for Network Services). In particular it points out the main steps followed to achieve the targets set. It starts with an overview of the project that the student was involved, the communication with the developers, the feature proposed for contribution and the implementation of that feature. The student chose that project because it seemed quite interesting and challenging since it deals with web services, an upcoming and promising technology.

2. Overview of XINS

XINS is an open-source Web Services technology, supporting SOAP, XML-RPC and REST. It consists mainly of an XML-based specification format and a Java-based implementation framework.

From its specifications, XINS can generate HTML, WSDL, client-side code, server-side code and test forms. Users do *not* require knowledge of any complex formats, such as XML Schema.

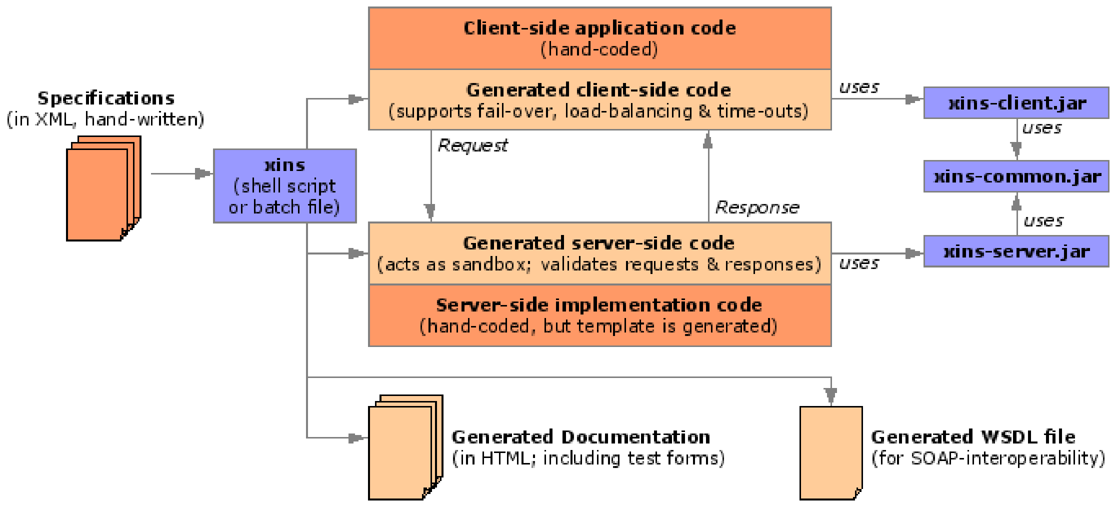
XINS is currently in version 1.4.1 and soon a new version 1.5 will be released with extra features. It is distributed under the license of Wanadoo Nederland B.V.

XINS is technically composed of:

- An XML-based specification format for projects, APIs, functions, types and error codes.
- A POX-style RPC protocol (called the *XINS Standard Calling Convention*), compatible with web browsers (HTTP parameters in, XML out).
- A tool for generating human-readable documentation, from the specifications.
- A tool for generating WSDL, from the specifications.

- A Log4j-based technology for logging (called *Logdoc*), offering a specification format, internationalization of log messages, generation of HTML documentation and generation of code.
- A Java client-side library for calliing XINS functions: the XINS/Java Client Framework; in xins-client.jar.
- A server-side container for Java-based XINS API implementations: XINS/Java Server Framework ; in xins-server.jar. This is like a servlet container for XINS APIs. It supports not only POX-style calls, but also SOAP and XML-RPC. And it supports conversion using XSLT.
- A Java library with some common functionality, used by both the XINS/Java Client Framework and the XINS/Java Server Framework: the XINS/Java Common Library, in xins-common.jar.

Picture 1: schematic overview of XINS


Client-side application code
(hand-coded)
Generated client-side code
(supports fail-over, load-balancing & time-outs)
uses
xins-client.jar
uses
Specifications
(in XML, hand-written)
xins
(shell script
or batch file)
Request
Response
xins-common.jar
uses
Generated server-side code
(acts as sandbox; validates requests & responses)
uses
xins-server.jar
Server-side implementation code
(hand-coded, but template is generated)
Generated Documentation
(in HTML; including test forms)
Generated WSDL file
(for SOAP-interoperability)

3. Communication with the developers

Below there are three representative e-mails that point out the communication with the developers. In particular, firstly is the answer of the main developer, then his propositions about contribution at the project and finally his answer about the files that I send him.

Subject:	xins features
From:	"Anthony Goubard" <anthony.goubard@nl.wanadoo.com>
Date:	Thu, May 18, 2006 3:44 pm
To:	gveiog@dmst.aueb.gr
Cc:	"Ernst de Haan" <ernst.dehaan@nl.wanadoo.com>

Hi Georgios,

Ernst has forwarded to me your request of working on XINS.
I'm the main XINS developer so I can help you if you have
questions/problems.
I'm glad that you considered XINS as subject to your subject.

Concerning the RFE, with XINS we try to keep the focus on simplicity of
developing Web Services and also on code quality. This mean that not all
RFE submitted to SourceForge are automatically accepted and will be
done. First there is a discussion between me and Ernst de Haan.

That's why I propose you the following:
 - First I'd like to ask you a few question to know what is your
technical level and what are you interested in.
 - Then I send you a list of XINS features that you could work on.
 - Then you come with a small design (a text is OK here) which explain
how you want to do it.
 - Then I review it.
 - Then you start working on it.

I'll start with the first step:
I saw that you're studying in a university of economics and business.
Does that mean that computer science is not the main subject of your degree?
What is your programming experience?
Have you already written programs in Java?
Do you know XML?
Do you know XSLT? (As you can see we also use it a lot in XINS)
How much time do you have to write the XINS feature (hours a week)? Do
you have a dead line?

Best regards,
Anthony

Developer's propositions

Subject:	RE: xins features
From:	"Anthony Goubard" <anthony.goubard@nl.wanadoo.com>
Date:	Fri, May 19, 2006 2:00 pm
To:	gveiog@dmst.aueb.gr
Cc:	"Ernst de Haan" <ernst.dehaan@nl.wanadoo.com>

```
Georgios,

 Unfortunately I'll be on holidays starting Thursday evening (May 25).

 Here is the list of the features you could work on:
1) _hex type
2) Write an article about XINS
3) Have an API example that allow to upload a file
4) Having the possibility to have a .version.properties per API
5) Multiplicity attribute for data section elements
6) Display the API version in the generated specdocs and javadoc
7) Improve XSLT unit tests

With your programming experience and time constraint I would advice you to
do the first feature (_hex type). This way you will touch a bit of
everything in XINS (Java code, XSLT, unit tests)
http://sourceforge.net/tracker/index.php?func=detail&aid=1483846&group_id=71
598&atid=531817
Tell me if you agree or if you prefer to do something else.

Here are some details about some of the features
1)
----------------------------------------------------------------------------
---
=> New type _hex
--
The implementation will be very similar to the _base64 type.

2)
----------------------------------------------------------------------------
---
=> More articles
--
 A new documentation page will be created on the web site.
 This documentation page will contain links to the actual XINS
documentation,
articles and presentations, each one having its own section.
 Also this page should be enhanced with more articles. The following article
are suggested for the 1.5 release:
 - Use XINS with Spring
 - Use XINS with Maven
 - Use XINS with JSF
 - XINS and BPEL - how does that work?

Note that the articles can be publish at any time unless the article depends
on
a XINS 1.5 functionality.

3)
----------------------------------------------------------------------------
---
=> Integration of the MultipartCallingConvention
--
XINS 1.5.0 should integrate an example of an API where a file is send with
the call.
I propose to add another example to the demos. The example will be simple
API
with one function that will upload the file. Note that the calling
convention
will be also part of the example.

Anthony
```

His answer

Subject:	RE: Support while implementing XINS _hex feature
From:	"Anthony Goubard" <anthony.goubard@nl.wanadoo.com>
Date:	Wed, June 7, 2006 3:53 pm
To:	gveiog@dmst.aueb.gr
Cc:	"Ernst de Haan" <ernst.dehaan@nl.wanadoo.com>

```
Georgios,

 I had a first look at you code.

Overall, it looks good. You even found some bugs in XINS as your remarks are
correct. For the SOAP HexBinary type, we just need to convert the output
parameters, I'll do it in the SOAPCallingConvention.

here are a few remarks
 - You should have made your modification based on the lastest CVS source
code. As you used XINS 1.3 to modify the XSLT file, the merge will be more
difficult as the XSLT files have already since since the 1.3 release (we're
now at XINS 1.4.1). I'll use the changes that you describe in the
Notes-modification.txt.
 - Be careful about the indentation of the files. The Java files contain
some mixure of spaces with tabs. Note that as standard we use 3 spaces
indentation. Hopefully I have an option in NetBeans to reformat the code.
  - I've also rename you test class HexTests in order to keep consistency
with the other tests.

 They will be a new demo that will use it, all of this will be in the XINS
1.5 alpha 2 release.

Anthony
```

4. Feature for contribution

XINS has some standard types which facilitate the use of input and output parameters at the implementation of a new project.
The developers proposed to create a new type Hex that will be available at the next release of the Xins (1.5.0). The Hex type will translate binary data (byte[]) to a Hexadecimal sting and vice versa.
That feature was published at XINS website in the feature request list. The feature code was 1483846, submitted by Anthony Goubard and proposed to the student.

Why a Hex type?

The main question that arises is why a hex type is needed. Here are the main reasons:

- Define a hexadecimal string (e.g. f12a0b9c8d4e)
- Used in XINS files .typ when we define a new type and in files .fnc at the input or output.
- A user may want to send binary information to a XINS-based API and prefers a hexadecimal encoding. So the hex type is chosen.
- A user wants a XINS API to return binary information across in hexadecimal format, with minimum and maximum length. In that case a new type that is based on hex is defined.

5. Implementation

Firstly a list of tools used for the implementation is mentioned and then the necessary modifications or additions to XINS files are explained.

5.1 Tools installed and used

For the implementation of the project several tools were used. Below is their list with a short description of their use in the project.

XINS 1.3 – the program was installed in order to view and understand its functionality. Finally it was used to test if the new class that was added works properly.
XINS 1.4.1 – in the meanwhile a new version was released and was also installed to work with it.
JUnit 4.1 - JUnit is a simple, open source tool to write and run repeatable tests. It was used to test the code of the new class.
Apache Ant 1.6.5 – is a tool for automating the process of compiling and running a project. XINS is based on Ant tool for compiling, deploying and running a new XINS project. Ant was also used for compiling and running the JUnit tests.
Java SDK 1.5.0

The source code of the program is available with the installation directory.

5.2 Modifications and Additions

1. A new java Class.
2. A new DTD file.
3. Existing XSLT files were modified.

5.2.1 A new Class

A new java Class named Hex was created.
The methodology followed for class Hex:

- Design the functions and their comments – create the Javadoc
- Send Javadoc to developers to review it
- Write code
- Compile
- Write unit tests, run the tests and check the code
- Send them to developers for final approval

The class Hex will be a part of the package org.xins.common.types.standard. The _hex type is similar to _base64 that converts a string to byte[] using base64 encoding. So, the implementation of the Hex class was similar to the Base64 class. The structure and functionality of the Base64 class was taken into account in order to start to create the Hex class.
The Hex class used three class functions of the class HexConverter in the package org.xins.common.text.

Functions used from Hexconverter class:

- isHexDigit (char)
  Checks if the specified character is a hexadecimal digit.
- parseHexBytes(String, int, int)
  Parses the specified string as a set of hex digits and converts it to a byte array.
- toHexString(byte[])
  Converts the specified byte array to an unsigned number hex string

Below I present the whole code of the Hex class including its javadoc comments that describe the functionality of the class. During the implementation I tried to conform to the conventions and programming style of the developers.
After writing the code of the Hex class the java file was compiled by running the command ant and using the build.xml file in the installed directory of XINS.

Figure 1: Hex class code

```
/*
 * Copyright 2003-2006 Wanadoo Nederland B.V.
 * See the COPYRIGHT file for redistribution and use restrictions.
 */
package org.xins.common.types.standard;

import org.xins.common.types.Type;
import org.xins.common.types.TypeValueException;
import org.xins.common.MandatoryArgumentChecker;
import org.xins.common.text.HexConverter;


/**
 * Standard type <em>_hex</em>.
 *
 * @version 1.0
 * @author gveiog
 *
 * @since XINS 1.5
 */

public class Hex extends Type {

   //-------------------------------------------------------------------------
   // Class fields
   //-------------------------------------------------------------------------

   /**
    * The only instance of this class. This field is never <code>null</code>.
    */

  public final static Hex SINGLETON = new Hex();

   //-------------------------------------------------------------------------
   // Class functions
   //-------------------------------------------------------------------------

   /**
    * Converts the specified non-<code>null</code> string value to a
    * <code>byte[]</code> value.
    *
    * @param string
    *    the hexadecimal string to convert, cannot be <code>null</code>.
    *
    * @return
    *    the <code>byte[]</code> value.
    *
    * @throws IllegalArgumentException
    *    if <code>string == null</code>.
    *
    * @throws TypeValueException
    *    if the specified string does not represent a valid value for this
    *    type.If the string does not have a hexadecimal value or have a character
    *     that is not hexadecimal digit.
    */

   public static byte[] fromStringForRequired(String string)
   throws IllegalArgumentException, TypeValueException {
      int index = 0;
      if (string == null) {
         throw new IllegalArgumentException("string == null");
      } else {
         try {// this method converts the string to byte and also checks if the string has hex
digits
                          return HexConverter.parseHexBytes(string,index,string.length());

                  } catch (Exception e){
                             throw new TypeValueException(SINGLETON, string);
                 }
      }
   }

   /**
    * Converts the specified string value to a <code>byte[]</code> value.
    *
    * @param string
    *    the hexadecimal string to convert, can be <code>null</code>.
    *
    * @return
    *    the byte[], or <code>null</code> if
    *    <code>string == null</code>.
    *
    * @throws TypeValueException
    *    if the specified string does not represent a valid value for this
    *    type.If the string does not have a hexadecimal value or
    *    have a character that is not hexadecimal digit.
    */
   public static byte[] fromStringForOptional(String string)
   throws TypeValueException {
           int index = 0;
      if (string == null) {
         return null;
      }
      try {
                 return HexConverter.parseHexBytes(string,index,string.length());

           } catch (Exception e){
                          throw new TypeValueException(SINGLETON, string);
           }
      }


   /**
    * Converts the specified <code>byte[]</code> to a hexadecimal string.
    *
    * @param value
    *    the value to convert, can be <code>null</code>.
    *
    * @return
    *    the textual representation of the value, or <code>null</code> if and
    *    only if <code>value == null</code>.
    */
   public static String toString(byte[] value) {
      if (value == null) {
         return null;
      } else {
         try {
            return HexConverter.toHexString(value);
         } catch (IllegalArgumentException iae) {
          with if(value==null)
            return null;
         }
      }
   }

   //------------------------------------------------------------------------
   // Constructors
   //------------------------------------------------------------------------

   /**
    * Constructs a new <code>Hex</code>.
    * This constructor is private, the field {@link #SINGLETON} should be
    * used.
    */
   private Hex() {
      this("_hex", 0, Integer.MAX_VALUE);
   }
   /**
    * Constructs a new <code>Hex</code> object (constructor for
    * subclasses).
         *
         * @param name
         *    the name of this type, cannot be <code>null</code>.
         *
         * @param minimum
         *    the minimum for the value.# minimum number of bytes this Hex can have
         *
         * @param maximum
         *    the maximum for the value.# maximum number of bytes this Hex can have
         */
         protected Hex(String name, int minimum, int maximum) {
            super(name, byte[].class);

            _minimum = minimum;
            _maximum = maximum;
         }
   //------------------------------------------------------------------------
   // Fields
   //------------------------------------------------------------------------

   /**
    * The minimum number of bytes this Hex can have.
    */
    private final int _minimum;
```

```
/**
 * The maximum number of bytes this Hex can have.
 */
 private final int _maximum;


//-------------------------------------------------------------------------
// Methods
//-------------------------------------------------------------------------

/**
 * Determines if the specified <code>String</code> value is considered
 * valid for this type (implementation method).
 *
 * <p>This method is called from {@link #isValidValue(String)}. When
 * called from that method, it is guaranteed that the argument is not
 * <code>null</code>.
 * # 1.check if the string has hex digits
 * # 2. if the byte[] created from that string has minimum < byte[].length < maximum
 * @param string
 *    the <code>String</code> value that should be checked for validity,
 *    never <code>null</code>.
 *
 * @return
 *    <code>true</code> if and only if the specified <code>String</code>
 *    value is valid, <code>false</code> otherwise.
 */
protected boolean isValidValueImpl(String string) {
   try {
      for (int i = 0; i < string.length(); i++) {
              if (!HexConverter.isHexDigit(string.charAt(i))) {
              return false;
              }
               }
      byte[] number = HexConverter.parseHexBytes(string,0,string.length());
      if (number.length < _minimum || number.length > _maximum) {
         return false;
      }
      return true;
   } catch (Exception ex) {
      // TODO: Log
      return false;
   }
}

/**
 * Converts from a <code>String</code> to an instance of the value class
 * for this type (implementation method).
 *
 * <p>This method is not required to check the validity of the specified
 * value (since {@link #isValidValueImpl(String)} should have been called
 * before) but if it does, then it may throw a {@link TypeValueException}.
 *
 * @param string
 *    the string to convert to an instance of the value class, guaranteed
 *    to be not <code>null</code> and guaranteed to have been passed to
 *    {@link #isValidValueImpl(String)} without getting an exception.
 *
 * @return
 *    an instance of the value class, cannot be <code>null</code>.
 *
 * @throws TypeValueException
 *    if <code>string</code> is considered to be an invalid value for this
 *    type.
 */
protected Object fromStringImpl(String string) throws TypeValueException {
   try {
      return HexConverter.parseHexBytes(string,0,string.length());
   } catch (Exception ex) {
      throw new TypeValueException(SINGLETON, string, ex.getMessage());
   }
}

/**
 * Generates a string representation of the specified value for this type.
 * The specified value must be an instance of the value class for this type
 * (see {@link #getValueClass()}). Also, it may have to fall within a
 * certain range of valid values, depending on the type.
 *
 * @param value
 *    the value, cannot be <code>null</code>.
 *
 * @return
 *    the string representation of the specified value for this type,
 *    cannot be <code>null</code>.
 *
```

```
     * @throws IllegalArgumentException
     *    if <code>value == null</code>.
     *
     * @throws ClassCastException
     *    if <code>getValueClass().isInstance(value) == false</code>.
     *
     * @throws TypeValueException
     *    if the specified value is not in the allowed range.
     *    #Maybe this is redundant because i dont think that there is any possibility to be out of
range
     *    but since it exists also to other standard types i left it.
     *
     */
   public final String toString(Object value)
   throws IllegalArgumentException, ClassCastException, TypeValueException {

      // Check preconditions
      MandatoryArgumentChecker.check("value", value);

      // Convert the argument to a byte array (may throw ClassCastException)
      byte[] b = (byte[]) value;

      // Try converting the byte array as a Hex string
      //isos na vgaleis to try
      try {
         return HexConverter.toHexString(b);
      } catch (Exception e) {

         throw new TypeValueException(SINGLETON, b.toString(), e.getMessage());
      }
         }
}
```

Unit tests

Unit tests are essential for testing the code of the Hex class. It will help check if the class works properly and gives the expected results. In that way it highlights possible problems in code that should be tackled. The tool JUNIT 4.1 was used for writing and running the appropriate tests. The unit test that was written for the Hex class is the following. It tests the constructor, the class functions and the methods of the Hex class. The code of the unit test is presented below.

Figure 2: Code of the unit test

```
package org.xins.common.types.standard;


import org.xins.common.types.standard.Hex;
import junit.framework.TestCase;
import junit.framework.TestSuite;
import junit.framework.Test;
import java.util.*;
import org.xins.common.types.TypeValueException;

public class HexTestNew extends TestCase {
    private Hex singleton = new Hex ("_hex", 0, Integer.MAX_VALUE);
    private Hex SINGLETON=Hex.SINGLETON;
         private String string;
         private String nullstring = null;


/*tests the contructor of the Hex class*/


public void testHexConstructor1() {

         //checks that singleton is not null
         assertTrue(!singleton.equals(null));

         //checks that SINGLETON is not null
         assertTrue(!SINGLETON.equals(null));


}

/*tests if fromStringForRequired() throws an IllegalArgumentException when the string is null*/


public void testfromStringForRequired1()
throws TypeValueException {
         try {
                          Hex.fromStringForRequired(nullstring);
```

```
                        fail("Expected IllegalArgumentException when arg is null");
        } catch (IllegalArgumentException iae) {
                        //ignore this because it means the test passed
        }
}

/*tests if fromStringForRequired() throws an TypeValueException when the string is not valid */


public void testfromStringForRequired2() {
        try {
                string = "1a2b3c4d5e6f7g";
                Hex.fromStringForRequired(string);
                fail("Expected TypeValueException when string is not valid");
        } catch (TypeValueException tve){
                //ignore this because it means the test passed
        }
}

/*tests if fromStringForOptional returns null when the parameter is null */


public void testfromStringForOptional()
throws TypeValueException {
        assertEquals(nullstring,Hex.fromStringForOptional(nullstring));
}

/*tests  if  the  class  functions  of  Hex  do  their  job,passing  a  hex  sting->byte[]->the  same  hex
string*/

/*NOTE FOR ERNST AND ANTHONY-this test fails if the string has odd chars because the method that
converts the byte
  to a hex string adds at the end of the string a 0.Actually this not a mistake,it is ok for but i
just wanted
  u to see it*/


public void testtoString1()
throws TypeValueException {
        //the string has odd chars
        string = "1a2b3c4d5e6f7";
        byte[]result = Hex.fromStringForRequired(string);
        assertEquals("toString should return the same hex",string, Hex.toString(result));
}

/*tests if toString() returns null when the parameter is null. Function toString() takes the null
byte[] value
  returned by fromStringOptional() and toString() returns also null*/

public void testtoString2()
throws TypeValueException {
        byte[]result = Hex.fromStringForOptional(nullstring);
        assertEquals(nullstring,Hex.toString(result));
}

/*tests if the method isValidValueImpl() returns false if the string is not hex*/


public void testisValidValueImpl1() {
        string = "12345adklm";
        assertTrue(!singleton.isValidValueImpl(string));
}

/*tests  if  the  method  isValidValueImpl()  returns  false  if  the  byte  is  not  at  at  the  specified
range*/


public void testisValidValueImpl2() {
        Hex single = new Hex("_hex", 0, 3);
        string = "12345ad";
        assertTrue("hex       is       expected       to       be       at       the       specified
range",!single.isValidValueImpl(string));
}


/*tests if the method isValidValueImpl() returns true for the correct hex string */


public void testisValidValueImpl3() {
        string = "12345adcef";
        assertTrue("the      string      is      expected      to      be      of      a      hex
type",singleton.isValidValueImpl(string));
}

/*tests if fromStringImpl() throws an TypeValueException when the string is not valid */
```

```
public void testfromStringImpl()
throws TypeValueException {
        try {
                string = "1a2b3c4d5e6f7g";
                Object object = singleton.fromStringImpl(string);
                fail("Expected TypeValueException when sting is not valid");
        } catch (TypeValueException tve) {
                //ignore this because it means the test passed
        }
}

/*tests if methods toString() and fromStringImpl() work properly*/
//THE TEST FAILS if the string has odd chars

public void testtoString_object()
throws TypeValueException {
        string = "1a2b3c4d5e6f72";
        Object object = singleton.fromStringImpl(string);
        assertEquals("toString should return the same string",string,singleton.toString(object));

}
/*tests if method toString() throws an exception when the parameter is null*/

public void testtoString_objectNotNull ()
throws IllegalArgumentException,TypeValueException {
        try {

                Object object = null;
                string = singleton.toString(object);
                fail("Expected IllegalArgumentException when parameter-object is null");
        } catch (IllegalArgumentException iae) {
                //ignore this because it means the test passed
        }
}


/*method for running the tests of the class*/

public static Test suite() {
        return new TestSuite(HexTestNew.class);
}

}
```

Ant was used to automate compiling of the unit test. The following build.xml file was used.

Figure 3: Build.xml

```
<project name="Compilation" default="compile" basedir=".">
    <description>
         build file for compiling java files at the current dir
    </description>
  <!-- set global properties for this build -->
  <path id="classpath.common">
        <pathelement path="org/xins/common/types/standard"/>
        <pathelement path="."/>
        <pathelement location="C:\junit4.1/junit-4.1.jar"/>
  </path>

  <property           name="src"           location="C:\xins-1.3.0/build/classes/java-
common/org/xins/common/types/standard"/>
  <property name="build" location=""/>

<target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

<target name="compile" depends="init" description="compile the source ">
    <!-- Compile the java code from ${src} into ${build} -->
    <javac
        srcdir="${src}"
        destdir="${build}">
        <classpath>
            <path refid="classpath.common" />
        </classpath>
    </javac>
</target>
```

```
</project>
```

5.2.2 A new DTD file

XINS uses DTD files to validate the XML based specifications that are written by the users. For the purposes of the new Hex feature a new dtd file was created in order to include the Hex type. The new DTD file was named type_1_5 and has the following content. It was created by adding the element Hex at the type_1_4.dtd file.

Figure 4: Code of type_1_5.DTD

```
<!--
 XINS Type DTD. This DTD should be added to all type files added to your
 project by adding the following lines at the top of the .typ file :
<!DOCTYPE type PUBLIC "-//XINS//DTD Type 1.5//EN" "http://www.xins.org/dtd/type_1_5.dtd">
-->
<!ELEMENT type (description, (pattern | enum | properties | int8 | int16 | int32 | int64 | float32
| float64 | base64 | hex | set | list)?)>
<!ATTLIST type
          name       NMTOKEN #REQUIRED
          rcsversion CDATA #IMPLIED
          rcsdate    CDATA #IMPLIED
>
<!ELEMENT description (#PCDATA|em)*>
<!ELEMENT em (#PCDATA)>
<!ELEMENT pattern (#PCDATA)>
<!ELEMENT enum (item*)>
<!ELEMENT item EMPTY>
<!ATTLIST item
          name    CDATA #IMPLIED
          value   CDATA #REQUIRED
>
<!ELEMENT properties EMPTY>
<!ATTLIST properties
          nameType     NMTOKEN #IMPLIED
          valueType    NMTOKEN #REQUIRED
>
<!ELEMENT int8 EMPTY>
<!ATTLIST int8
          min    NMTOKEN #IMPLIED
          max    NMTOKEN #IMPLIED
>
<!ELEMENT int16 EMPTY>
<!ATTLIST int16
          min    NMTOKEN #IMPLIED
          max    NMTOKEN #IMPLIED
>
<!ELEMENT int32 EMPTY>
<!ATTLIST int32
          min    NMTOKEN #IMPLIED
          max    NMTOKEN #IMPLIED
>
<!ELEMENT int64 EMPTY>
<!ATTLIST int64
          min    NMTOKEN #IMPLIED
          max    NMTOKEN #IMPLIED
>
<!ELEMENT float32 EMPTY>
<!ATTLIST float32
          min    NMTOKEN #IMPLIED
          max    NMTOKEN #IMPLIED
>
<!ELEMENT float64 EMPTY>
<!ATTLIST float64
          min    NMTOKEN #IMPLIED
          max    NMTOKEN #IMPLIED
>
<!ELEMENT base64 EMPTY>
<!ATTLIST base64
          min    NMTOKEN #IMPLIED
          max    NMTOKEN #IMPLIED
>
<!ELEMENT hex EMPTY>
<!ATTLIST hex
          min    NMTOKEN #IMPLIED
          max    NMTOKEN #IMPLIED
>
<!ELEMENT set EMPTY>
<!ATTLIST set
```

```
         type   NMTOKEN #IMPLIED
>
<!ELEMENT list EMPTY>
<!ATTLIST list
         type   NMTOKEN #IMPLIED
>
```

5.2.3 Modifications in existing XSLT files

XINS uses XSLT in order to generate skeleton java code. Also, XSLT is used to convert the result XML to text, HTML or other forms. For the new Hex feature the following four XSLT files were modified.
types.xslt - (provides templates to convert a type to something else.)
standard_types.xslt - (provides templates to convert a standard XINS type to something else.)
type_to_java.xslt - (generates the java representation of the type.)
type_to_html.xslt - (generates the type.html file that contains the description of the type.)

Figure 5: Modifications at the lines of the originals xslt files

```
@types.xslt
Line 264 comments add (_hex)
Line 319 add (<xsl:when test="$type_node/hex">
        <xsl:text>_hex</xsl:text>
        </xsl:when>)
Line 709 add(<xsl:when test="$paramtype = '_hex'">hexBinary</xsl:when>)


@standard_types.xslt
Line 34 add <xsl:when test="$type = '_hex'">Byte Array, Hex encoded.</xsl:when>
Line 66 add <xsl:when test="$type = '_hex'">byte[]</xsl:when>
Line 91 add <xsl:when test="$type = '_hex'">byte[]</xsl:when>

@type_to_java.xslt
Line 55 add <xsl:when test="hex">hex</xsl:when>
Line           72           add           <xsl:when           test="$kind           =
'hex'">org.xins.common.types.standard.Hex</xsl:when>

Line 601 <xsl:if test="$typeIsPrimary = 'fasle'"> it should be false
Line 473 <xsl:text>Integer.MIN_VALUE</xsl:text> I think it should be 0
Line 486 add <xsl:when test="$kind = 'hex'">
               <xsl:choose>
                       <xsl:when test="hex/@min">
                         <xsl:value-of select="hex/@min" />
                              </xsl:when>
                              <xsl:otherwise>
                                      <xsl:text>0</xsl:text>
                              </xsl:otherwise>
                       </xsl:choose>
                       <xsl:text>, </xsl:text>
               <xsl:choose>
                         <xsl:when test="hex/@max">
                              <xsl:value-of select="hex/@max" />
                              </xsl:when>
                              <xsl:otherwise>

       <xsl:text>Integer.MAX_VALUE</xsl:text>
                              </xsl:otherwise>
               </xsl:choose>
               </xsl:when>

@type_to_html.xslt
Line 103  add <xsl:apply-templates select="hex"        />
Line 216 add | hex
```

5.3 Final test

Finally after completing the steps above the student performed a final test to check the overall fiunctionality of XINS with the new Hex type. All the new files were inserted in the \src file of the XINS's installation directory. With the command ant the whole program was build again from the beginning. Then in order to check if the type Hex works properly in XINS, the demo API xins *myproject*, distributed with XINS, was used. In *myproject* a new function was created having as an input a hex type and as an output a new defined type that is based on Hex e.g. (Hex with min and max value). The *myproject* was build again with the command *xins war myproject*. The generated code expected was created and the function was implemented right. This means that the Hex type was integrated in XINS and works properly.

6. Conclusion

Participating in XINS project was a really interesting experience. Although working in a big project in not always easy and often having to go into deep waters quite early, it is really challenging. Looking back you can only see good things like improving your code reading, your programming skills and experience by cooperating with talented professionals. Open source community thrives and is here to stay. Above all is the joy of creating something.

7. Bibliography

1. XINS website.
   http://xins.sourceforge.net/index.html
2. XINS user's guide
   Available at http://xins.sourceforge.net/docs/XINSGuide.pdf