How to implement export and import tool content

There are 2 methods in ToolContentManager to support tool export and import tool content. One is void exportToolContent(Long, String). Another is void importToolContent(Long , Integer , String). Following description is how to implement them with helper methods from LAMS ExportToolContentService.

Export tool content

In exportToolContent method, the work need to are

- Sort out what content is need to be exported, what is not necessary to exported
- Serialize content to XML file. The default name conversion is tool.xml which is under a subfolder which named by given toolContentId.
- Download all relevant attachment files from LAMS repository and save them into same folder with tool.xml, and their file name will be fileUuid rather than the original file name to avoid conflict.

Hibernate is data access API in LAMS tools. So we can easily reuse hibernate model class as content object and serialize it. To achieve this, ExportToolContentService in LAMS common package provides helper methods to make things easier. Usually, it needs 2 helper methods from this service bean: One is registerFileClassForExport(). Another is exportToolContent(). The former can tell service bean how to find out the attachment files information. The latter one can serialize content object to XML. Furthermore, it could retrieve content object and download all attachment files to target folder according to the register information in last step.

Below is two examples for export tool content. Notes:

- To detach useless object for content, e.g., Tool Session, ToolContentHandler etc. their value need to be set to null.
- If any subobjects (e.g. attachments) contain a link back to the main object, then clear the link to the main object. For example, the ScribeHeading has a back link to Scribe and this must be cleared.

```
public void exportToolContent(Long toolContentId, String rootPath) throws DataMissingException, ToolException {
                Resource toolContentObj = resourceDao.getByContentId(toolContentId);
                 if(toolContentObj == null)
                         throw new DataMissingException("Unable to find tool content by given id :" +
toolContentId);

                 //set ResourceToolContentHandler as null to avoid copy file node in repository again.
                 toolContentObj = Resource.newInstance(toolContentObj,toolContentId,null);
                //detach useless information from tool content object
                toolContentObj.setToolContentHandler(null);
                 toolContentObj.setOfflineFileList(null);
                 toolContentObj.setOnlineFileList(null);
                 toolContentObj.setMiniViewNumberStr(null);
                try {
                        exportContentService.registerFileClassForExport(ResourceAttachment.class.getName(),"
fileUuid","fileVersionId");
                        exportContentService.registerFileClassForExport(ResourceItem.class.getName(),"
fileUuid","fileVersionId");
                        exportContentService.exportToolContent( toolContentId, toolContentObj,
resourceToolContentHandler, rootPath);
                } catch (ExportToolContentException e) {
                        throw new ToolException(e);
                }
        }
```

```
public void exportToolContent(Long toolContentId, String rootPath)
                throws DataMissingException, ToolException {
        Scribe scribe = scribeDAO.getByContentId(toolContentId);
        if (scribe == null) {
                scribe = getDefaultContent();
        }
        if (scribe == null)
                throw new DataMissingException("Unable to find default content for the scribe tool");

        // set ResourceToolContentHandler as null to avoid copy file node in
        // repository again.
        scribe = Scribe.newInstance(scribe, toolContentId, null);
        scribe.setToolContentHandler(null);
        scribe.setScribeSessions(null);
        // wipe out the links from ScribeAttachments, ScribeHeading back to Scribe, or it will try to
        // include the hibernate object version of the Scribe within the XML
        Set<ScribeAttachment> atts = scribe.getScribeAttachments();
        for (ScribeAttachment att : atts) {
                att.setScribe(null);
        }
        Set<ScribeHeading> headings = scribe.getScribeHeadings();
        for (ScribeHeading heading : headings) {
                heading.setScribe(null);
        }
        try {
                exportContentService.registerFileClassForExport(ScribeAttachment.class.getName(),
                                                "fileUuid", "fileVersionId");
                exportContentService.exportToolContent(toolContentId,
                                scribe, scribeToolContentHandler, rootPath);
        } catch (ExportToolContentException e) {
                throw new ToolException(e);
        }
```

Import tool content.

It is a reverse process with export content.

- Deserialize tool.xml to content object
- Upload attachment files to LAMS repository and refresh file uuid, version, type (ONLINE/OFFLINE) etc.
- Refresh any user information to new user. The new user usually be the person who handle importing.
- Refresh toolContentId.

There are helper methods from ExportToolContentService bean. They registers imported attachment files information and deserializes tool XML file to content object. The helper method, importToolContent(), also upload attachment files and refresh relevant fields in content object.

Except that, tool need refresh the user to the given user (newUserUid). The toolContentId need refresh as well. Below is example code:

```
public void importToolContent(Long toolContentId, Integer newUserUid, String toolContentPath, String
fromVersion, String toVersion)
    throws ToolException
{

        try {
                exportContentService.registerFileClassForImport(ResourceAttachment.class.getName()
                                ,"fileUuid","fileVersionId","fileName","fileType",null,null);
                exportContentService.registerFileClassForImport(ResourceItem.class.getName()
                                ,"fileUuid","fileVersionId","fileName","fileType",null,"initialItem");
                   exportContentService.registerImportVersionFilterClass(RsrcImportContentVersionFilter.class);

                Object toolPOJO =  exportContentService.importToolContent(toolContentPath,
resourceToolContentHandler,fromVersion,toVersion);
                if(!(toolPOJO instanceof Resource))
                        throw new ImportToolContentException("Import Share resources tool content failed.
Deserialized object is " + toolPOJO);
                Resource toolContentObj = (Resource) toolPOJO;

        //                      reset it to new toolContentId
                toolContentObj.setContentId(toolContentId);

        //                      Refresh user information for tool content object
                ResourceUser user = resourceUserDao.getUserByUserID(new Long(newUserUid.longValue()));
                if(user == null){
                        user = new ResourceUser();
                        UserDTO sysUser = userManagementService.getUserById(newUserUid).getUserDTO();
                        user.setFirstName(sysUser.getFirstName());
                        user.setLastName(sysUser.getLastName());
                        user.setLoginName(sysUser.getLogin());
                        user.setUserId(new Long(newUserUid.longValue()));
                }
                toolContentObj.setCreatedBy(user);

        //                      refresh all resourceItem createBy user
                Set<ResourceItem> items = toolContentObj.getResourceItems();
                for(ResourceItem item:items){
                        item.setCreateBy(user);
                }
                resourceDao.saveObject(toolContentObj);
        } catch (ImportToolContentException e) {
                throw new ToolException(e);
        }
}
```

Why use helper methods?

Except serialize or deserialize content object, helper methods also handle upload/download stuff. The reasons are:

- This make program simple.
- Upload/Download file/package is common operation for export/import and it is possible to abstract them to helper methods.
- The attachment files may be contained in content object directly, maybe not. For instance, in share resources tool, it has offline/online instruction files in content object. It also has attached files in resource item. Helper methods can retrieve content object in and out and pick them up wherever they are.

Import tool content from different version

Tool could have different version. A subclass of ToolContentVersionFilter could help tool to import old version content package to lams server which running latest tools (or in reverse). The methods in subclass must follow below name convension.

- upXXXToYYY() : this method will help tool to import old version to new version. In this case, XXX < YYY.
- downXXXToYYY() : this method will help tool to import new version to old version. In this case, XXX > YYY.

The XXX and YYY is tool's version number. It should be integer format digit. Our recommend tool version format is YYYYMMDD. The new version number MUST greater than older version.

In the up/down method, it simply calls 2 methods from its parent class, ToolContentVersionFilter:

- public void removeField(Class ownerClass, String fieldname)
- public void addField(Class ownerClass, String fieldname, Object defaultValue)

Any removed fields must be declared by removeField() method. addField() is optional, except when the added fields need some special default value.

Further, register this Version Filter class to exportContentService. In ToolContentManager.importToolContent(), add following sentence:

```
exportContentService.registerImportVersionFilterClass(RsrcImportContentVersionFilter.class);
```

Check Your XML for Hibernate Objects

When you have written your logic, check the tool.xml produced by the export. It should not include any Hibernate class references. If you find reference to Hibernate object, you probably have either not set an unneeded value to null, or one of the subobjects contains a link back to an initial object. See the example code under Export tool content

In the example below, Scribe contains a set of ScribeHeadings. The ScribeHeading contains a reference to its Scribe, so if it is converted to XML as is, we end up with a Scribe, containing ScribeHeadings, which in turn contain the Hibernate version of Scribe. When the file is imported then the ScribeHeadings try to point to the wrong Scribe. This was fixed by first creating a new instance of Scribe and then going through the ScribeHeadings and setting their reference to Scribe to null.

Bad tool.xml

```
<createDate>2006-10-25 15:28:05.765 EST</createDate>
<title>LAMS Scribe</title>
<instructions>Scribe Instruction</instructions>
<runOffline>false</runOffline>
<lockOnFinished>true</lockOnFinished>
<reflectOnActivity>false</reflectOnActivity>
<onlineInstructions>Online instructions</onlineInstructions>
<offlineInstructions>Offline instructions</offlineInstructions>
<contentInUse>false</contentInUse>
<defineLater>false</defineLater>
<autoSelectScribe>true</autoSelectScribe>
<toolContentId>60</toolContentId>
<scribeAttachments/>
<scribeHeadings>
  <org.lamsfoundation.lams.tool.scribe.model.ScribeHeading>
    <scribe>
      <uid>1</uid>
      <title>LAMS Scribe</title>
      <instructions>Scribe Instruction</instructions>
      <runOffline>false</runOffline>
      <lockOnFinished>true</lockOnFinished>
      <reflectOnActivity>false</reflectOnActivity>
      <onlineInstructions>Online instructions</onlineInstructions>
      <offlineInstructions>Offline instructions</offlineInstructions>
      <contentInUse>false</contentInUse>
      <defineLater>false</defineLater>
      <autoSelectScribe>true</autoSelectScribe>
      <toolContentId>11</toolContentId>
      <scribeAttachments class="org.hibernate.collection.PersistentSet">
        <set/>
        <initialized>true</initialized>
        <collectionSnapshot class="org.hibernate.engine.CollectionEntry">
          <dirty>false</dirty>
          <initialized>true</initialized>
          <loadedKey class="long">1</loadedKey>
          <snapshot class="map"/>
          <role>org.lamsfoundation.lams.tool.scribe.model.Scribe.scribeAttachments</role>
        </collectionSnapshot>
        <owner class="org.lamsfoundation.lams.tool.scribe.model.Scribe" reference="../.."/>
      </scribeAttachments>
      <scribeSessions class="org.hibernate.collection.PersistentSet">
        <initialized>false</initialized>
        <collectionSnapshot class="org.hibernate.engine.CollectionEntry">
          <dirty>false</dirty>
          <initialized>false</initialized>
```

```
            <loadedKey class="long">1</loadedKey>
            <role>org.lamsfoundation.lams.tool.scribe.model.Scribe.scribeSessions</role>
          </collectionSnapshot>
          <owner class="org.lamsfoundation.lams.tool.scribe.model.Scribe" reference="../.."/>
        </scribeSessions>
        <scribeHeadings class="org.hibernate.collection.PersistentSet">
          <set>
            <org.lamsfoundation.lams.tool.scribe.model.ScribeHeading>
              <uid>1</uid>
              <scribe reference="../../../.."/>
              <headingText>Scribe Heading</headingText>
              <displayOrder>0</displayOrder>
            </org.lamsfoundation.lams.tool.scribe.model.ScribeHeading>
          </set>
          <initialized>true</initialized>
          <collectionSnapshot class="org.hibernate.engine.CollectionEntry">
            <dirty>false</dirty>
            <initialized>true</initialized>
            <loadedKey class="long">1</loadedKey>
            <snapshot class="map">
              <entry>
                <org.lamsfoundation.lams.tool.scribe.model.ScribeHeading \
reference="../../../../set/org.lamsfoundation.lams.tool.scribe.model.ScribeHeading"/>
                <org.lamsfoundation.lams.tool.scribe.model.ScribeHeading \
reference="../../../../set/org.lamsfoundation.lams.tool.scribe.model.ScribeHeading"/>
              </entry>
            </snapshot>
            <role>org.lamsfoundation.lams.tool.scribe.model.Scribe.scribeHeadings</role>
          </collectionSnapshot>
          <owner class="org.lamsfoundation.lams.tool.scribe.model.Scribe" reference="../.."/>
        </scribeHeadings>
      </scribe>
      <headingText>Scribe Heading</headingText>
      <displayOrder>0</displayOrder>
    </org.lamsfoundation.lams.tool.scribe.model.ScribeHeading>
  </scribeHeadings>
</org.lamsfoundation.lams.tool.scribe.model.Scribe>
```

Good tool.xml

```
<org.lamsfoundation.lams.tool.scribe.model.Scribe>
  <createDate>2006-10-26 12:29:52.359 EST</createDate>
  <updateDate class="sql-timestamp">2006-10-26 12:29:46.0</updateDate>
  <title>LAMS Scribe</title>
  <instructions>These are my instructions.</instructions>
  <runOffline>false</runOffline>
  <lockOnFinished>false</lockOnFinished>
  <reflectOnActivity>false</reflectOnActivity>
  <reflectInstructions></reflectInstructions>
  <onlineInstructions>Online instructions</onlineInstructions>
  <offlineInstructions>Offline instructions</offlineInstructions>
  <contentInUse>false</contentInUse>
  <defineLater>false</defineLater>
  <autoSelectScribe>true</autoSelectScribe>
  <toolContentId>58</toolContentId>
  <scribeAttachments>
    <org.lamsfoundation.lams.tool.scribe.model.ScribeAttachment>
      <fileVersionId>1</fileVersionId>
      <fileType>ONLINE</fileType>
      <fileName>all.zip</fileName>
      <fileUuid>1</fileUuid>
    </org.lamsfoundation.lams.tool.scribe.model.ScribeAttachment>
  </scribeAttachments>
  <scribeHeadings>
    <org.lamsfoundation.lams.tool.scribe.model.ScribeHeading>
      <headingText>This is a plain heading.</headingText>
      <displayOrder>1</displayOrder>
    </org.lamsfoundation.lams.tool.scribe.model.ScribeHeading>
    <org.lamsfoundation.lams.tool.scribe.model.ScribeHeading>
      <headingText>This is a &lt;strong&gt;formatted &lt;/strong&gt;&lt;em&gt;heading&lt;/em&gt;.</headingText>
      <displayOrder>0</displayOrder>
    </org.lamsfoundation.lams.tool.scribe.model.ScribeHeading>
  </scribeHeadings>
</org.lamsfoundation.lams.tool.scribe.model.Scribe>
```