Tool Adapters - Developers Page

Introduction

Tool adapter tools follow the same tool contract as all the other native tools which come packaged with LAMS, with a few slight variations. As with these native tools, a tool adapter tool must have author, learner and monitor interfaces, as well managing user and collaborative content through several tool sessions. The main difference when it comes to tool-adapter tools is that the content and business logic for the tool is all managed by the external LMS tool.

The tool adapter tool is (in simple terms) a gateway that points the user to the correct page and content on the external server based on their role and the content they ask for. For example when the teacher opens a tool adapter author page, they are merely redirected to the tool's authoring page on the external server and when the user is done, they click the finish button and the external LMS returns the user to LAMS having saved their content. Of-course it is not as simple as this statement may seem, but that covers the general idea of the tool adapter. In the later sections there are detailed explanations of how it all works and how to create your own tool adapter tools

Integration

Tool adapter tools need to have an external server mapping, because the external tool almost always needs to know user and course information before it will save any content, also there needs to be an authenticated session or the user will not see anything. So the sequence of events goes something like this:

1. A user called "Bill" starts from external LMS "Moodle" logged in as a teacher
2. User clicks on the LAMS "Open Author" link which is controlled by the integration.
3. A request is made to LAMS to open author for a user, LAMS looks for a user called "Bill" from the list of "Moodle" users, if it cannot find one, it creates one on the fly.
4. User is authenticated in LAMS as "moodle_Bill".
5. User drags a moodle forum tool in LAMS author onto the authoring frame and opens the authoring page for the tool.
6. User is redirected to the moodle forum tool author page which is located on the external "Moodle" server.
7. User's session credentials are maintained on external Server.

See this Use case in animations for Moodle and .LRN

Course/User Mapping - CustomCSV

One of the key differences betweens LAMS and other LMS systems is that learning designs and activities do no sit in the context of a course or a group. LAMS learning designs are completely independent, and are not applied to a group/course until they are used to start a lesson. In most other LMS systems this is not the case, to create an instance of an activity you must save it in a course, and also save it with a user who has permissions to do so for that course. The permissions for viewing/editing/monitoring are managed by the course that the activity is saved in.

So here in lies one of the main difficulties in developing a tool adapter tool, when the the user drags the tool adapter tool onto the canvas (see animation) and clicks on it to open author, how will the external tool know which course it is supposed to be made for on the external LMS, or which external user? The LAMS tool by default does not know this sort of information. Somehow this information has to get from the original author call from the external LMS to the LAMS tool adapter's authoring action.

The way this problem has been resolved in LAMS is to create a new class of tool called the tool_adapter tool that expects an extra parameter called **"customCSV"**. This parameter is a comma-separated-value string that is created by the external LMS tool and passed to LAMS when the user clicks author, or another LAMS action. Since the requirements to create a new tool activity instance may vary between each tool adapter tool, and the external LMS requirements may also vary, the customCSV can contain whatever parameters are needed in order to make the tool adapter work in each situation. For example in .LRN to create a new Forum instance, you need to have a user ID, a course ID, and you need to know the course URL to get to the authoring page, so the form of the customCSV is (userId, courseId, courseUrl). Moodle on the other hand does not require a course URL, but to create a new Forum instance there you need both the user and course ID as well as a section ID so moodle knows where to display the activity on the main course page. So the customCSV was made in the form (userId, courseId, sectionId).

CustomCSV is used in a lot of the extension points in the tool adapter so take note of it when reading the next sections.

Extension Points

There are a couple of key points where the LAMS server will have to talk to the external integrated server in order for the tool adapter to work. Many of the calls will be made to the **tool adapter servlet** which will handle the special functions required for LAMS tools that are not often directly implemented by other LMS tools. The tool adapter servlet should be in a freely accessible location on the external LMS server open to external calls,

For all the extension points to work for a tool adapter tool, it is **imperative that the tool's service implements the special org.lams.lamsfoundation.tool.ToolAdapterContentManager interface**. This interface not only identifies to Java that this is a tool adapater type tool, but also implements special methods to pass the customCSV where it is required.

Authoring

As was discussed earlier, in order for this tool adapter concept to work, we need to pass the authoring page the customCSV parameter so it can correctly create the activity instance on the external LMS. There are a few steps to follow before we can get to this point.

External LMS ID Mapping

One of the key concerns with a tool adapter tool is when to display it or allow it. We only really want to see the tool in the tool kit view in authoring when the authoring call originated from the external LMS, otherwise we wont have the required customCSV parameter and the tool will not perform correctly. So the way this has been implemented is to add an extra collumn to the lams_tool table called ext_lms_id. This property does two things: it identifies it as a tool_adapter type tool in flash, as well as mapping the tool directly to a integrated server id, so if the call does not appear from the integrated server (like for instance if the user goes directly to LAMS and opens author), the tool will not appear in the toolkit view. If the call does originate from the external LMS and the serverId in the authoring call matches the ext_lms_id stored in the tool's lams_tool table row, then the tool will be displayed. This allows for multiple integrations on one LAMS instance with tool_adapter tools mapping to separate integration instances.

To make this ext_lms_id configurable, the tool will need a admin page accessible through the sysadmin interfaces, you may need to add further configuration items to this page as your tool adapter requires including a external tool adapter servlet url which will be discussed later in this document.

Authoring Action

Once you have your ext_lms_id happening, you need to manage actually redirecting to the external tool's authoring page. When the user clicks on the activity to open author, a request is made to the tool's authoring action. If you have configured the ext_lms_id correctly, and you have passed the customCSV from the original author login request call, then the customCSV will be passed in the request to the authoring action. You can decode it here and then construct the authoring url based on the parameters. If the base authoring URL is static and unchanging (minus parameters) as with the moodle forum tool, then you may wish to store a server URL in the tool's admin page and hard-code the relative URL. If the author URL is dynamically changing as is the case with the .LRN Forum tool, you may need to include a author URL in the customCSV passed form the external server.

In your call to the external tool's authoring page, you will need to add a parameter that tells the external server the return or save-content url. That way when the user clicks save on the external tool, they can be returned to LAMS to save the tool. The external tool should add a extra parameter for the external content id, so it can be saved in the LAMS content instance and used for future references to this tool content.

Excerpt from Moodle Forum Tool's Authoring Action:

```
String customCSV = WebUtil.readStrParam(request, "customCSV", true);

String splitCSV[] = customCSV.split(",");
if (splitCSV.length != 3)
{
        logger.error("mdlForum tool customCSV not in required (user,course,courseURL) form: " + customCSV);
        throw new ToolException("mdlForum tool cusomCSV not in required (user,course,courseURL) form: " +
customCSV);
}
else
{
        userFromCSV = splitCSV[0];
        courseFromCSV = splitCSV[1];
        sectionFromCSV = splitCSV[2];
}

...

String responseUrl = mdlForumService.getConfigItem(MdlForumConfigItem.KEY_EXTERNAL_SERVER_URL).getConfigValue();
responseUrl += RELATIVE_MOODLE_AUTHOR_URL;

String returnUpdateUrl = URLEncoder.encode(
        TOOL_APP_URL + "/authoring.do?dispatch=updateContent" + "&"
        +AttributeNames.PARAM_TOOL_CONTENT_ID + "=" + toolContentID.toString(), "UTF8"
        );

responseUrl +=  "&lamsUpdateURL=" + returnUpdateUrl;

if (mdlForum.getExtSection() != null){
        responseUrl +=  "&section=" + mdlForum.getExtSection();
}
else{
        responseUrl +=  "&section=" + sectionFromCSV;
}

if (mdlForum.getExtToolContentId()!=null){
        responseUrl += "&update=" + mdlForum.getExtToolContentId().toString();
}
else{
        responseUrl += "&add=forum";
}

if (mdlForum.getExtCourseId() != null){
        responseUrl +="&course=" + mdlForum.getExtCourseId()
}
else{
        responseUrl +="&course=" + courseFromCSV;
}

log.debug("Sending to moodle forum edit page: " + responseUrl);
response.sendRedirect(responseUrl);
```

See complete AuthoringAction.java file.

Cloning Tool Content

For LAMS tools, there are several events that will trigger a request to clone the tool's content **including starting a lesson, starting preview, copy-pasting an activity in author, and inserting a sequence in author**. This is one of the most important extension points for a tool adapter tool as it takes care of a lot of the workflow and group implementation that LAMS is renowned for.

For native LAMS tools, the normal implementation when starting a lesson is to go through each tool in the sequence and copy the tool content. This copy is then used the lesson so the original does not get altered. Then all the run time (learner) data generated is referenced by a tool session id generated for each group in the lesson. This serves two purposes: firstly, the tool session id ensures that data stored through the lesson is kept separate to tool content stored when the teacher authored the sequence (thus lessons can be re-used); and secondly, collaborative work done by a group will in a lesson will only be displayed to people in the group thus providing a real group based workflow.

This raises another difficulty with external tool adapter tools in that external LMS tools rarely have such a tool-content/tool-session implementation, they do not ordinarily keep authored content and run-time content separate. So when it comes time to create a tool session for our tool adapter, there is not much we can tell the external tool adapter tool about the tool session. Instead the way this can be fixed is to make another clone copy of the tool content for each tool session, that way for each group in a LAMS lesson, there will be a special run-time copy of the tool-content and the group-based workflow will be maintained. The downside of this is that the separation between authored content and learner content is lost. Subsequently some functionalities like editing existing activities in live edit will not produce desired results, however the main functionality of the tool in the context of LAMS can be maintained.

The customCSV is required here also to ensure the cloned external tool content is cloned in the correct course. Take this scenario for instance, if a teacher was to be teaching in two seperate courses and they created tool content in one of those courses. If they were to start a lesson with the learning design saved in the first course, then in the second course they would find that the learners not enrolled in the first course would not be able to see the content. Therefore when we clone a tool adapter activity for a lesson, the course the lesson is started from must also be saved.

So in order to implement this cloning, a call must be made to the external tool adapter servlet set up for this tool adapter. Here we simply pass the external content id of the tool, and the course id for the lesson, then the servlet clones the tool and returns a new external tool content id.The customCSV param should be added to any calls from the external servlet to start a new lesson. Check out the code below for the moodle/LAMS forum tool that calls the external servlet for cloning.

```
 /* ************ Methods from ToolSessionManager ************* */
public void createToolSession(Long toolSessionId, String toolSessionName,
                Long toolContentId) throws ToolException {
        if (logger.isDebugEnabled()) {
                logger.debug("entering method createToolSession:"
                                + " toolSessionId = " + toolSessionId
                                + " toolSessionName = " + toolSessionName
                                + " toolContentId = " + toolContentId);
        }

        MdlForumSession session = new MdlForumSession();
        session.setSessionId(toolSessionId);
        session.setSessionName(toolSessionName);


        // learner starts
        MdlForum mdlForum = mdlForumDAO.getByContentId(toolContentId);
        session.setMdlForum(mdlForum);

        try
        {
                // Get the required params, then call the eternal server
                HashMap<String, String> params = getRequiredExtServletParams(mdlForum);
                params.put(EXT_SERVER_PARAM_EXT_TOOL_CONTENT_ID, mdlForum.getExtToolContentId().toString());
                session.setExtSessionId(copyExternalToolContent(params));
        }
        catch(Exception e)
        {
                logger.error("Failed to call external server to copy tool content" + e);
                throw new ToolException("Failed to call external server to copy tool content" + e);
        }

        mdlForumSessionDAO.saveOrUpdate(session);
}


/**
 * Calls the external server to copy the content and return a new id
 * @param extToolContentId
 * @return
 */
public Long copyExternalToolContent(HashMap<String, String> params)
throws ToolException, Exception
{

         String cloneServletUrl = mdlForumConfigItemDAO.getConfigItemByKey(MdlForumConfigItem.
KEY_EXTERNAL_TOOL_SERVLET).getConfigValue();

         // add the method to the params
         params.put(EXT_SERVER_PARAM_METHOD, EXT_SERVER_METHOD_CLONE);

        // Make the request
         InputStream is = WebUtility.getResponseInputStreamFromExternalServer(cloneServletUrl, params);
        BufferedReader isReader = new BufferedReader(new InputStreamReader(is));
        String str = isReader.readLine();
        if (str == null) {
            throw new UserInfoFetchException("Fail to clone forum in .LRN:"
                    + " - No data returned from external server");
        }

        return Long.parseLong(str);
}
```

See complete MdlForumService.java file

Learner/Monitor

As you may have suspected another extension point that must be implemented is the learner and monitor interfaces. All the information you need to redirect to these pages should have been stored in the tool adapter's tables by the time you get to this point. So it should simply be a matter of constructing the learner and monitor URL based on the save parameters and then redirecting the user when the request is made.

Excerpt from LearningAction.java

```
public ActionForward unspecified(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {

        //LearningForm learningForm = (LearningForm) form;

        // 'toolSessionID' and 'mode' paramters are expected to be present.
        // TODO need to catch exceptions and handle errors.
        //ToolAccessMode mode = WebUtil.readToolAccessModeParam(request,AttributeNames.PARAM_MODE,
MODE_OPTIONAL);

        Long toolSessionID = WebUtil.readLongParam(request, AttributeNames.PARAM_TOOL_SESSION_ID);

        // set up mdlForumService
        if (mdlForumService == null) {
                mdlForumService = MdlForumServiceProxy.getMdlForumService(this.getServlet().
getServletContext());
        }

        // Retrieve the session and content.
        MdlForumSession mdlForumSession = mdlForumService.getSessionBySessionId(toolSessionID);
        if (mdlForumSession == null) {
                throw new MdlForumException("Cannot retreive session with toolSessionID: "+ toolSessionID);
        }

        MdlForum mdlForum = mdlForumSession.getMdlForum();

        // check defineLater
        if (mdlForum.isDefineLater()) {
                return mapping.findForward("defineLater");
        }

        MdlForumDTO mdlForumDTO = new MdlForumDTO();
        request.setAttribute("mdlForumDTO", mdlForumDTO);

        // Set the content in use flag.
        if (!mdlForum.isContentInUse()) {
                mdlForum.setContentInUse(new Boolean(true));
                mdlForumService.saveOrUpdateMdlForum(mdlForum);
        }

        // check runOffline
        if (mdlForum.isRunOffline()) {
                return mapping.findForward("runOffline");
        }

        if (mdlForum.getExtToolContentId()!=null)
        {
                try{
                        String responseUrl = mdlForumService.getConfigItem(MdlForumConfigItem.
KEY_EXTERNAL_SERVER_URL).getConfigValue();
                        responseUrl += RELATIVE_LEARNER_URL;

                        String returnUrl = TOOL_APP_URL + "learning.do?"
                                + AttributeNames.PARAM_TOOL_SESSION_ID + "=" + toolSessionID.toString()
                                + "&dispatch=finishActivity";

                        returnUrl = URLEncoder.encode(returnUrl, "UTF8");

                        responseUrl += "&id=" + mdlForumSession.getExtSessionId()
                                + "&returnUrl=" + returnUrl;

                        log.debug("Redirecting for mdl forum learner: " + responseUrl);
                        response.sendRedirect(responseUrl);
```

```
                }
                catch (Exception e)
                {
                        log.error("Could not redirect to mdl forum authoring", e);
                }
        }
        else
        {
                throw new MdlForumException ("External content id null for learner");
        }
        return null;
}
```

See complete LearningAction.java file

Export Learning Design

To export a LAMS learning design, the authoring tool content needs to be serialised into a file so it can be imported again later in a different context. LAMS will go through a sequence and call each tool to output its tool content. This is made a bit more complicated for tool adapter tools, since they do not have the content stored locally, they only have an external content id. When the call is made to export the tool adapter content they have to call the external tool adapter servlet to request the content in a server to server call.

You will not need the customCSV for this so long as you saved the info in the customCSV in the LAMS tool content instance object, you will only need to retrieve the external tool adapter servlet url, and then add the parameters from the saved tool content.
Take note that you will need to handle the case where there is no external tool content (ie the authoring page for the external tool was never opened or saved).

Authentication (Required for all external server-to-server calls)

As with other integration server-to-server calls, the call to the tool adapter servlet will require a hash containing the timestamp, user, server id and secret server key. View the integrations page for more information on this.

Excerpt from MdlForumService

```
 /**
 * Export the XML fragment for the tool's content, along with any files
 * needed for the content.
 *
 * @throws DataMissingException
 *             if no tool content matches the toolSessionId
 * @throws ToolException
 *             if any other error occurs
 */

public void exportToolContent(Long toolContentId, String rootPath)
                throws DataMissingException, ToolException {


        MdlForum mdlForum = mdlForumDAO.getByContentId(toolContentId);
        if (mdlForum == null) {
                mdlForum = getDefaultContent();
        }
        if (mdlForum == null)
                throw new DataMissingException("Unable to find default content for the mdlForum tool");

        // If no external content was found, export empty mdlForum
        // Otherwise, call the external servlet to get the export file
        if (mdlForum.getExtToolContentId() == null)
        {
                mdlForum.setExtToolContentId(null);
                mdlForum.setToolContentHandler(null);
                mdlForum.setMdlForumSessions(null);

                try {
                        exportContentService.exportToolContent(toolContentId,
                                        mdlForum, mdlForumToolContentHandler, rootPath);
                } catch (ExportToolContentException e) {
                        throw new ToolException(e);
                }
        }
```

```
        else
        {

                URLConnection conn = null;
                try
                {
                        // Create the directory to store the export file
                        String toolPath = FileUtil.getFullPath(rootPath,toolContentId.toString());
                        FileUtil.createDirectory(toolPath);

                        String exportServletUrl = mdlForumConfigItemDAO.getConfigItemByKey(MdlForumConfigItem.
KEY_EXTERNAL_TOOL_SERVLET).getConfigValue();

                        // setting these to arbitrary values since they are only used to construct the hash

                        mdlForum.setExtCourseId("extCourse");
                        mdlForum.setExtSection("0");
                        mdlForum.setExtUsername("authUser");
                        HashMap<String, String> params = this.getRequiredExtServletParams(mdlForum);
                        params.put(EXT_SERVER_PARAM_METHOD, EXT_SERVER_METHOD_EXPORT);
                        params.put(EXT_SERVER_PARAM_EXT_TOOL_CONTENT_ID, mdlForum.getExtToolContentId().
toString());

                        // Get the reponse stream from the external server (hopefully containing the export file
                        InputStream in = WebUtility.getResponseInputStreamFromExternalServer(exportServletUrl,
params);

                        // Get the output stream to write the file for extport
                        OutputStream out = new BufferedOutputStream(new FileOutputStream(toolPath + "/ext_tool.
txt"));

                        byte[] buffer = new byte[1024];
                        int numRead;
                        long numWritten = 0;
                        while ((numRead = in.read(buffer)) != -1)
                        {
                                out.write(buffer, 0, numRead);
                                numWritten += numRead;
                        }
                        logger.debug("Path to mdlForum export content: " + toolPath + "/ext_tool.txt");

                        out.flush();
                        out.close();
                        in.close();
                }
                catch (Exception e)
                {
                        logger.error("Problem exporting data from external .LRN servlet", e);
                }
        }
}
```

See complete MdlForumService.java file

Import Learning Design

Once you have handled the export correctly, you must handle the import. For this, the exported files that are stored on the user's hard drive will have to be transported in a multi-part post to the external server. Again, here we will need to set up an action in the external tool adapter servlet. When a tool adapter tool is called to import tool content, it will be sent the path to a file, and the customCSV parameter. The customCSV parameter is included here because it is often the case that to create a new instance of an external tool, the course and user information is required. With this parameters you can construct the multipart request using the WebUtility.java file used for tool adapters.

Excerpt from MdlForumService.importToolContent()

```
File extExportFile = new File(toolContentPath + "/ext_tool.txt");

// if specially exported tooladapter file is found, send it to the external server
// otherwise, simply import an empty mdlForum
if (extExportFile.exists())
{

        try {

                String importServletUrl = mdlForumConfigItemDAO.getConfigItemByKey(MdlForumConfigItem.
KEY_EXTERNAL_TOOL_SERVLET).getConfigValue();

                if (customCSV== null)
                {
                        logger.error("Could not retrieve customCSV required for importing tool adapter tool.
CustomCSV: " + customCSV);
                        throw new ToolException("Could not retrieve customCSV required for importing tool
adapter tool. CustomCSV: " + customCSV);
                }

                HashMap<String, String> params = getRequiredExtServletParams(customCSV);
                params.put(EXT_SERVER_PARAM_METHOD, EXT_SERVER_METHOD_IMPORT);

                // Do the external multipart post to upload the file to external LMS (returns an
extToolContentId)
                InputStream is = WebUtility.performMultipartPost(extExportFile, EXT_SERVER_PARAM_UPLOAD_FILE,
importServletUrl, params);
                DataInputStream inStream = new DataInputStream ( is );
                String str = inStream.readLine();
                Long extContentId = Long.parseLong(str);
                inStream.close();

                // Save the resulting mdl forum
                MdlForum mdlForum = new MdlForum();
                mdlForum.setToolContentId(toolContentId);
                mdlForum.setCreateDate(new Date());
                mdlForum.setExtToolContentId(extContentId);
                saveOrUpdateMdlForum(mdlForum);

        } catch (Exception e) {
                logger.error("Problem passing mdlForum export file to external server", e);
                throw new ToolException(e);
        }
}
```

See complete MdlForumService.java file

Export Portfolio

Export Portfolio is another extension point that can be implemented although it is not essential as most LMS tool do not have a concept of export portfolio. Basically, another call should be made to the external servlet requesting the file, this request should pass the external tool content id and the user id.

Excerpt from ExportServlet.java

```
private void doTeacherExport(HttpServletRequest request,
                HttpServletResponse response, String directoryName, Cookie[] cookies)
                throws MdlForumException {

        logger.debug("doExportTeacher: toolContentID:" + toolContentID);

        // check if toolContentID available
        if (toolContentID == null) {
                String error = "Tool Content ID is missing. Unable to continue";
                logger.error(error);
                throw new MdlForumException(error);
        }

        MdlForum mdlForum = mdlForumService.getMdlForumByContentId(toolContentID);
        MdlForumDTO mdlForumDTO = new MdlForumDTO(mdlForum);
        request.getSession().setAttribute("mdlForumDTO", mdlForumDTO);

        Set <MdlForumSession> sessions = mdlForum.getMdlForumSessions();
        for (MdlForumSession session : sessions)
        {
                try {
                        String fullPath = directoryName + "/"+ session.getSessionName();
                        exportFileFromExternalServer(request, response, session.getExtSessionId(), mdlForum,
fullPath);
                } catch (Exception e) {
                        logger.error("Could not fetch export file from external servlet", e);
                        throw new MdlForumException("Could not fetch export file from external servlet", e);
                }
        }

        request.getSession().setAttribute("sessions", sessions);
}

private void exportFileFromExternalServer(HttpServletRequest request,
                HttpServletResponse response, Long extToolSessionId, MdlForum mdlForum, String fullPath) throws
Exception
{
        String exportPortFolioServletUrl = mdlForumService.getConfigItem(MdlForumConfigItem.
KEY_EXTERNAL_TOOL_SERVLET).getConfigValue();

        String extUsername="user"; // setting user to arbitrary values since they are only used to construct
the hash

        HashMap<String, String> params = mdlForumService.getRequiredExtServletParams(mdlForum);
        params.put("method", IMdlForumService.EXT_SERVER_METHOD_EXPORT_PORTFOLIO);
        params.put("extToolContentID", extToolSessionId.toString());

        InputStream in = WebUtility.getResponseInputStreamFromExternalServer(exportPortFolioServletUrl, params);
        OutputStream out = new BufferedOutputStream(new FileOutputStream(fullPath));
        byte[] buffer = new byte[1024];
        int numRead;
        long numWritten = 0;
        logger.debug("Getting file...");
        while ((numRead = in.read(buffer)) != -1)
        {
                out.write(buffer, 0, numRead);
                logger.debug(new String(buffer));
                numWritten += numRead;
        }
        logger.debug("Path to mdlForum export portfolio content: " + fullPath);
        out.flush();
        out.close();
}
```

See complete ExportServlet.java file

Tool Outputs

Tool Outputs are also an options extension point that you may like to implement. Here you can follow the same steps as are outline in the tool outputs wiki page. But instead of the getToolOutputs() call being a local call, you will need to again call the tool adapter servlet and pass the external tool content id and the user id, then return any data you wish about what the student has achieved in the activity.

Excerpt from MdlFroumOutputFactory.java

```
protected final static String OUTPUT_NAME_LEARNER_NUM_POSTS = "learner.number.of.posts";
protected final static String OUTPUT_NAME_LEARNER_NUM_WORDS = "learner.number.of.words";

public MdlForumOutputFactory() {}

/**
 * @see org.lamsfoundation.lams.tool.OutputDefinitionFactory#getToolOutputDefinitions(java.lang.Object)
 */
public SortedMap<String, ToolOutputDefinition> getToolOutputDefinitions(
                Object toolContentObject) throws ToolException {
        TreeMap<String, ToolOutputDefinition> definitionMap =  new TreeMap<String, ToolOutputDefinition>();

        ToolOutputDefinition definition = buildRangeDefinition(OUTPUT_NAME_LEARNER_NUM_POSTS, new Long(0),
null);
        definitionMap.put(OUTPUT_NAME_LEARNER_NUM_POSTS, definition);

        ToolOutputDefinition definition2 = buildRangeDefinition(OUTPUT_NAME_LEARNER_NUM_WORDS, new Long(0),
null);
        definitionMap.put(OUTPUT_NAME_LEARNER_NUM_WORDS, definition2);

        return definitionMap;
}

...

public ToolOutput getToolOutput(String name, IMdlForumService dlForumService,
                Long toolSessionId, Long learnerId, MdlForum getToolOutput, Long extSessionId) {
        if ( name != null)
        {
                return getExtToolOutput(name, dlForumService, getToolOutput, learnerId, extSessionId.toString(),
toolSessionId);
        }
        return null;

}

public ToolOutput getExtToolOutput(String outputName, IMdlForumService mdlForumService, MdlForum mdlForum, Long
userId, String extToolContentId, Long toolSessionId)
{
        int number = mdlForumService.getExternalToolOutputInt(outputName, mdlForum, userId, extToolContentId,
toolSessionId);
        return new ToolOutput(outputName, getI18NText(outputName, true), new Long(number));
}
```

See complete MdlForumOutputFactory.java file

Changed Java Files

Here is an itemised list of java files, and what has been changed in order to make the tool adapter work. It should be noted that the tool adapter tools were developed using the tool builder, with the Notebook tool being the template used.

When reading this section, all directories are relative to the lams_tool_mdlforum/src/java/org/lamsfoundation/lams/tool/mdfrum directory. See http://code.lamsfoundation.org/fisheye/browse/lams/tool_adapters/moodle/lams_tool_mdlforum/src/java/org/lamsfoundation/lams/tool/mdfrum

/dao

This is the "data access object" directory, the java files under this directory talk through hibernate to the database. These files are not implemented differently to other tools

/dto

This is the "data transfer object" directory, which manages the object that talk through struts to the jsp user interface. Again, these files are not implemented differenlty to other tools. But they are much simpler, since most of the user interface is handled by the external tool.

/model

This directory contains all the POJO simple java classes that will be used as persistence objects in the db. You may notice that it does not include the attachments or the notebook entry objects here as they do not apply for tool adapters.

- **MdlForum.java**: This contains all the base required/expected fields for a LAMS tool instance (eg uid, createDate, toolContentId etc). There are also 4 extra fields here, all of which correspond to values returned from the external LMS (in this case moodle). The first of these attributes (extToolContentID) is required by all tool adapter tools to identify the instances of the tool. The other three are there as a requirement for this particular implementation in moodle, and may be different depending on you tool's requirements. The other fields are passed in through the customCSV parameter and are basically all the information required to construct the author/learner URL, save, clone, or create new instances of the tool in the external server.
    - **extToolContentId**: This is the content id given by the external tool instance on the external LMS. It should be returned after the user first saves the tool in author. Unlike the other variables, this one is required for ALL tool adapter tools, as it is the identifier for the tool instance on the external server.
    - **extUserName**: This the external username of the user who creates the tool instance. This is one of the parameters passed through the customCSV at author time and is required by moodle to save, clone or create new instances of their forum tool.
    - **extCourseId**: This is the course id for the tool instance. Simularly to extUsername, this is passed through customCSV and is required by moodle to save, clone or create new instances.
    - **extSection**: This tells moodle where to display the tool instance on its user interface.
- **MdlConfigItem.java**: This is a simple POJO describing a key, value pair which will be used for the moodle config page. All tool adapters will require a configItem persistence object, you can basically put any configurations needed here that are static and may require configuration, for example, if your tool adapter requires a server url base path to construct the author url, you should put it here. There is also a integrated server key mapping that needs to be handled in the configuration pages, but we will discuss this later. The following config items were used for the forum tool.
    - **toolAdapterServlet**: This stores the URL location of the tool adapter servlet that handles all the external tool actions required for the tool adapter (as discussed earlier).
    - **extServerUrl**: This stores the base server URL, this is used to constuct the author and learner urls.
- **MdlForumSession.java**: This class refers to the tool sessions that get created when you start LAMS lessons. The only difference between this class and other tools is one parameter:
    - **extSessionId**: This varible is pretty much another word for the extToolContentId mentioned above in MdlForum.java, however this one was created by the cloning operation and acts as a run-time copy for learner and author. So pretty much, when you start the lesson, the tool adapter will call the clone method, which will clone the external tool instace and save the resulting id in the MdlForumSession. That way there is a new runtime copy for each LAMS tool session.
- **MdlForumUser.java**: This class has no discernable difference from other tools.

/service

The two files changed here were MdlForumService.java and MdlForumOutputFactory.java

- **MdlForumService.java**
    - Ordinarily, tool services must implement the core "ToolContentManager" interface instead, tool adapter tools implement the "ToolAdapterContentManager" interface, which in turn inherits from the "ToolContentManager" interface. This not only allows the core ToolManager service to identify the tool adapter tools, it also allows it to pass in the customCSV param to the methods that require it, without affecting the functionality of normal tools.
    - **createToolSession**() : Changed to accept the customCSV parameter. After decoding the custom CSV, this method will call the external tool's clone method to create a new instance of the tool for the LAMS tool session.
    - **copyExternalToolConten**t() : Added to the tool service. This method makes the call to the external tool adapter servlet to clone the tool instance.
    - **decodeCustomCSV**() : This method takes the customCSV argument and decodes it to produce a hashmap of parameters. This is implemented differently for each tool adapter (depending on what is required in the customCSV)
    - **getRequiredExtServletParams**() : This method is passed in a tool object and returns the required params (the same as decodeCustomCSV(). This method must only be called after the tool content has been saved.
    - **getExternalToolOutputInt**() : This method was added. It is an example on how to get the tool output. It makes a call to the external servlet requesting tool output for a tool instance.
    - **copyToolContent**() : This method has had an extra paramter added (customCSV) so the call can be made to the external tool adapter servlet to clone the tool instance.
    - **exportToolContent**() : This method has been altered so that if the external tool content is not null (ie the extTooLContentId is not null), it will call the external tool adapter servlet requesting a serialised version of the tool content. If there external tool content is null, it just outputs a simple serial representation of the tool in the same way as other tools do. So basically it is up to the external tool to determine the format it want to export it's tool to. Later on, if this needs to be imported, then it is up to the external tool to determine how to import the file.
    - **importToolContent**() : This method was altered to include customCSV parameter. The if the specially exported tool adapter file is found, it will send it to the external server, which should return a new extToolContentId to signify that a new instance of the tool has been imported. The external server must interpret the file and create an instance of the tool using the information in the file.
    - **getIntegrationService**() : Unlike other tools, tool adapter tools require an integration, so in turn the service needs to have access to integration information and hence requires the integration service. Because of the shortcomings of the Spring configuration, we were unable to inject this service, so it had to be fetched manually using this method.
- **MdlForumOutputFactory.java**
    - This class works in the same way as other examples in Forum and Multple choice, however the call back to the tool service will call the external tool adapter servlet to get the tool outputs. This is done through the getExternalToolOutputInt() method.

/util

There is very little difference between the the tool adapter's util dir and that of other tools apart from the WebUtility.java file.

- **WebUtility.java**: This file basically handles all the communication between the external tool adapter servlet and the LAMS tool adapter. It can have several methods depending on what you need to commnuicate. The two required for the moodle forum tool are:

- **getResponseInputStreamFromExternalServer**(String urlStr, HashMap<String, String> params): This makes a request to the url and appends the parameters in the hash. It returns a inputStream.
- **performMultipartPost** : This is required for the multi-part post (used for import/export) sequence.

/web/actions

These classes are used for struts as struts actions, they basically forward the user to the next location and take their imput using the ActionForms (see below).

- **AuthoringAction.java:** This action does two things, takes the user to the author page, and saves the content.
  - **unspecified() :** This is the default action. It will go here when the user clicks on the tool for author. Basically the change here was to redirect the user to the external tool's authoring page instead of returning an ActionForward. The customCSV is also passed to the action here so it can be used to construct the author url if neccessary. When the user is redirected, an extra param is attached to the URL, that is the LAMS return URL. The external tool can then redirect back to lams after they have saved their tool content.
  - **updateContent() :** This action is invoked when the user returns after saving the external tool content. It expects an extra parameter "extToolContentId" which is the id obtained by the external server when the user saved the tool content. The external tool content id is then saved before returning the user to the correct LAMS page.
- **LearingAction.java :** This action also only one thing differently to other tools.
  - **unspecified() :** This is the default action. Instead of returning an ActionForward like other tools, it will simply redirect the user to the external tool learning page. It gets the required information to construct the URL from the information saved earlier in author, and also from the information saved in the tool session.
- **MonitoringAction.java**
  - **unspecified() :** This is the default action. This behaves almost identically to the learning action. However, instead of redirecting to one URL, it constructs several URLs (one for each tool session) so that the teacher can monitor all groups. A page is then rendered using jsp.
- **AdminAction.java**
  - **unspecified() :** This is the default action. This populates the information in the config form
  - **saveContent**() : This saves the information in the form.

/web/forms

These classes are used for struts as the "ActionForm" POJOs. There is no diffeerence between these and notebook's apart from the extra class: AdminForm.

- **AdminForm.java**: This is a java representation of the tool's admin page form, it is used in the admin action in AdminAction.java. For the moodle forum tool adapter it had three parameters:
  - **toolAdapterServlet**: As explained earlier in MdlConfigItem
  - **extServerUrl**: As explained earlier in MdlConfigItem
  - **serverIdMapping**: This parameter is set here and will be sent to flash. Basically it tells flash to display the tool in author only when the request has come from the specified server with the server id mapping.

/web/servlets

- **ExportServlet.java**: This class handles the export portfolios for learner and monitor. We have left it up to the external tool to handle the form of this, as getting this information all in one go is not always possible. The main difference between this exportServlet and that in other tools is that instead of rendering its own html file from a jsp. This class will simply call the external tool adapter servlet and request the export portfolio file. Then it simple copied the file to the export location.

Code and examples

Currently there are two Tool Adapters, one for Moodle and one for .LRN. The Java code for these Tool Adapters can be browse in our CVS server. For changes to the LMS Tools to work with these Tool Adapters, see the Moodle Tool Adapter and .LRN Tool Adapters] pages respectively. Details for our CVS server are here.