**Overview**

This sample code demonstrates how to convert a set of related CSV files into a SLI-EdFi xml. The latest Ed-Fi schema can be found here: http://www.ed-fi.org/wp-content/uploads/2011/06/Ed-Fi-Core-1.0.03.xsd\_.zip

In this example, we convert data from csv files (Student.csv, StudentAddress.csv, StudentLanguage.csv, Parent.csv and StudentParentAssociation.csv) into an xml file consisting of student, parent and student-parent-association entities.

All sample csv files are located in the data folder in this sample project.

Those files only capture the data that can be supported by the SLI-EdFi schema. If the csv file contains unsupported data, it will be ignored when the xml is generated.

**Some assumptions on the CSV files:**

1. **Records in the csv files are sorted. For example, records in student.csv, studentAddress.csv and studentLanguage.csv are sorted by studentUSI in ascending order, and records in parent.csv are sorted by ParentUSI.**
2. **References, e.g. StudentUSI and ParentUSI, in studentParentAssociation.csv are legitimate and exist in both student.csv and parent.csv. This sample code does not check the reference integrity across csv files.**

**Sample Project Structure**

The developer can download the full Sample project from here: (...), and unzip it into a folder, such as "c:\SLI\_Project". The developer can import it into Eclipse as an existing java project. This project is built and tested using java 1.6.0\_31 and Eclipse Indigo with SP1.

The project structure as shown below:

**project structure**

|  |
| --- |
| --csv2xml      .classpath      .project      --src          --org              --slc                  --sli                      --sample                          --entities                          --transform      --data      --edfiXsd |

* ***csv2xml*** is the root folder for the project.
* ***src/org/slc/sli/sample/entities*** contains JAXB java source code for all SLI-EdFi entities.
* ***src/org/slc/sli/sample/transform*** contains the source code of the CSV2XMLTransformer
* ***data*** contains the sample csv files for this project.
* ***edfiXsd*** contains the SLI-EdFi schema xsd files.

**Sample csv files**

**Student.csv**

|  |
| --- |
| StudentUSI,Verification,PersonalTitlePrefix,FirstName,MiddleName,LastSurname,GenerationCodeSuffix,Sex,BirthDate,ProfileThumbnail,HispanicLatinoEthnicity,RacialCategory  100000,Previously verified school records,Mr,Brett,Sergio,Townsend,,Male,1997-13-03,,1,Asian  100001,Other official document,Mr,Lee,Rick,Yang,,Male,2000-13-30,thumbnail,1,American Indian - Alaskan Native  100002,Previously verified school records,Ms,Angelique,Ann,Meyer,,Female,2001-13-20,,1,White |

**StudentAddress.csv**

|  |
| --- |
| StudentUSI,AddressType,StreetNumberName,City,StateAbbreviation,PostalCode,NameOfCounty  100001,Physical,477 Ash Street,JAMAICA,NY,11433,Queens  100001,Other,520 Hill Street,BOONVILLE,NY,13309,Oneida  100002,Billing,582 Valley Street,NORWOOD,NY,13668,Saint Lawrence |

**StudentLanguage.csv**

|  |
| --- |
| StudentUSI,Language  100001,Spanish  100001,English  100002,Turkish |

As shown in the three csv files above, three students, e.g. 100000, 100001, 100002, are listed.

1. Student 100000 has no address and does not know any languages.
2. Student 100001 has two addresses and knows two languages: Spanish and English.
3. Student 100002 has one addresses and knows one language: Turkish.

As we see in the sample student csv files, if a value, e.g. FirstName or LastName, only appears once for each student, then that value can/should be captured by the main file, e.g. Student.csv. If there are multiple values associated with one student, such as address and language, they should be captured by a separate csv file, e.g. StudentAddress.csv and StudentLanguage.csv.

**Parent.csv**

|  |
| --- |
| ParentUSI,Verification,PersonalTitlePrefix,FirstName,MiddleName,LastSurname,GenerationCodeSuffix,MaidenName,Sex  900000,Other non-official document,Mr,Tracy,Marc,Pugh,,,Male  900001,Parents affidavit,Ms,Louisa,Christine,Mullins,,,Female  900002,State-issued ID,Mr,Allan,Allan,Edwards,Jr,,Male |

Three parents are listed in this sample csv file.

**StudentParentAssociation.csv**

|  |
| --- |
| StudentUSI,ParentUSI,Relation,PrimaryContactStatus,LivesWith,EmergencyContactStatus  100000,900000,Father,1,0,0  100000,900001,Mother,0,0,0  100001,900002,Father,0,1,1 |

Student 100000 has both parents: Parent 900000 and Parent 900001.  
Student 100001 has one parent: Parent 900002.

**How the sample code works?**

**Prepare JAXB classes**

The source code in org.slc.sli.sample.entities package are generated by a JAXB utility (xjc.sh/xjc.exe). The generated classes are the ed-fi schema entities that you choose to support, represented as Java classes. If the EdFi schema is updated, the developer needs to regenerate the source code for this package again.

Here are the steps to do it:

1. Follow the link http://www.ed-fi.org/wp-content/uploads/2011/06/Ed-Fi-Core-1.0.03.xsd\_.zip to download and replace all schema files in edfiXsd folder in this project.
2. Open a cmd/shell Window, and go to edfiXsd folder, then execute the command "***xjc -b Ed-Fi-Core.xjb -p org.slc.sli.sample.entities .***"
3. Replace all java code in org.slc.sli.sample.entities with the newly generated code from step two.
4. Update Adapter2.java with the following code.

|  |
| --- |
| //  // This file was generated by the JavaTM Architecture for XML Binding(JAXB) Reference Implementation, vJAXB 2.1.10 in JDK 6  // See <a href=">  // Any modifications to this file will be lost upon recompilation of the source schema.  // Generated on: 2012.04.20 at 03:09:04 PM EDT  //    package org.slc.sli.sample.entities;    import java.util.Calendar;  import javax.xml.bind.annotation.adapters.XmlAdapter;    public class Adapter2      extends XmlAdapter<String, Calendar>  {      public Calendar unmarshal(String value) {          return (javax.xml.bind.DatatypeConverter.parseDate(value));      }        public String marshal(Calendar value) {          if (value == null) {              return null;          }          return (javax.xml.bind.DatatypeConverter.printDateTime(value).substring(0, 10));      }  } |

**Java Class Code**

**CSVReader.java**

The code reads records from csv file one by one and converts them into a record (hashmap) of the columnname/value pairs.  
This code also contains a utility method, getDate(...), that converts a date string (in the format of "yyyy-mm-dd") into Calendar object. The developer may need to update method according to the date format in his data file.

**SchemaValidator.java**

This code validates the generated xml file against the SLI-EdFi schema.

**CSV2XMLTransformer.java**

This is the main part of this transformer sample code. It converts records into Jaxb java objects and marshals them into a SLI-EdFi xml file.

**Student Conversion**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | private Student getStudent() {          Map<String, String> studentRecord = studentReader.getCurrentRecord();            Student student = new Student();            // set id          String studentId = studentRecord.get("StudentUSI");          student.setStudentUniqueStateId(studentId);          student.setId("STUD\_" + studentId);            // set name          student.setName(this.getName(studentRecord));            // set sex          student.setSex(SexType.fromValue(studentRecord.get("Sex")));            // set birthData          BirthData birthData = new BirthData();          birthData.setBirthDate(CSVReader.getDate(studentRecord.get("BirthDate")));          student.setBirthData(birthData);            // set ProfileThumbnail          String profile = studentRecord.get("ProfileThumbnail");          if (!profile.isEmpty()) {              student.setProfileThumbnail(profile);          }            // set HispanicLatinoEthnicity          student.setHispanicLatinoEthnicity(studentRecord.get("HispanicLatinoEthnicity") == "1" ? true : false);            // set raceType          String racial = studentRecord.get("RacialCategory");          if (!racial.isEmpty()) {              RaceType raceType = new RaceType();              raceType.getRacialCategory().add(RaceItemType.fromValue(racial));              student.setRace(raceType);          }            // set addresses          addStudentAddresses(student);            // set languages          addStudentLanguages(student);            return student;      } |

* lines 7-9 show where to populate StudentUSI
* line 12 shows how to set student name, which is constructed by the following code

|  |
| --- |
| private Name getName(Map<String, String> nameRecord) {          Name name = new Name();            String verification = nameRecord.get("Verification");          if (!verification.isEmpty()) {              name.setVerification(PersonalInformationVerificationType.fromValue(verification));          }            String prefix = nameRecord.get("PersonalTitlePrefix");          if (!prefix.isEmpty()) {              name.setPersonalTitlePrefix(PersonalTitlePrefixType.fromValue(prefix));          }            name.setFirstName(nameRecord.get("FirstName"));            String middleName = nameRecord.get("MiddleName");          if (!middleName.isEmpty()) {              name.setMiddleName(middleName);          }            name.setLastSurname(nameRecord.get("LastSurname"));            String generation = nameRecord.get("GenerationCodeSuffix");          if (!generation.isEmpty()) {              name.setGenerationCodeSuffix(GenerationCodeSuffixType.fromValue(generation));          }            return name;      } |

* line 15 shows how to set an Enum value that is defined by the SLI-EdFi schema.
* lines 18-20 show how to set BirthData that only contains one required field: BirthDate.
* lines 23-26 show how to set an optional field, ProfileThumbnail. Ignore the setting if there is no value for it.
* line 29 shows how to convert "0"/"1" value into true/false.
* lines 32-37: according to the SLI-EdFi schema, 'racial' has multiple values. In this example, we assume that each student only has one value. If a student has multiple values for this field, please follow the language approach below.
* line 40 shows how to handle multiple addresses for a student. When the student has multiple addresses, the address info needs to be captured by a separate file (e.g. studentAddress.csv). The StudentUSI is the key to associate StudentAddress.csv with Student.csv. As we stated earlier, the records in both files must be sorted based on StudentUSI in ascending order.

|  |
| --- |
| private void addStudentAddresses(Student student) {          String studentId = student.getStudentUniqueStateId();            while (studentAddressReader.getCurrentRecord() != null) {              Map<String, String> studentAddressRecord = studentAddressReader.getCurrentRecord();                String id = studentAddressRecord.get("StudentUSI");              if (id.compareTo(studentId) > 0) {                  // if the studentUSI of the address record is larger than studentUSI of the student                  // which means the current address record belongs to next student, not the current one.                  break;              } else if (id.equals(studentId)) {                  student.getAddress().add(this.getAddress(studentAddressRecord));              }                studentAddressReader.getNextRecord();          }      } |

* line 43 shows how to handle multiple Languages for one student. Similar to address, multiple language info needs to be captured by a separate file, StudentLanguage.csv. Language list is added to the student object only at least one language exists for a student.

|  |
| --- |
| private void addStudentLanguages(Student student) {          String studentId = student.getStudentUniqueStateId();            LanguagesType languages = new LanguagesType();          while (studentLanguageReader.getCurrentRecord() != null) {              Map<String, String> studentLanguageRecord = studentLanguageReader.getCurrentRecord();                String id = studentLanguageRecord.get("StudentUSI");              if (id.compareTo(studentId) > 0) {                  // if the studentUSI of the language record is larger than studentUSI of the student                  // which means the current language record belongs to next student, not the current one.                  break;              } else if (id.equals(studentId)) {                  String ls = studentLanguageRecord.get("Language");                  if (!ls.isEmpty()) {                      LanguageItemType language = LanguageItemType.fromValue(ls);                      languages.getLanguage().add(language);                  }              }                studentLanguageReader.getNextRecord();          }          if (languages.getLanguage().size() > 0) {              student.setLanguages(languages);          }      } |

**Parent Conversion**

Parent Conversion is very similar to the student conversion but simpler.

|  |
| --- |
| private Parent getParent() {          Map<String, String> parentRecord = parentReader.getCurrentRecord();            Parent parent = new Parent();            // set Id          String parentId = parentRecord.get("ParentUSI");          parent.setParentUniqueStateId(parentId);          parent.setId("PRNT\_" + parentId);            // set name          parent.setName(this.getName(parentRecord));            // set sex          parent.setSex(SexType.fromValue(parentRecord.get("Sex")));            return parent;      } |

**StudentParentAssociation Conversion**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | private StudentParentAssociation getStudentParentAssociation() {          Map<String, String> studentParentAssociationRecord = studentParentAssociationReader.getCurrentRecord();            StudentParentAssociation studentParentAssociation = new StudentParentAssociation();            // set student reference          StudentIdentityType sit = new StudentIdentityType();          sit.setStudentUniqueStateId(studentParentAssociationRecord.get("StudentUSI"));          StudentReferenceType srt = new StudentReferenceType();          srt.setStudentIdentity(sit);          studentParentAssociation.setStudentReference(srt);            // set parent reference          ParentIdentityType pit = new ParentIdentityType();          pit.setParentUniqueStateId(studentParentAssociationRecord.get("ParentUSI"));          ParentReferenceType prt = new ParentReferenceType();          prt.setParentIdentity(pit);          studentParentAssociation.setParentReference(prt);              // set relation          if (!studentParentAssociationRecord.get("Relation").isEmpty()) {              studentParentAssociation                      .setRelation(RelationType.fromValue(studentParentAssociationRecord.get("Relation")));          }            // set primary contact status          String primaryContact = studentParentAssociationRecord.get("PrimaryContactStatus");          if (!primaryContact.isEmpty()) {              studentParentAssociation.setPrimaryContactStatus(primaryContact.equals("1") ? true : false);          }            // set lives with          String livesWith = studentParentAssociationRecord.get("LivesWith");          if (!livesWith.isEmpty()) {              studentParentAssociation.setLivesWith(livesWith.equals("1") ? true : false);          }            // set emergency contact status          String emergencyContact = studentParentAssociationRecord.get("EmergencyContactStatus");          if (!emergencyContact.isEmpty()) {              studentParentAssociation.setEmergencyContactStatus(emergencyContact.equals("1") ? true : false);          }            return studentParentAssociation;      } |

* lines 7-11 show how to set studentReference for a StudentParentAssociation. In this example we use StudentIdentityType rather than ID/IDReF pair. Because StudentIdentityType is more flexible than the ID/IDRef, which requires both Student and StudentParentAssociation exist in the same xml file.
* lines 14-18 show how to set parentReference for a studentParentAssociation, which is similar to the previous case.

**Print xml file**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | private void printInterchangeStudentParent(PrintStream ps) throws JAXBException {          int studentCounter = 0;          int parentCounter = 0;          int studentParentAssociationCounter = 0;            Marshaller marshaller = getMarshaller();            InterchangeStudentParent interchangeStudentParent = new InterchangeStudentParent();          List<Object> list = interchangeStudentParent.getStudentOrParentOrStudentParentAssociation();            // process student          while (studentReader.getCurrentRecord() != null) {              list.add(this.getStudent());              studentReader.getNextRecord();              studentCounter++;          }            // process parent          while (parentReader.getCurrentRecord() != null) {              list.add(this.getParent());              parentReader.getNextRecord();              parentCounter++;          }            // process studentParentAssociation          while (studentParentAssociationReader.getCurrentRecord() != null) {              list.add(this.getStudentParentAssociation());              studentParentAssociationReader.getNextRecord();              studentParentAssociationCounter++;          }            marshaller.marshal(interchangeStudentParent, ps);            System.out.println("Total " + studentCounter + " students are exported.");          System.out.println("Total " + parentCounter + " parents are exported.");          System.out.println("Total " + studentParentAssociationCounter                  + " student-parent-associations are exported.");          System.out.println("Total " + ( studentCounter + parentCounter + studentParentAssociationCounter)                  + " entities are exported.");      } |

* lines 12-16, 19-23 and 26-30 go through csv files, generate all students, parents and studnetParentAssociations and add them into a list.
* line 32 marshals the list into an xml file.
* lines 34-39 generate a report of the csv to xml conversion.

After the xml file is generated, the main method calls SchemaValidator to validate the file against the schema.

|  |
| --- |
| public static void main(String[] args) throws Exception {          CSV2XMLTransformer transformer = new CSV2XMLTransformer();          transformer.loadData();            PrintStream ps = new PrintStream(new File(interchangeStudentParentFile));          transformer.printInterchangeStudentParent(ps);            SchemaValidator.check(outputPath);      } |

**How to Support new Entities**

We will use Teacher as an example to guide the steps of adding support to a new entity.

1. Generate a set of teacher-related csv file.
   1. All csv files are sorted by teacherId, which uniquely identifies each teacher, in ascending order.
   2. Make sure all data required by SLI-Edfi schema (e.g. Ed-Fi-Core-XXXXX.xsd) for a Teacher entity is captured by the csv files.
   3. Understand the mapping from the columns in the csv files to the fields of the Teacher entity.
2. Find the correct Interchange-XXXXX.xsd that contains the Teacher entity. In this case it is Interchange-StaffAssociation.xsd.
3. Follow the student approach to construct the teacher entities from the csv files, and generate interchangeStaffAssociation.xml file.