**Migration Tool business logic**

**In TransfercaseController:**

String[] getUserNameAndPasswdByRequestCookie(HttpServletRequest request)

This function is used to get username and password from cookie of request, and return a 2-elements string array which contains username and password from the log in process.

**public** List<TestRail> getTestRailProjects(HttpServletRequest request, Model model)

This function defines the method to get all projects from TestRail system by reading DB.

**public** List<TestRailProject> getTestRailProjects2(HttpServletRequest request, Model model)

This function defines the method to get all projects from TestRail system by user interface.

**public** ReturnResult askForProcess(HttpServletRequest request, Model model)

This function allows browser ask for process of the current transforming. The return value contains percentage and process-string formatted as “transferred/total”

**public** Map<Integer, Integer> getGroupIdsByUserName(HttpServletRequest request, Model model)

This function achieves group information from SCTM DB by username. The result of keyset is group ids and the respective values are 1 when this group offers enough authorities, and 2 when that doesn’t.

**public** Integer getSCTMSupportTeamId(HttpServletRequest request, Model model)

This function achieves team id of “SCTM Support Team” in DB of SCTM system.

**public** List<SCTMProduct> getSCTMProduct(HttpServletRequest request, Model model)

This function achieves all product information from SCTM DB

**public** List<SCTMGroup> getSCTMGroup(HttpServletRequest request, Model model)

This function achieves all group information from SCTM DB

**public** List<TestRailSuite> getTestRailSuites(HttpServletRequest request, Model model)

This function achieves all suites information from TestRail DB

**public** ReturnResult run(HttpServletRequest request, Model model)

This function offers process importing data from TestRail to SCTM.

**if** (projectImportId < 0) {

***logger***.error("wrong selection, return -1");

**return** **new** ReturnResult(-1, **null**); // wrong selection

}

If user chooses wrong selection, -1 will be returned.

// judge assigned group

List<Integer> assigendGroupIds = **new** ArrayList<Integer>();

String[] stringGroupIds = groupInfo.split(", ");

**for**(String stringId : stringGroupIds)

assigendGroupIds.add(Integer.*parseInt*(stringId));

List<Integer> userGroupsList = **new** ArrayList<Integer>();

String[] stringUserGroups = userGroups.split(", ");

**for**(String stringId : stringUserGroups)

userGroupsList.add(Integer.*parseInt*(stringId));

userGroupsList.retainAll(assigendGroupIds);

**if**(userGroupsList.size() <= 0) **return** **new** ReturnResult(-4, **null**);

If user doesn’t have enough authorities (no green rows are selected), -4 will be returned.

TestRailProject project = tansferCaseService.getTestRailByProjectId(projectImportId);

TestPlanningHandler tph = **null**;

EntitiesHandler eh = **null**;

**try** {

String[] userNameAndPasswd = getUserNameAndPasswdByRequestCookie(request);

**if**(userNameAndPasswd == **null**) **throw** **new** Exception("cookie error");

ConnectionHandler ch = **new** ConnectionHandler();

LoginContext lc = ch.login(sctmUrl, userNameAndPasswd[0], userNameAndPasswd[1]);

tph = **new** TestPlanningHandler(lc);

eh = **new** EntitiesHandler(lc);

} **catch** (Exception ex) { // other exceptions

ex.printStackTrace();

***logger***.error("other exception, return -3, " + ex.getMessage());

**return** **new** ReturnResult(-3, ex.getMessage());

}

The handlers are created in SCTM and log in is initialed.

// add project to SCTM

**int** appTestManagerId = -1;

com.borland.testmanager.stubs.sccentities.webservice.Project p = **new** com.borland.testmanager.stubs.sccentities.webservice.Project(-1, outputName, project.getDescription() == **null** ? "" : project.getDescription(), **true**, appTestManagerId == -1 ? (**new** **int**[] {}) : (**new** **int**[] { appTestManagerId }));

**int** projectId = -1;

**try** {

**for** (SCTMProject sctmProject : tansferCaseService.getSCTMProjects()) {

**if** (sctmProject.getName().trim().equals(p.getName().trim()))

{

***logger***.error("repeated project name, return -2");

**return** **new** ReturnResult(-2, **null**);

}

}

projectId = eh.createProject(p);

} **catch** (Exception ex) {

ex.printStackTrace();

***logger***.error("repeated project name, return -2, " + ex.getMessage());

**return** **new** ReturnResult(-3, ex.getMessage()); // already exist

}

The project is created in SCTM

eh.assignProjectsById(projectId, assigendGroupIds);

tph.setCurrentProject(tph.getProjectById(projectId));

List<TestRailSuite> lstSuite = tansferCaseService.getTestRailSuitesByProjectId2(projectImportId);

List<Integer> lstSuiteIds = **new** ArrayList<Integer>();

**for** (TestRailSuite trsuite : lstSuite)

lstSuiteIds.add(trsuite.getId());

List<TestRailSection> lstSection = tansferCaseService.getTestRailSectionsBySuiteIds(lstSuiteIds);

List<TestRailCase> lstCase = tansferCaseService.getTestRailCasesBySuiteIds(lstSuiteIds);

counter.addNewSession(sessionId, lstSuite.size() + lstSection.size() + lstCase.size());

Map<Integer, Integer> suiteMap = **new** HashMap<Integer, Integer>();

Map<Integer, Integer> sectionMap = **new** HashMap<Integer, Integer>();

Map<Integer, TestRailSection> sectionFindMap = **new** HashMap<Integer, TestRailSection>();

TestRailSection tp = lstSection.get(0);

lstSection.set(0, lstSection.get(1));

lstSection.set(1, tp);

List<Integer> productsIds = **new** ArrayList<Integer>();

String[] productsIdsStr = productInfo.split(", ");

**for** (String productId : productsIdsStr) {

productsIds.add(Integer.*parseInt*(productId));

}

**if** (productsIds.size() == 1) {

**for** (**int** k = 0; k < lstSuite.size() - 1; k++)

productsIds.add(productsIds.get(0));

}

**int** cnt = 0;

**for** (TestRailSuite trSuite : lstSuite) {

**int** id = tph.addNode(-1, DemoUtil.*getDefaultContainer*(trSuite.getName(), trSuite.getDescription(),

projectId, productsIds.get(cnt++)), **false**);

suiteMap.put(trSuite.getId(), id);

counter.addOne(sessionId);

}

**for** (TestRailSection trSection : lstSection)

sectionFindMap.put(trSection.getId(), trSection);

**for** (TestRailSection trSection : lstSection) {

insertSection(sessionId, trSection, suiteMap, sectionMap, sectionFindMap, tph);

}

**for** (TestRailCase trCase : lstCase) {

**int** nodeid = tph.addNode(sectionMap.get(trCase.getSectionid()),

DemoUtil.*getDefaultTest*(trCase.getName(),trCase.getDescription(), projectId), **false**);

counter.addOne(sessionId);

**if** (!DemoUtil.*isNull*(trCase.getCustomSteps())

|| !DemoUtil.*isNull*(trCase.getCustomExpected()) || !DemoUtil.*isNull*(trCase.getCustomStepsSeparated())) {

DemoUtil.*appendStepToNode*(nodeid, trCase.getCustomSteps(), trCase.getCustomExpected(),

trCase.getCustomStepsSeparated(), tph);

}

}

counter.destroy(sessionId);

This process achieve transfer suites(containers), folders(sections), cases from TestRail to SCTM. Class DemoUtil offers a series of methods to generate new nodes in SCTM.

**void** insertSection(String sessionId, TestRailSection trSection, Map<Integer, Integer> suiteMap, Map<Integer, Integer> sectionMap, Map<Integer, TestRailSection> sectionFindMap, TestPlanningHandler tph)

This function helps transfer and insert sections by recursion because sequence in DB may not satisfy that father-nodes always appear before their children. If so, children-node will not find their respective father and data loss will be caused.

Annotations in this function is detailed.

**public** ReturnResult importCase(HttpServletRequest request, Model model) **throws** Exception

This function is an entrance of transferring cases from SCTM to TestRail. When project modes (single, single with baseline, multi-suite) are selected, different methods will be called.

**private** ReturnResult transferToMultipleProject(String sessionId, **int** testRailProjectId, List<TestDefinition> allTestDefinitions, List<TestRail> tempLists, APIClient client)

This function defines how to transfer cases from SCTM to TestRail with a multi-suite mode.

counter.addNewSession(sessionId, allTestDefinitions.size());

**for** (TestDefinition td : allTestDefinitions) {

**if** (td.getParentNodeId() == 0)

suite.add(td);

**else** **if** (td.getIsLeaf() == 1)

testCase.add(td);

**else**

section.add(td);

}

Collections.*sort*(suite, **new** Comparator<TestDefinition>() {

**public** **int** compare(TestDefinition o1, TestDefinition o2) {

**if** (o1.getName().equals("Master"))

**return** -1;

**if** (o2.getName().equals("Master"))

**return** 1;

**return** o1.getName().compareTo(o2.getName());

}

});

This part classifies different nodes in SCTM (containers, folders, cases) and sort is conducted for suites(containers).

**for** (TestDefinition td : suite) {

// transfer

Map<String, Object> data = **new** HashMap<String, Object>();

data.put("name", td.getName());

data.put("description", td.getDescription());

String urlString = "add\_suite/" + testRailProjectId;

JSONObject rootSection = (JSONObject) client.sendPost(urlString, data);

sendCnt++;

counter.addOne(sessionId);

Object newLongId = rootSection.get("id");

Integer newId = (newLongId **instanceof** Integer ? ((Integer) newLongId) : ((Long) newLongId).intValue());

idMap.put(td.getNodeId(), newId);

idSuiteIdMap.put(td.getNodeId(), newId);

}

This part helps transfer containers.

List<TestDefinition> bufferSpaceForSections = **new** ArrayList<TestDefinition>();

**for** (TestDefinition td : section) {

**int** faterId = td.getParentNodeId();

// father has been created

**if** (idMap.containsKey(faterId)) {

// transfer

Map<String, Object> data = **new** HashMap<String, Object>();

data.put("name", td.getName());

data.put("description", td.getDescription());

data.put("suite\_id", idSuiteIdMap.get(faterId));

**if** (idMap.get(faterId).equals(idSuiteIdMap.get(faterId))) {

// father is a suite

// data.put("parent\_id", idMap.get(faterId));

} **else** {

// father is a section

data.put("parent\_id", idMap.get(faterId));

}

String urlString = "add\_section/" + testRailProjectId;

JSONObject rootSection = (JSONObject) client.sendPost(urlString, data);

sendCnt++;

counter.addOne(sessionId);

Object newLongId = rootSection.get("id");

Integer newId = (newLongId **instanceof** Integer ? ((Integer) newLongId) : ((Long) newLongId).intValue());

idMap.put(td.getNodeId(), newId);

idSuiteIdMap.put(td.getNodeId(), idSuiteIdMap.get(faterId));

} **else** {

// father hasn't been created

bufferSpaceForSections.add(td);

}

}

***logger***.info("bufferSpaceForSections size: " + bufferSpaceForSections.size());

**boolean** first = **true**;

**int** lastSize = -1;

**while** (first || lastSize != bufferSpaceForSections.size()) {

first = **false**;

lastSize = bufferSpaceForSections.size();

**for** (**int** i = 0; i < bufferSpaceForSections.size();) {

TestDefinition td = bufferSpaceForSections.get(i);

**int** faterId = td.getParentNodeId();

// father has been created

**if** (idMap.containsKey(faterId)) {

// transfer

Map<String, Object> data = **new** HashMap<String, Object>();

data.put("name", td.getName());

data.put("description", td.getDescription());

data.put("suite\_id", idSuiteIdMap.get(faterId));

**if** (idMap.get(faterId).equals(idSuiteIdMap.get(faterId))) {

// father is a suite

// data.put("parent\_id", idMap.get(faterId));

} **else** {

// father is a section

data.put("parent\_id", idMap.get(faterId));

}

String urlString = "add\_section/" + testRailProjectId;

JSONObject rootSection = (JSONObject) client.sendPost(urlString, data);

sendCnt++;

counter.addOne(sessionId);

Object newLongId = rootSection.get("id");

Integer newId = (newLongId **instanceof** Integer ? ((Integer) newLongId): ((Long) newLongId).intValue());

idMap.put(td.getNodeId(), newId);

idSuiteIdMap.put(td.getNodeId(), idSuiteIdMap.get(faterId));

bufferSpaceForSections.remove(i);

} **else**

i++;

}

}

***logger***.info("bufferSpaceForSections size: " + bufferSpaceForSections.size());

invalid = bufferSpaceForSections.size();

This part transfers sections from TestRail to SCTM. For sections are tree constructions, a new loop is need to check if all father-nodes are inserted before their child-nodes. Here I use a queue to store all nodes whose father are not found temporarily and it is scanned over and over again until length keeps fixed. The nodes still in queue are dirty data (with a tiny probability), which should be dropped.

// create cases

**for** (TestDefinition td : testCase) {

**int** faterId = td.getParentNodeId();

// father has been created

**if** (idMap.containsKey(faterId)) {

// transfer

Map<String, Object> data = **new** HashMap<String, Object>();

data.put("title", td.getName());

data.put("custom\_description", td.getDescription());

**if**(idMap.get(faterId).equals(idSuiteIdMap.get(faterId))) { // father is a container

**if**(defaultSectionMap.containsKey(idMap.get(faterId))) { // default has been created

**int** defaultSectionId = defaultSectionMap.get(idMap.get(faterId));

client.sendPost("add\_case/" + defaultSectionId, data);

sendCnt++;

counter.addOne(sessionId);

} **else** {

// create a default section

Map<String, Object> data2 = **new** HashMap<String, Object>();

data2.put("name", "default section");

data2.put("description", "this section includes cases whose father nodes are containers");

data2.put("suite\_id", idMap.get(faterId));

String urlString = "add\_section/" + testRailProjectId;

JSONObject rootSection = (JSONObject) client.sendPost(urlString, data2);

sendCnt++;

counter.addOne(sessionId);

Object newLongId = rootSection.get("id");

Integer newId = (newLongId **instanceof** Integer ? ((Integer) newLongId): ((Long) newLongId).intValue());

defaultSectionMap.put(idMap.get(faterId), newId);

// insert this case into default section

client.sendPost("add\_case/" + newId, data);

sendCnt++;

counter.addOne(sessionId);

}

}

**else** {

client.sendPost("add\_case/" + idMap.get(faterId), data);

sendCnt++;

counter.addOne(sessionId);

}

} **else**

invalid++;

When sections are inserted, each case can be linked directly to its father-node. If some case’s father is not found, this case should be dropped as dirty data.

**private** ReturnResult transferToSingle2Project(String sessionId, **int** testRailProjectId, List<TestDefinition> allTestDefinitions, List<TestRail> tempLists, APIClient client)

This function defines how to transfer cases from SCTM to TestRail with a single-baseline mode. Detailed process is similar to the above.

**private** ReturnResult transferToSingle1Project(String sessionId, **int** testRailProjectId, List<TestDefinition> allTestDefinitions, List<TestRail> tempLists, APIClient client)

This function defines how to transfer cases from SCTM to TestRail with a single mode. Detailed process is similar to the above.

**In Login Controller:**

@RequestMapping(value = "/", method = RequestMethod.***GET***)

**public** String transferCase(HttpServletRequest request, Model model)

This function offer the entrance of this migration tool system. When user hasn’t logged in before (no message in cookie), it guides to logging in page. On the contrary, if usernames and passwords are in cookie, it will link to branch page.

@RequestMapping(value = "/function", method = RequestMethod.***GET***)

**public** String transferCase2(HttpServletRequest request, HttpServletResponse response)

This method will lead you to the function page (migration or inactive subpage) by the value of “target” in cookie.

@RequestMapping(value = "/login", method = RequestMethod.***POST***)

@ResponseBody

**public** **boolean** login(HttpServletRequest request, Model model)

This function returns a boolean value to judge if username and password (in cookie) is legal in TestRail. A tiny TestRail logging-in-test is conducted.