Title:

Deliverable D5: XML for AIM

Comment:

This work was done for the IDA-STEP project, Funded by the European Commission

For internal use within the IDA-STEP project only

Author:

Vaidas Nargėlas, UAB LKSoft Baltic

File-name: D5-1-v1-doc

Version	Date	Status	Evolutions
V1	**2002-09-20**	**Final**	**Created**

Introduction

A JSDAI extensions to support XML (JSDAI-XML) was designed to achieve the following goals:

- To implement all bindings defined in ISO/PRF TS 10303-28 (ISO TC184/SC4/WG11 N198 2002-07-16)
- To allow flexible ways to import/export data to/from STEP databases using XML
- To make XML operations integrated with JSDAI
- To follow other XML APIs (JAXP, TRaX) as close as possible

JSDAI-XML was realized within the IDA-STEP project as a prototype implementation thus achieving most of the goals above.

Overview

ISO 10303-28 (Part28) defines how to represent EXPRESS schemas and data in XML. A late binding (LB) and two early bindings (ETEB, OSEB) are specified to represent EXPRESS driven data. EXPRESS schemas are represented as SCHEMA_DECL or SCHEMA_TEXT. Another early binding for express driven data is defined outside of the standard (CEB). All these bindings are based on DTD. A second edition of Part28 is under preparation using XML schemas.

The JSDAI-XML was designed as a general solution for XML support for JSDAI which makes it easy to adopt for all bindings mentioned above as well as other representations. Therefore the result of this work will also be the basis for the deliverable **XML for ARM** (D6). JSDAI-XML is currently implemented and tested for the late binding representation (LB) only. Other bindings will be added later as needed.

JSDAI-XML is designed as several SAX compatible classes which work directly on data in JSDAI. It works as on the fly bridge between JSDAI and XML. This way makes using XML technologies simple and straightforward. Reading of XML data from JSDAI works as an implementation of the SAX XMLReader interface, and writing of XML data to JSDAI acts as ContentHandler. XML operations were tested in using JUnit test suite, writing test XSLT stylesheets, and adding XML import/export to the StepBook user interface.

Operations

XML support contains three of operations: high level, low level, and XSLT support.

- High level operations can be used for import from/export to a file or other stream
- Low level operations allows fine grained control on bridging SDAI to XML. They provide interfaces for using STEP data in XML applications.
- XSLT support provides extension functions and elements to be used in stylesheets.

High level

The High level operations are integrated in the JSDAI-API, so that they can be used by any JSDAI application to import and export XML steams. The following additional methods where added to Java classes of the jsdai.lang package:

`SdaiRepository.exportXml`
: Exports SDAI repository to output stream

`SdaiRepository.importXml`
: Exports SDAI repository to output stream

`SdaiModel.exportXml`
: Exports SDAI model to output stream

`ASdaiModel.exportXml`
: Exports collection of SDAI model to output stream

`SchemaInstance.exportXml`
: Exports SDAI schema instance to output stream

`ASchemaInstance.exportXml`
: Exports collection of SDAI schema instances to output stream

Low level

The Low level operations are for advanced users, which needs a closer integration on the basis of the JAXP API. The following classes are provided in package jsdai.xml:

`InstanceReader`
: Abstract class which acts as SAX XMLReader of STEP instances

`InstanceWriter`
: SAX ContentHandler which writes STEP instances from parsing events

`LateBindingReader`
: ISO-10303-28 late binding representation reader. This class extends InstanceReader

`SdaiInputSource`
: Extension of SAX InputSource which can specify SdaiRepository, SdaiModel, SchemaInstance, ASdaiModel, and ASchemaInstance

XSLT support

These operations consist of the following classes in package jsdai.xml:

`SdaiXmlBridge`
: Entry point to XSLT extension functions and elements.

`SdaiSessionReader`
: SAX XMLReader which represents active SDAI session (repository list) as XML

`RepositoryReader`
: SAX XMLReader which represents SDAI repository (schema instance and model lists) as XML

Application

Using in Java programs

This portion of a JUnit test demonstrates importing and late binding exporting of XML data into JSDAI:

```java
public void testImportAp214() throws SdaiException, IOException {
    FileInputStream fileStream =
        new FileInputStream("testExportAp214-1.xml");
    InputStream fromStream = new BufferedInputStream(fileStream);
    try {
        repository.importXml(fromStream);
    }
    finally {
        fromStream.close();
    }
}


public void testExportAp214() throws SdaiException, IOException {
    LateBindingReader reader = new LateBindingReader();
    reader.setFeature("schema-population-names", true);
    reader.setFeature("make-schema-populations", true);
    FileOutputStream fileStream =
        new FileOutputStream("testExportAp214-1.xml");
    OutputStream toStream =
        new BufferedOutputStream(fileStream, 4096);
    try {
        repository.exportXml(toStream, reader);
    }
    finally {
        toStream.close();
    }
}
```

Using in XSLT

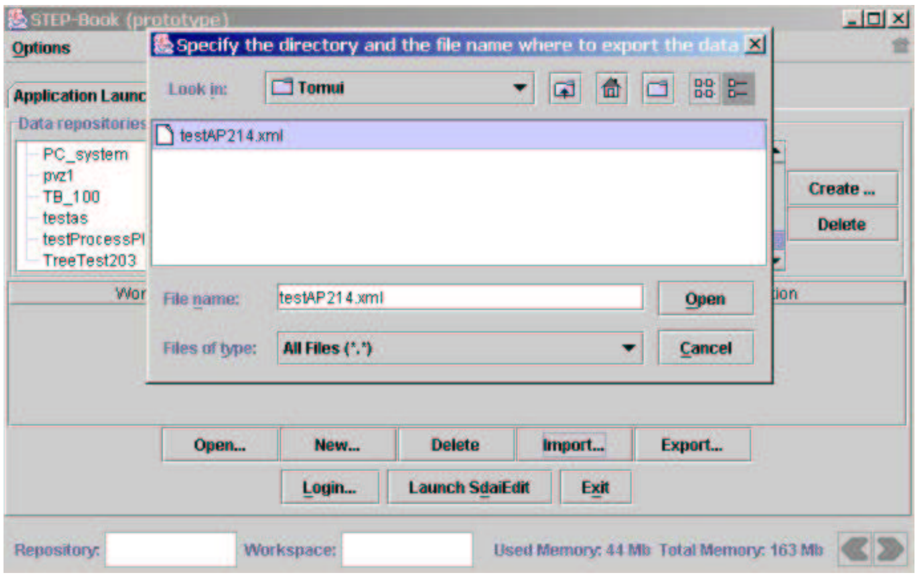
This example XSL stylesheet demonstrates accessing JSDAI-XML. The stylesheet exports SDAI model with the name `default` of the repository `cityBike` to late binding XML file.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:exsl="http://exslt.org/common"
                xmlns:jsdai="class://jsdai.xml.SdaiXmlBridge"
                xmlns:xalan="http://xml.apache.org/xslt"
                extension-element-prefixes="jsdai exsl"
                exclude-result-prefixes="xalan"
                version="1.0">

  <xsl:output method="xml" indent="yes" xalan:indent-amount="2"/>

  <xsl:variable name="jsdai-session" select="jsdai:open_session()"/>
  <xsl:variable name="jsdai-repository"
    select="jsdai:get_repository($jsdai-session, 'cityBike')"/>
  <xsl:variable name="jsdai-model"
    select="jsdai:get_model($jsdai-repository, 'default')"/>

  <xsl:template match="/">
    <xsl:variable name="query">
      <model var="jsdai-model"/>
    </xsl:variable>
    <xsl:variable name="query-result"
      select="jsdai:get_instances($query)"/>
    <xsl:copy-of select="$query-result"/>
  </xsl:template>
</xsl:stylesheet>
```

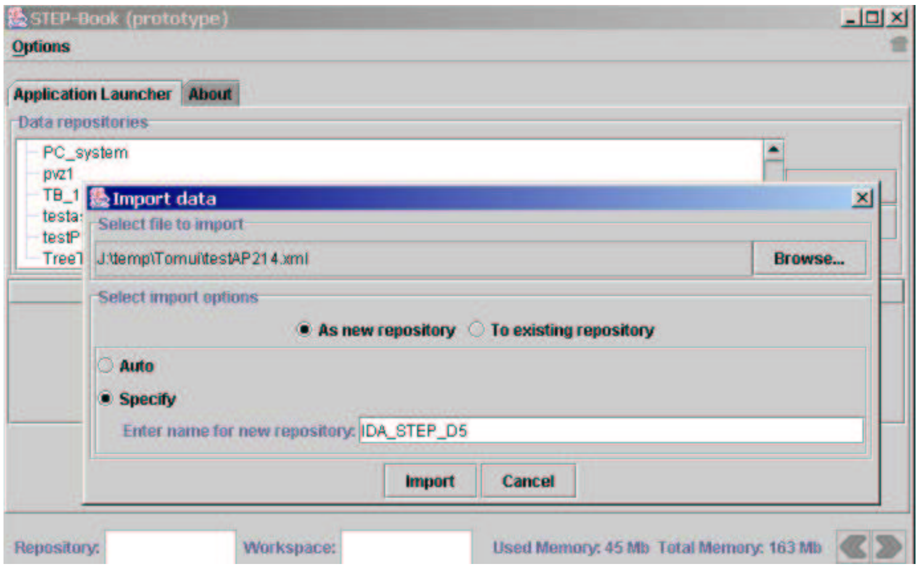
Using in StepBook

Importing from XML Part28 late binding file.

1. Choose the file you want to import:



2. Enter repository name you want STEP data to be imported to:


STEP-Book (prototype)

Options

Application Launcher | About

Data repositories: PC_system, pvz1, TB_1, testa, testP, TreeT

Import data

Select file to import: J:\temp\Tomui\testAP214.xml [Browse...]

Select import options: [✓ As new repository] To existing repository

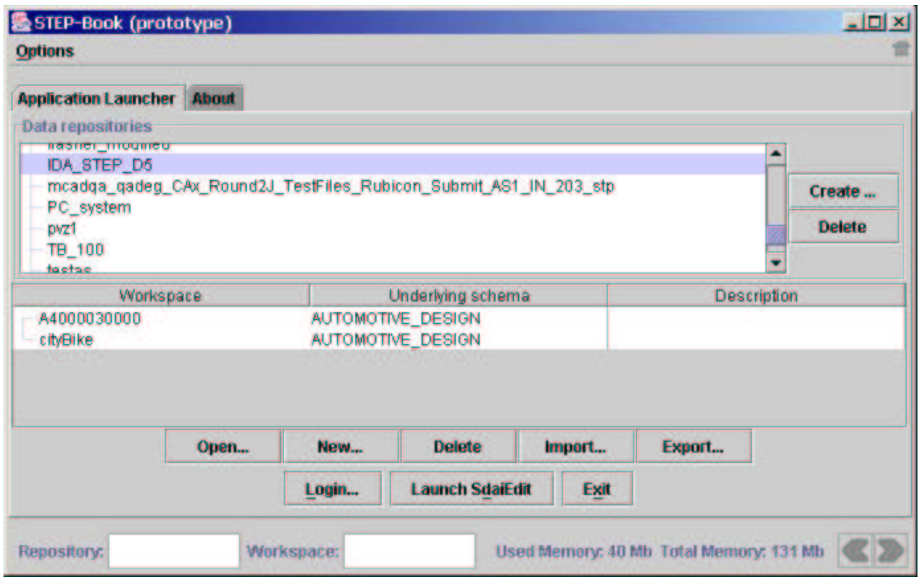
Auto [✓ Specify]

Enter name for new repository: IDA_STEP_D5

[Import] [Cancel]

Repository: Workspace: Used Memory: 45 Mb Total Memory: 163 Mb

3. Listing schema instances in the newly imported repository:


STEP-Book (prototype)

Options

Application Launcher | About

Data repositories:

IDA_STEP_D5
mcadqa_qadeg_CAx_Round2J_TestFiles_Rubicon_Submit_AS1_IN_203_stp
PC_system
pvz1
TB_100

[Create ...] [Delete]

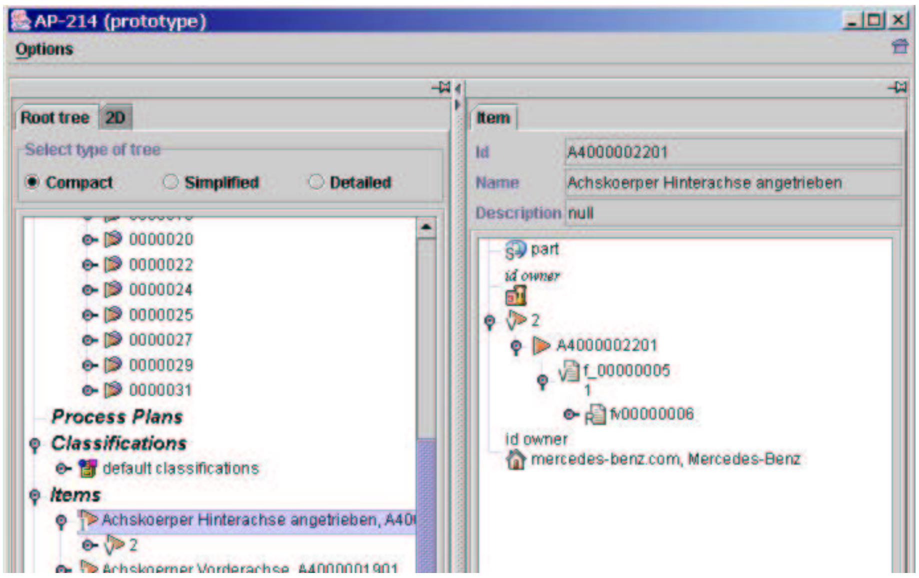
Workspace	Underlying schema	Description
A4000030000	AUTOMOTIVE_DESIGN	
cityBike	AUTOMOTIVE_DESIGN	

[Open...] [New...] [Delete] [Import...] [Export...]

[Login...] [Launch SdaiEdit] [Exit]

Repository: Workspace: Used Memory: 40 Mb Total Memory: 131 Mb

4. Open the book for selected schema instance:



Limitations/To-do

- Complex entity import/export is not done in compliance with Part28.
- Other Part28 mappings are not implemented and only partially supported by current class structure.
- External references are not supported. External references could be used for referencing existing instance in SDAI repositories.
- XML import operations should return the list of created SchemaInstances and SdaiModels.