Integration with ScriptRunner

Some customers and some partners may be using ScriptRunner in order to automate some tasks and even extend the features provided by Jira.

You can also do some automation-related with Xray, especially because we use Jira entities and concepts.

ScriptRunner may be used in order to access existing features or even to extend the built-in features.

Let us know if you're using also ScriptRunner and your use cases so we can improve and share them with other users.



Please note

The following scripts are provided as-is, no warranties attached. Use these scripts carefully.

Please feel free to adapt them to your needs.

Note: We don't provide support for ScriptRunner; if you have doubts concerning its usage, please contact ScriptRunner's support.

- Create a Test Set or a Test Plan
- Create a Test Execution
- Validate requirement before alowing to make a transition
- Reopen/transition linked Tests to a requirement
- Requirement projects
- Create Test Execution from Test Set issue screen
- Calculate requirement status for a certain version
- Show Tests Count for a requirement
- Show Defects Count for a requirement
- Configured Issue Types as being requirements or defects
- Trigger a Jenkins project build from a Test Plan
  - Example
    - ScriptRunner configuration
    - Jenkins configuration
- Trigger a Jenkins project build from a Test Plan, for the Tests contained in the Test Plan
  - Example
    - ScriptRunner configuration
    - Jenkins configuration
- Trigger a Bamboo plan build from a Test Plan
  - Example
    - ScripRunner configuration
    - Bamboo configuration
- Trigger a Bamboo plan/stage build from a Test Plan, for the Tests contained in the Test Plan
  - Example
    - ScripRunner configuration
    - Bamboo configuration
- Extending REST API for interacting with requirement projects
  - Example of requests
    - Obtaining all projects with requirement coverage enabled
    - Enabling requirement coverage for a project
    - Disabling requirement coverage for a project
- Synchronize Tests from "related" Test Sets to a Test Execution or to a Test Plan
  - ScripRunner configuration
    - Example
- Add a custom link to the Tests top menu
  - ScripRunner configuration
    - Example
    -

Create a Test Set or a Test Plan

Sometimes you may need to create a Test Set or a Test Plan programmatically.

The following example shows how to create a Test Plan or a Test Set based on setting the value for specific Xray custom fields.

create_xray_entities.groovy

```
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.link.IssueLinkManager
import com.atlassian.jira.issue.link.IssueLinkType
```

```
import com.atlassian.jira.issue.link.IssueLinkTypeManager
import com.atlassian.jira.ComponentManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.jql.builder.JqlQueryBuilder
import com.atlassian.jira.user.util.UserUtil
import com.atlassian.jira.user.util.UserManager;
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.search.SearchService
import com.atlassian.jira.issue.search.SearchProvider
import com.atlassian.jira.issue.search.SearchResults
import com.atlassian.jira.web.bean.PagerFilter;
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.user.UserPropertyManager
import com.atlassian.jira.propertyset.JiraPropertySetFactory;
import com.google.common.collect.ImmutableMap;
import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.module.propertyset.PropertySetManager;
import com.atlassian.jira.util.BuildUtils
import com.atlassian.jira.util.BuildUtilsInfo
import com.atlassian.jira.util.BuildUtilsInfoImpl
import com.atlassian.plugin.PluginAccessor
import com.atlassian.plugin.PluginManager
import com.atlassian.jira.bc.license.JiraLicenseService
import com.atlassian.jira.bc.license.JiraLicenseServiceImpl
import org.apache.log4j.Level
import org.apache.log4j.Logger
import com.atlassian.jira.issue.IssueManager
import org.ofbiz.core.entity.GenericValue
import com.atlassian.jira.ComponentManager
import com.atlassian.jira.event.type.EventDispatchOption
import groovy.json.JsonOutput
import groovy.transform.BaseScript
import groovy.json.JsonSlurper;
import groovy.json.StreamingJsonBuilder;
import javax.ws.rs.core.MultivaluedMap
import javax.ws.rs.core.Response
import com.atlassian.jira.issue.index.IssueIndexingService
import com.atlassian.jira.util.ImportUtils
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.IssueService.CreateValidationResult
import com.atlassian.jira.bc.issue.IssueService.IssueResult
import com.atlassian.jira.user.ApplicationUser



Logger.getLogger("com.onresolve").setLevel(Level.DEBUG)

// creates a Sub Test Execution from a requirement issue, with all linked Tests


projectManager = ComponentAccessor.getProjectManager()
componentManager = ComponentManager.getInstance()
issueManager = ComponentAccessor.getIssueManager()
def issueFactory = ComponentAccessor.getIssueFactory()
issueService = ComponentAccessor.issueService
searchService =  ComponentAccessor.getComponent(SearchService.class);
serviceAccount = ComponentAccessor.getJiraAuthenticationContext().getLoggedInUser()
customFieldManager = ComponentAccessor.getCustomFieldManager()
def subTaskManager = ComponentAccessor.getSubTaskManager()
issueService = ComponentAccessor.getIssueService()
def user = ComponentAccessor.jiraAuthenticationContext.getLoggedInUser()



Object getIssues(jqlQuery){
    // A list of GenericValues representing issues
    List<Issue> searchResults = null;

    SearchService.ParseResult parseResult =  searchService.parseQuery(serviceAccount, jqlQuery);

    if (parseResult.isValid()) {
```

```
        // throws SearchException
        SearchResults results = searchService.search(serviceAccount, parseResult.getQuery(), PagerFilter.
getUnlimitedFilter());
        searchResults = results.getIssues();
        return searchResults;
    }

     return []
}

Object getFieldValueByName(issue,customField) {
    def cField = customFieldManager.getCustomFieldObjectByName(customField)
    def cFieldValue = issue.getCustomFieldValue(cField)
    return cFieldValue
}


Object setFieldValueByName(issue,customField,value) {
    def cField = customFieldManager.getCustomFieldObjectByName(customField)
    issue.setCustomFieldValue(cField,*value)
}


Object setFieldValueByNameInParameters(inputParameters,customFieldName,value) {
    def customField = customFieldManager.getCustomFieldObjectByName(customFieldName)
    inputParameters.addCustomFieldValue(customField.id, value)
}



def project = projectManager.getProjectObjByKey("CALC")
def newIssueType = ComponentAccessor.issueTypeSchemeManager.getIssueTypesForProject(project).find { it.name ==
"Test Plan" }

def newIssue


def issueInputParameters = issueService.newIssueInputParameters()
issueInputParameters.with {
    projectId = project.id
    summary = "Issue created from script"
    issueTypeId = newIssueType.id
    reporterId = user.name
}


def jql = "project = ${project.key} and issuetype  = Test and component = UI"
def issues = getIssues(jql)
def arr = issues.collect{ it.key }
log.debug("testKeys: "+arr)
testKeys=arr.toArray(new String[arr.size()])

// Tests association with a Test Execution: setting it through the CF is currently not possible due to bug XRAY-
2010
//setFieldValueByNameInParameters(issueInputParameters,"Tests association with a Test Set",testKeys)
setFieldValueByNameInParameters(issueInputParameters,"Tests associated with a Test Plan",testKeys)


appUser = user.getDirectoryUser()
CreateValidationResult createValidationResult = issueService.validateCreate(user, issueInputParameters)
if (!createValidationResult.isValid()) {
    log.error "Error validating new issue"+createValidationResult.getErrorCollection()
} else {
    IssueResult createResult = issueService.create(user, createValidationResult)
    newIssue = createResult.issue
    log.debug(newIssue.key)

}
```

Create a Test Execution

Sometimes you may need to create a Test Execution programmatically.

Currently there is a limitation to associate the Tests by custom field. Thus, a possible workaround using Xray's REST API is shown in the following example.

create_test_execution.groovy

```
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.link.IssueLinkManager
import com.atlassian.jira.issue.link.IssueLinkType
import com.atlassian.jira.issue.link.IssueLinkTypeManager
import com.atlassian.jira.ComponentManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.jql.builder.JqlQueryBuilder
import com.atlassian.jira.user.util.UserUtil
import com.atlassian.jira.user.util.UserManager;
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.search.SearchService
import com.atlassian.jira.issue.search.SearchProvider
import com.atlassian.jira.issue.search.SearchResults
import com.atlassian.jira.web.bean.PagerFilter;
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.user.UserPropertyManager
import com.atlassian.jira.propertyset.JiraPropertySetFactory;
import com.google.common.collect.ImmutableMap;
import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.module.propertyset.PropertySetManager;
import com.atlassian.jira.util.BuildUtils
import com.atlassian.jira.util.BuildUtilsInfo
import com.atlassian.jira.util.BuildUtilsInfoImpl
import com.atlassian.plugin.PluginAccessor
import com.atlassian.plugin.PluginManager
import com.atlassian.jira.bc.license.JiraLicenseService
import com.atlassian.jira.bc.license.JiraLicenseServiceImpl
import org.apache.log4j.Level
import org.apache.log4j.Logger
import com.atlassian.jira.issue.IssueManager
import org.ofbiz.core.entity.GenericValue
import com.atlassian.jira.ComponentManager
import com.atlassian.jira.event.type.EventDispatchOption
import groovy.json.JsonOutput
import groovy.transform.BaseScript
import groovy.json.JsonSlurper;
import groovy.json.StreamingJsonBuilder;
import javax.ws.rs.core.MultivaluedMap
import javax.ws.rs.core.Response
import com.atlassian.jira.issue.index.IssueIndexingService
import com.atlassian.jira.util.ImportUtils
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.IssueService.CreateValidationResult
import com.atlassian.jira.bc.issue.IssueService.IssueResult
import com.atlassian.jira.user.ApplicationUser



Logger.getLogger("com.onresolve").setLevel(Level.DEBUG)


projectManager = ComponentAccessor.getProjectManager()
componentManager = ComponentManager.getInstance()
issueManager = ComponentAccessor.getIssueManager()
def issueFactory = ComponentAccessor.getIssueFactory()
issueService = ComponentAccessor.issueService
searchService =  ComponentAccessor.getComponent(SearchService.class);
serviceAccount = ComponentAccessor.getJiraAuthenticationContext().getLoggedInUser()
customFieldManager = ComponentAccessor.getCustomFieldManager()
```

```
def subTaskManager = ComponentAccessor.getSubTaskManager()
issueService = ComponentAccessor.getIssueService()
def user = ComponentAccessor.jiraAuthenticationContext.getLoggedInUser()



Object getIssues(jqlQuery){
    // A list of GenericValues representing issues
    List<Issue> searchResults = null;

    SearchService.ParseResult parseResult =  searchService.parseQuery(serviceAccount, jqlQuery);

    if (parseResult.isValid()) {
        // throws SearchException
        SearchResults results = searchService.search(serviceAccount, parseResult.getQuery(), PagerFilter.
getUnlimitedFilter());
        searchResults = results.getIssues();
        return searchResults;
    }

     return []
}

Object getFieldValueByName(issue,customField) {
    def cField = customFieldManager.getCustomFieldObjectByName(customField)
    def cFieldValue = issue.getCustomFieldValue(cField)
    return cFieldValue
}


Object setFieldValueByName(issue,customField,value) {
    def cField = customFieldManager.getCustomFieldObjectByName(customField)
    issue.setCustomFieldValue(cField,*value)
}


Object setFieldValueByNameInParameters(inputParameters,customFieldName,value) {
    def customField = customFieldManager.getCustomFieldObjectByName(customFieldName)
    inputParameters.addCustomFieldValue(customField.id, value)
}

boolean associateTestsToTestExecution(testExecKey,listOfTestKeys){
    def jiraBaseUrl = com.atlassian.jira.component.ComponentAccessor.getApplicationProperties().getString("jira.
baseurl")
    def endpointUrl = "${jiraBaseUrl}/rest/raven/1.0/api/testexec/${testExecKey}/test"
    url = new URL(endpointUrl);
    def body_req = [ "add": listOfTestKeys ]

    // you should use a specific user for this purpose
    username = "admin"
    password = "admin"
    def authString = "${username}:${password}".bytes.encodeBase64().toString()

    URLConnection connection = url.openConnection();
    connection.requestMethod = "POST"
    connection.doOutput = true
    connection.addRequestProperty("Authorization", "Basic ${authString}")
    connection.setRequestProperty("Content-Type", "application/json;charset=UTF-8")
    connection.outputStream.withWriter("UTF-8") { new StreamingJsonBuilder(it, body_req) }
    connection.connect();
    log.debug(connection.getResponseCode())
    log.debug(connection.getResponseMessage())

    if (connection.getResponseCode() == 200) {
        // OK
        return true;
    } else {
        // error
        return false;
    }
}
```

```
def project = projectManager.getProjectObjByKey("CALC")
def newIssueType = ComponentAccessor.issueTypeSchemeManager.getIssueTypesForProject(project).find { it.name ==
"Test Execution" }

def newIssue


def issueInputParameters = issueService.newIssueInputParameters()
issueInputParameters.with {
    projectId = project.id
    summary = "Issue created from script"
    issueTypeId = newIssueType.id
    reporterId = user.name
}


def jql = "project = ${project.key} and issuetype  = Test and component = UI"
def issues = getIssues(jql)
def testKeys = issues.collect{ it.key }


appUser = user.getDirectoryUser()
CreateValidationResult createValidationResult = issueService.validateCreate(user, issueInputParameters)
if (!createValidationResult.isValid()) {
    log.error "Error validating new issue"+createValidationResult.getErrorCollection()
} else {
    IssueResult createResult = issueService.create(user, createValidationResult)
    newIssue = createResult.issue
    log.debug(newIssue.key)
    associateTestsToTestExecution(newIssue.key, testKeys)
}
```

Validate requirement before alowing to make a transition

Sometimes you may need to assure that the requirement is actually OK before transitioning it to some status, or before resolving it.

The following script validates the requirement based on the tests executed for the version assigned to the requirement issue.

You can either make the validation based on the existence of failed tests (i.e. requirement status is "NOK") or, in a more complete way by making sure all of them are passing (i.e. requirement status is "OK").

requirement_validation.groovy

```
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.link.IssueLinkManager
import com.atlassian.jira.issue.link.IssueLinkType
import com.atlassian.jira.issue.link.IssueLinkTypeManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.jql.builder.JqlQueryBuilder
import com.atlassian.jira.user.util.UserManager;
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.search.SearchService;
import com.atlassian.jira.issue.search.SearchProvider
import com.atlassian.jira.issue.search.SearchResults
import com.atlassian.jira.web.bean.PagerFilter;
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.user.UserPropertyManager
import com.atlassian.jira.propertyset.JiraPropertySetFactory;
import com.google.common.collect.ImmutableMap;
import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.module.propertyset.PropertySetManager;
import com.atlassian.jira.util.BuildUtils
import com.atlassian.jira.util.BuildUtilsInfo
import com.atlassian.jira.util.BuildUtilsInfoImpl
import com.atlassian.plugin.PluginManager
import com.atlassian.jira.bc.license.JiraLicenseService
import com.atlassian.jira.bc.license.JiraLicenseServiceImpl
```

```
import org.apache.log4j.Level
import org.apache.log4j.Logger
import com.atlassian.jira.issue.IssueManager
import com.opensymphony.workflow.InvalidInputException


Object getIssues(jqlQuery){
    // A list of GenericValues representing issues
    List<Issue> searchResults = null;

    SearchService.ParseResult parseResult =  searchService.parseQuery(serviceAccount, jqlQuery);

    if (parseResult.isValid()) {
        // throws SearchException
        SearchResults results = searchService.search(serviceAccount, parseResult.getQuery(), PagerFilter.
getUnlimitedFilter());
        searchResults = results.getIssues();
        return searchResults;
    }

     return []
}


searchService =  ComponentAccessor.getComponent(SearchService.class);
issueManager = ComponentAccessor.getIssueManager()
serviceAccount = ComponentAccessor.getJiraAuthenticationContext().getLoggedInUser()

Issue issue = issue;

log.debug(issue.project.name)
projectName = issue.project.name
version = issue.fixVersions.join('')
log.debug(version)

//jql = "key = ${issue.key} and issue in requirements('NOK','${projectName}','${version}')"
jql = "key = ${issue.key} and issue in requirements('OK','${projectName}','${version}')"
issues = getIssues(jql)
count = issues.size
//log.debug(count)

/*
if (count>0) {
    invalidInputException = new InvalidInputException("Some tests are failing. You must assure that they pass
before making the transition.")
}
*/

if (count == 0) {
    invalidInputException = new InvalidInputException("Some tests need to be executed. You must assure that
they pass before making the transition.")
}
```

Reopen/transition linked Tests to a requirement

Whenever you change the specification of a requirement, you most probably will need to review the Tests that you have already specified.

The following script tries to make a transition on all linked Tests to a requirement. You can hook it to a post-function on the transition of the requirement.

reopen_linked_tests.groovy

```
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.link.IssueLinkManager
import com.atlassian.jira.issue.link.IssueLinkType
import com.atlassian.jira.issue.link.IssueLinkTypeManager
```

```
import com.atlassian.jira.ComponentManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.jql.builder.JqlQueryBuilder
import com.atlassian.jira.user.util.UserManager;
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.search.SearchService;
import com.atlassian.jira.issue.search.SearchProvider
import com.atlassian.jira.issue.search.SearchResults
import com.atlassian.jira.web.bean.PagerFilter;
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.user.UserPropertyManager
import com.atlassian.jira.propertyset.JiraPropertySetFactory;
import com.google.common.collect.ImmutableMap;
import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.module.propertyset.PropertySetManager;
import com.atlassian.jira.util.BuildUtils
import com.atlassian.jira.util.BuildUtilsInfo
import com.atlassian.jira.util.BuildUtilsInfoImpl
import com.atlassian.plugin.PluginManager
import com.atlassian.jira.bc.license.JiraLicenseService
import com.atlassian.jira.bc.license.JiraLicenseServiceImpl
import org.apache.log4j.Level
import org.apache.log4j.Logger
import com.atlassian.jira.issue.IssueManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.config.ResolutionManager
import com.atlassian.jira.workflow.WorkflowTransitionUtil
import com.atlassian.jira.workflow.WorkflowTransitionUtilImpl
import com.atlassian.jira.util.JiraUtils
import com.atlassian.jira.bc.ServiceResultImpl
import com.atlassian.jira.bc.issue.IssueService.TransitionValidationResult
import com.atlassian.jira.issue.IssueInputParametersImpl
import com.atlassian.jira.bc.issue.DefaultIssueService
import com.opensymphony.workflow.InvalidActionException
import com.atlassian.jira.workflow.IssueWorkflowManager



Object getIssues(jqlQuery){
    // A list of GenericValues representing issues
    List<Issue> searchResults = null;

    SearchService.ParseResult parseResult =  searchService.parseQuery(serviceAccount, jqlQuery);

    if (parseResult.isValid()) {
        // throws SearchException
        SearchResults results = searchService.search(serviceAccount, parseResult.getQuery(), PagerFilter.
getUnlimitedFilter());
        searchResults = results.getIssues();
        return searchResults;
    }

     return []
}


searchService =  ComponentAccessor.getComponent(SearchService.class);
issueManager = ComponentAccessor.getIssueManager()
customFieldManager = ComponentAccessor.getCustomFieldManager()
serviceAccount = ComponentAccessor.getJiraAuthenticationContext().getLoggedInUser()
IssueWorkflowManager issueWorkflowManager = ComponentAccessor.getComponentOfType(IssueWorkflowManager.class);

Issue issue = issue;

jql = "issue in requirementTests('${issue.key}')"
issues = getIssues(jql)


issues.each {
    WorkflowTransitionUtil workflowTransitionUtil = ( WorkflowTransitionUtil ) JiraUtils.loadComponent(
WorkflowTransitionUtilImpl.class );
```

```
    MutableIssue tempissue = issueManager.getIssueObject(it.key)
    workflowTransitionUtil.setIssue(tempissue);
    workflowTransitionUtil.setUsername(serviceAccount.getUsername());

    def actionId = 3  // change it accordingly
    if (issueWorkflowManager.isValidAction(tempissue, actionId)){
        workflowTransitionUtil.setAction(actionId);//Id of the status you want to transition to
        try {
            workflowTransitionUtil.progress();
        } catch (InvalidActionException e) {
            log.error("Caught exception trying to transition issue" + e.getMessage());
        }
    }
}
```

Requirement projects

Although you can go to Xray settings, in the "Requirement Projects" tab, and define a project as containing requirement issues, sometimes you may want to automate this.

The following script shows several functions that you can use to obtain the list of projects with requirement coverage enabled and to set or unset a project as being a requirements project.

requirement_projects.groovy

```
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.link.IssueLinkManager
import com.atlassian.jira.issue.link.IssueLinkType
import com.atlassian.jira.issue.link.IssueLinkTypeManager
import com.atlassian.jira.ComponentManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.jql.builder.JqlQueryBuilder
import com.atlassian.jira.user.util.UserUtil
import com.atlassian.jira.user.util.UserManager;
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.search.SearchService;
import com.atlassian.jira.issue.search.SearchProvider
import com.atlassian.jira.issue.search.SearchResults
import com.atlassian.jira.web.bean.PagerFilter;
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.user.UserPropertyManager
import com.atlassian.jira.propertyset.JiraPropertySetFactory;
import com.google.common.collect.ImmutableMap;
import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.module.propertyset.PropertySetManager;

ENTITY_NAME = "com.xpandit.raven";
ENTITY_ID = 12345678987654321L;
REQUIREMENT_PROJECTS_SETTING = "requirement-coverage.projects";

projectManager = ComponentAccessor.getProjectManager()

Object obtainRequirementProjectsIds() {
        JiraPropertySetFactory jiraPropertySetFactory = ComponentAccessor.getComponent(JiraPropertySetFactory.
class);
        def setting = jiraPropertySetFactory.buildCachingPropertySet(ENTITY_NAME, ENTITY_ID, true);
        def requirementProjects = Eval.me(setting.getText(REQUIREMENT_PROJECTS_SETTING))
}


Object obtainRequirementProjects() {
        def requirementProjects = obtainRequirementProjectsIds()
    log.debug("requirementProjects: "+requirementProjects)
    def availableProjects = projectManager.getProjectObjects()
    availableProjects.findAll { it.id.toInteger() in requirementProjects}
```

```
}

boolean enableRequirementCoverageForProject(project){
         JiraPropertySetFactory jiraPropertySetFactory = ComponentAccessor.getComponent(JiraPropertySetFactory.
class);
        def setting = jiraPropertySetFactory.buildCachingPropertySet(ENTITY_NAME, ENTITY_ID, true);
    projectList = obtainRequirementProjectsIds()
    if (!projectList.contains(project.id.toInteger())){
                setting.setText(REQUIREMENT_PROJECTS_SETTING,(projectList << project.id).toString())
    }
}

boolean disableRequirementCoverageForProject(project){
         JiraPropertySetFactory jiraPropertySetFactory = ComponentAccessor.getComponent(JiraPropertySetFactory.
class);
        def setting = jiraPropertySetFactory.buildCachingPropertySet(ENTITY_NAME, ENTITY_ID, true);
    projectList = obtainRequirementProjectsIds()
    if (projectList.contains(project.id.toInteger())){
        projectList.removeAll{it == project.id.toInteger()}
                setting.setText(REQUIREMENT_PROJECTS_SETTING,projectList.toString())
    }
}



boolean requirementCoverageEnabledForProject(project){
        JiraPropertySetFactory jiraPropertySetFactory = ComponentAccessor.getComponent(JiraPropertySetFactory.
class);
        def setting = jiraPropertySetFactory.buildCachingPropertySet(ENTITY_NAME, ENTITY_ID, true);
        def requirementProjects = Eval.me(setting.getText(REQUIREMENT_PROJECTS_SETTING))
    (project.id.toInteger() in requirementProjects)
}



projects = obtainRequirementProjects()
project = projectManager.getProjectObjByKey("CALC")
log.debug(requirementCoverageEnabledForProject(project))
enableRequirementCoverageForProject(project)
log.debug(requirementCoverageEnabledForProject(project))
```

Create Test Execution from Test Set issue screen

In this example, we're adding a new option in the "More" menu, by adding a new "web section", "web item" ScriptRunner elements.

The following script will create a Test Execution containing all the Tests that are part of the current Test Set.

It will set some fields as read-only.

create_test_exec_from_testset.groovy

```
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.link.IssueLinkManager
import com.atlassian.jira.issue.link.IssueLinkType
import com.atlassian.jira.issue.link.IssueLinkTypeManager
import com.atlassian.jira.ComponentManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.jql.builder.JqlQueryBuilder
import com.atlassian.jira.user.util.UserUtil
import com.atlassian.jira.user.util.UserManager;
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.search.SearchService
import com.atlassian.jira.issue.search.SearchProvider
import com.atlassian.jira.issue.search.SearchResults
import com.atlassian.jira.web.bean.PagerFilter;
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.user.UserPropertyManager
import com.atlassian.jira.propertyset.JiraPropertySetFactory;
import com.google.common.collect.ImmutableMap;
import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.module.propertyset.PropertySetManager;
import com.atlassian.jira.util.BuildUtils
import com.atlassian.jira.util.BuildUtilsInfo
import com.atlassian.jira.util.BuildUtilsInfoImpl
import com.atlassian.plugin.PluginAccessor
import com.atlassian.plugin.PluginManager
import com.atlassian.jira.bc.license.JiraLicenseService
import com.atlassian.jira.bc.license.JiraLicenseServiceImpl
import org.apache.log4j.Level
import org.apache.log4j.Logger
import com.atlassian.jira.issue.IssueManager


// creates a Test Execution from a Test Set and fillouts the summary, as readonly, and the Tests associated

// projectManager = ComponentAccessor.getProjectManager()
issueManager = ComponentAccessor.getIssueManager()
searchService =  ComponentAccessor.getComponent(SearchService.class);
serviceAccount = ComponentAccessor.getJiraAuthenticationContext().getLoggedInUser()


Object getIssues(jqlQuery){
    // A list of GenericValues representing issues
    List<Issue> searchResults = null;

    SearchService.ParseResult parseResult =  searchService.parseQuery(serviceAccount, jqlQuery);

    if (parseResult.isValid()) {
        // throws SearchException
        SearchResults results = searchService.search(serviceAccount, parseResult.getQuery(), PagerFilter.
getUnlimitedFilter());
        searchResults = results.getIssues();
        return searchResults;
    }

     return []
}

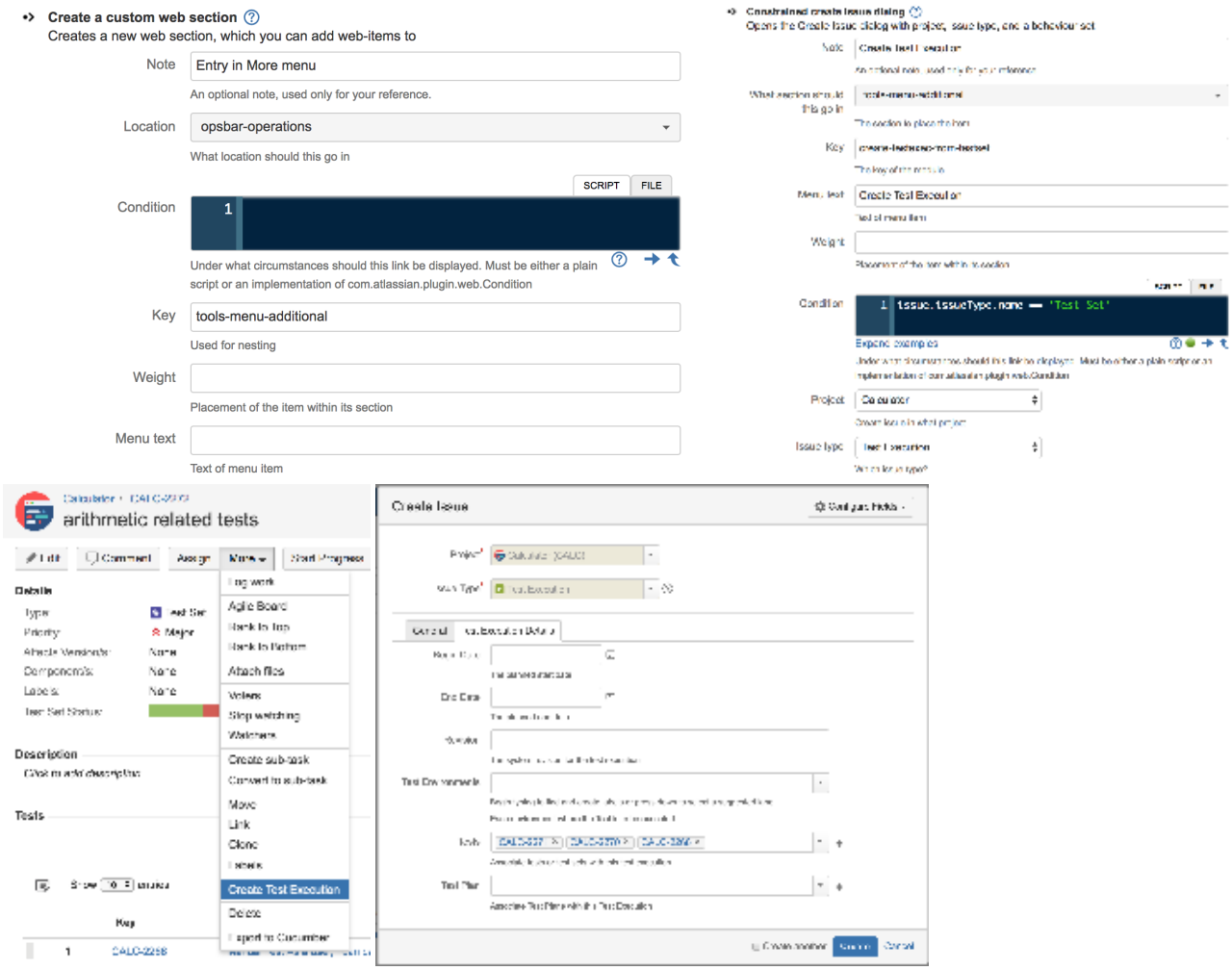

if (getBehaviourContextId() == "create-testexec-from-testset") {
    getFieldById("project-field").setReadOnly(true)
    getFieldById("issuetype-field").setReadOnly(true)

    def contextIssue = issueManager.getIssueObject(getContextIssueId())
    getFieldById("summary").setFormValue("Issue created from ${contextIssue.key}").setReadOnly(true)

    def jql = "issue in testSetTests('${contextIssue.key}')"
    def issues = getIssues(jql)

    getFieldByName("Tests association with a Test Execution").setFormValue(issues.collect { it.key})
}
```

As a mere example, you can see below how it could be setup, by creating a web section and then a webitem to trigger the actual script for creating the Test Execution.



Calculate requirement status for a certain version

Xray provides the "Requirement Status" custom field that shows the calculated coverage status for some version(s) depending on the configuration of that field in Xray's Custom Fields Preferences settings.

Sometimes you may need to evaluate the calculate requirement coverage status on some specific version, based on the executions made for that version.

A possible use case could be defining a scripted field present on the requirement issue screen that calculates the coverage status for some specific hardcoded version.

calculate_requirement_status_for_some_version.groovy

```
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.link.IssueLinkManager
import com.atlassian.jira.issue.link.IssueLinkType
import com.atlassian.jira.issue.link.IssueLinkTypeManager
import com.atlassian.jira.ComponentManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.jql.builder.JqlQueryBuilder
import com.atlassian.jira.user.util.UserUtil
import com.atlassian.jira.user.util.UserManager;
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.search.SearchService;
```

```
import com.atlassian.jira.issue.search.SearchProvider
import com.atlassian.jira.issue.search.SearchResults
import com.atlassian.jira.web.bean.PagerFilter;
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.user.UserPropertyManager
import com.atlassian.jira.propertyset.JiraPropertySetFactory;
import com.google.common.collect.ImmutableMap;
import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.module.propertyset.PropertySetManager;
import com.atlassian.jira.util.BuildUtils
import com.atlassian.jira.util.BuildUtilsInfo
import com.atlassian.jira.util.BuildUtilsInfoImpl
import com.atlassian.plugin.PluginAccessor
import com.atlassian.plugin.PluginManager
import com.atlassian.jira.bc.license.JiraLicenseService
import com.atlassian.jira.bc.license.JiraLicenseServiceImpl
import org.apache.log4j.Level
import org.apache.log4j.Logger
import com.atlassian.jira.issue.IssueManager


Object getIssues(jqlQuery){
    // A list of GenericValues representing issues
    List<Issue> searchResults = null;

    SearchService.ParseResult parseResult =  searchService.parseQuery(serviceAccount, jqlQuery);

    if (parseResult.isValid()) {
        // throws SearchException
        SearchResults results = searchService.search(serviceAccount, parseResult.getQuery(), PagerFilter.
getUnlimitedFilter());
        searchResults = results.getIssues();
        return searchResults;
    }

     return []
}

projectManager = ComponentAccessor.getProjectManager()
componentManager = ComponentManager.getInstance();
searchService =  ComponentAccessor.getComponent(SearchService.class);
issueManager = ComponentAccessor.getIssueManager()
customFieldManager = ComponentAccessor.getCustomFieldManager()
userUtil = ComponentAccessor.getUserUtil();
serviceAccount = ComponentAccessor.getJiraAuthenticationContext().getLoggedInUser()
pluginAccessor = componentManager.getPluginAccessor();

Issue issue = issue;

log.debug(issue.project.name)
projectName = issue.project.name
version = 'v3.0'

statuses = ['OK','NOK','NOTRUN','UNCOVERED','UNKNOWN']
def status = ''


statuses.find {
    jql = "key = ${issue.key} and issue in requirements('${it}','${projectName}','${version}')"
    issues = getIssues(jql)
    count = issues.size
    if (count > 0) {
        status = it;
        return true; // break
    } else {
            return false;
    }
}

status
```

Show Tests Count for a requirement

In the following example, a "script field" is used to show the total amount of linked Tests to a given requirement.

total_linked_tests_to_requirement.groovy

```
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.link.IssueLinkManager
import com.atlassian.jira.issue.link.IssueLinkType
import com.atlassian.jira.issue.link.IssueLinkTypeManager
import com.atlassian.jira.ComponentManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.jql.builder.JqlQueryBuilder
import com.atlassian.jira.user.util.UserUtil
import com.atlassian.jira.user.util.UserManager;
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.search.SearchService;
import com.atlassian.jira.issue.search.SearchProvider
import com.atlassian.jira.issue.search.SearchResults
import com.atlassian.jira.web.bean.PagerFilter;
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.user.UserPropertyManager
import com.atlassian.jira.propertyset.JiraPropertySetFactory;
import com.google.common.collect.ImmutableMap;
import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.module.propertyset.PropertySetManager;
import com.atlassian.jira.util.BuildUtils
import com.atlassian.jira.util.BuildUtilsInfo
import com.atlassian.jira.util.BuildUtilsInfoImpl
import com.atlassian.plugin.PluginAccessor
import com.atlassian.plugin.PluginManager
import com.atlassian.jira.bc.license.JiraLicenseService
import com.atlassian.jira.bc.license.JiraLicenseServiceImpl
import org.apache.log4j.Level
import org.apache.log4j.Logger
import com.atlassian.jira.issue.IssueManager


Object getIssues(jqlQuery){
    // A list of GenericValues representing issues
    List<Issue> searchResults = null;

    SearchService.ParseResult parseResult =  searchService.parseQuery(serviceAccount, jqlQuery);

    if (parseResult.isValid()) {
        // throws SearchException
        SearchResults results = searchService.search(serviceAccount, parseResult.getQuery(), PagerFilter.
getUnlimitedFilter());
        searchResults = results.getIssues();
        return searchResults;
    }

     return []
}

projectManager = ComponentAccessor.getProjectManager()
componentManager = ComponentManager.getInstance();
searchService =  ComponentAccessor.getComponent(SearchService.class);
issueManager = ComponentAccessor.getIssueManager()
customFieldManager = ComponentAccessor.getCustomFieldManager()
userUtil = ComponentAccessor.getUserUtil();
serviceAccount = ComponentAccessor.getJiraAuthenticationContext().getLoggedInUser()
pluginAccessor = componentManager.getPluginAccessor();

IssueManager iManager = ComponentAccessor.getIssueManager();
Issue issue = issue;

jql = "issue in requirementTests('${issue.key}')"
issues = getIssues(jql)
issues.size
```

Show Defects Count for a requirement

In the following example, a "script field" is used to show the total amount of linked defects to a given requirement and also provide a link to easily obtain those defects in the Issue search page.

This script field is configured to generate the output as HTML.

total_linked_tests_to_requirement.groovy

```
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.link.IssueLinkManager
import com.atlassian.jira.issue.link.IssueLinkType
import com.atlassian.jira.issue.link.IssueLinkTypeManager
import com.atlassian.jira.ComponentManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.jql.builder.JqlQueryBuilder
import com.atlassian.jira.user.util.UserUtil
import com.atlassian.jira.user.util.UserManager;
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.search.SearchService;
import com.atlassian.jira.issue.search.SearchProvider
import com.atlassian.jira.issue.search.SearchResults
import com.atlassian.jira.web.bean.PagerFilter;
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.user.UserPropertyManager
import com.atlassian.jira.propertyset.JiraPropertySetFactory;
import com.google.common.collect.ImmutableMap;
import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.module.propertyset.PropertySetManager;
import com.atlassian.jira.util.BuildUtils
import com.atlassian.jira.util.BuildUtilsInfo
import com.atlassian.jira.util.BuildUtilsInfoImpl
import com.atlassian.plugin.PluginAccessor
import com.atlassian.plugin.PluginManager
import com.atlassian.jira.bc.license.JiraLicenseService
import com.atlassian.jira.bc.license.JiraLicenseServiceImpl
import org.apache.log4j.Level
import org.apache.log4j.Logger
import com.atlassian.jira.issue.IssueManager


Object getIssues(jqlQuery){
    // A list of GenericValues representing issues
    List<Issue> searchResults = null;

    SearchService.ParseResult parseResult =  searchService.parseQuery(serviceAccount, jqlQuery);

    if (parseResult.isValid()) {
        // throws SearchException
        SearchResults results = searchService.search(serviceAccount, parseResult.getQuery(), PagerFilter.
getUnlimitedFilter());
        searchResults = results.getIssues();
        return searchResults;
    }

     return []
}

projectManager = ComponentAccessor.getProjectManager()
componentManager = ComponentManager.getInstance();
searchService =  ComponentAccessor.getComponent(SearchService.class);
issueManager = ComponentAccessor.getIssueManager()
customFieldManager = ComponentAccessor.getCustomFieldManager()
userUtil = ComponentAccessor.getUserUtil();
serviceAccount = ComponentAccessor.getJiraAuthenticationContext().getLoggedInUser()
pluginAccessor = componentManager.getPluginAccessor();

IssueManager iManager = ComponentAccessor.getIssueManager();
Issue issue = issue;

jql = "issue in defectsCreatedForRequirement('${issue.key}')"
issues = getIssues(jql)
"<a href='/issues/?jql=issue%20in%20defectsCreatedForRequirement(${issue.key})'>${issues.size}</a>"
```

Configured Issue Types as being requirements or defects

If you need to obtain the issue types configured as being handled as requirements or as defects by Xray, you may use the following script.

With some additional code, you may filter this out based on the issue types available in some certain project.

requirement_and_defect_issue_types.groovy

```
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.link.IssueLinkManager
import com.atlassian.jira.issue.link.IssueLinkType
import com.atlassian.jira.issue.link.IssueLinkTypeManager
import com.atlassian.jira.ComponentManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.jql.builder.JqlQueryBuilder
import com.atlassian.jira.user.util.UserUtil
import com.atlassian.jira.user.util.UserManager;
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.search.SearchService;
import com.atlassian.jira.issue.search.SearchProvider
import com.atlassian.jira.issue.search.SearchResults
import com.atlassian.jira.web.bean.PagerFilter;
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.user.UserPropertyManager
import com.atlassian.jira.propertyset.JiraPropertySetFactory;
import com.google.common.collect.ImmutableMap;
import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.module.propertyset.PropertySetManager;

ENTITY_NAME = "com.xpandit.raven";
ENTITY_ID = 12345678987654321L;
REQUIREMENT_ISSUE_TYPES = "issue-type-mapping.requirements";
DEFECT_ISSUE_TYPES = "issue-type-mapping.defects";

user = ComponentAccessor.jiraAuthenticationContext.getLoggedInUser()
constantsManager = ComponentAccessor.getConstantsManager()
availableIssueTypes = constantsManager.getAllIssueTypeObjects()


Object getRequirementIssueTypes(){
    JiraPropertySetFactory jiraPropertySetFactory = ComponentAccessor.getComponent(JiraPropertySetFactory.
class);
    setting = jiraPropertySetFactory.buildCachingPropertySet(ENTITY_NAME, ENTITY_ID, true);
    configuredIssueTypes = Eval.me(setting.getText(REQUIREMENT_ISSUE_TYPES))
    availableIssueTypes.findAll{it.id.toInteger() in configuredIssueTypes}
}

Object getDefectIssueTypes(){
    JiraPropertySetFactory jiraPropertySetFactory = ComponentAccessor.getComponent(JiraPropertySetFactory.
class);
    setting = jiraPropertySetFactory.buildCachingPropertySet(ENTITY_NAME, ENTITY_ID, true);
    configuredIssueTypes = Eval.me(setting.getText(DEFECT_ISSUE_TYPES))
    availableIssueTypes.findAll{it.id.toInteger() in configuredIssueTypes}
}

getRequirementIssueTypes()
getDefectIssueTypes()
```

Trigger a Jenkins project build from a Test Plan

In scenarios with CI implemented, you may want to trigger certain Jenkins "jobs" (i.e. project build) from a Test Plan and link back the result to Xray.

In this example, we're assuming that the list of automated tests that will be run is managed in the CI side, depending on the project configuration.

The results will be submitted back to Xray, if the project is configured to do so in Jenkins.

In order to add this option in Jira's UI, we'll need to add a custom "web item" that provides an action that will interact with a custom ScriptRunner endpoint, which will be the one doing the HTTP request to the Jenkins server, passing the Test Plan issue key. In order to submit the request to Jenkins, we need to obtain Jenkins username and respective API token along with the project-specific authentication token.

trigger_jenkins_build_restapi_endpoint.groovy

```
import com.onresolve.scriptrunner.runner.rest.common.CustomEndpointDelegate
import groovy.json.JsonOutput
import groovy.transform.BaseScript
import groovy.json.JsonSlurper;
import groovy.json.StreamingJsonBuilder;
import javax.ws.rs.core.MultivaluedMap
import javax.ws.rs.core.Response

@BaseScript CustomEndpointDelegate delegate

triggerJenkinsBuild(httpMethod: "GET") { MultivaluedMap queryParams ->

        def issueId = queryParams.getFirst("issueId") as String // use the issueId to retrieve this issue

        def flag = [
        type : 'success',
        title: "Build scheduled",
        close: 'auto',
        body : "A new build has been scheduled related with "+issueId
        ]


        URL url;
        def jobName = "java-junit-calc"                         // could come from a CF in the Test Plan
        def jenkinsHostPort = "192.168.56.102:8081"             // could be defined elsewhere
        def token = "iFBDOBhNhaxL4T9ass93HRXun2JF161Z"                  // could also come from a CF in the
Test Plan
        def username = "admin"                                                                          //
probably, would need to be stored elsewhere
        def password = "fa02840152aa2e4da3d8db933ec708d6"               // probably, would need to be stored
elsewhere
        def baseURL = "http://${jenkinsHostPort}/job/${jobName}/buildWithParameters?token=${token}
&TESTPLAN=$issueId"

        url = new URL(baseURL);
        def body_req = []

        def authString = "${username}:${password}".bytes.encodeBase64().toString()

        URLConnection connection = url.openConnection();
        connection.requestMethod = "POST"
        connection.doOutput = true
        connection.addRequestProperty("Authorization", "Basic ${authString}")
        connection.setRequestProperty("Content-Type", "application/json;charset=UTF-8")
        connection.outputStream.withWriter("UTF-8") { new StreamingJsonBuilder(it, body_req) }
        connection.connect();
        log.debug(connection.getResponseCode())
        log.debug(connection.getResponseMessage())


        if (connection.getResponseCode() == 201) {
                 Response.ok(JsonOutput.toJson(flag)).build()
         } else {
                 //Response.status(Response.Status.NOT_FOUND).entity("Problem scheduling job!").build();
         }

}
```



Calculator / CALC-2283

CLONE - all my relevant tests for v3.0

Edit | Comment | Trigger Jenkins Build | More | Start Progress | Resolve Issue | Close Issue | Admin

Details

Type:	Test Plan	Status:	OPEN (View Workflow)
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	v3.0
Component/s:	None		
Labels:	None		

Example

ScriptRunner configuration

Custom web item

Creates a web item (a button or link) at the location you specify

Note: Trigger Jenkins build

An optional note, used only for your reference.

What section should this go in: operations-top-level

The section to place the item

Key: trigger-jenkins-build-button

The key of the module

Menu text: Trigger Jenkins Build

Text of menu item

Weight: 1

Placement of the item within its section

Condition: SCRIPT | FILE

```
1 issue.issueType.name == 'Test Plan'
```

Expand examples

Under what circumstances should this link be displayed. Must be either a plain script or an implementation of com.atlassian.plugin.web.Condition

Do what: Run code and show a flag

What do you want to do when the item is clicked

Link: /rest/scriptrunner/latest/custom/triggerJenkinsBuild?issueId=${issue.key}

The relative URL to direct to. If external, include the scheme, eg http://google.com

Custom endpoint

Define a REST endpoint in a script

Note: jenkins build

An optional note, used only for your reference.

Script file: Start typing to search for script files...

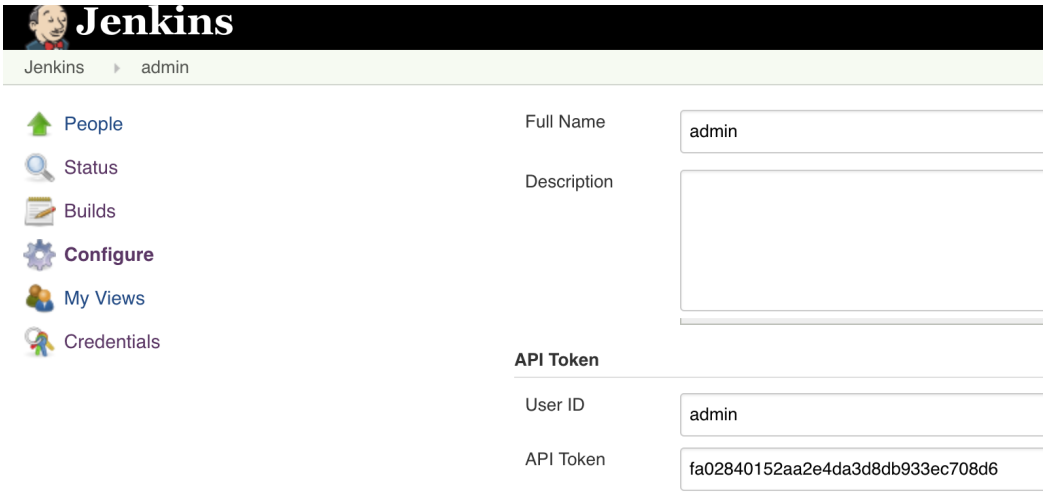
Path to the script file accessible on the server

Inline script

```
1  import com.onresolve.scriptrunner.runner.rest.commo
2  import groovy.json.JsonOutput
3  import groovy.transform.BaseScript
4  import groovy.json.JsonSlurper;
5  import groovy.json.StreamingJsonBuilder;
6  import javax.ws.rs.core.MultivaluedMap
7  import javax.ws.rs.core.Response
8
9  @BaseScript CustomEndpointDelegate delegate
10
11 triggerJenkinsBuild(httpMethod: "GET") { Multivalue
12
```

Jenkins configuration

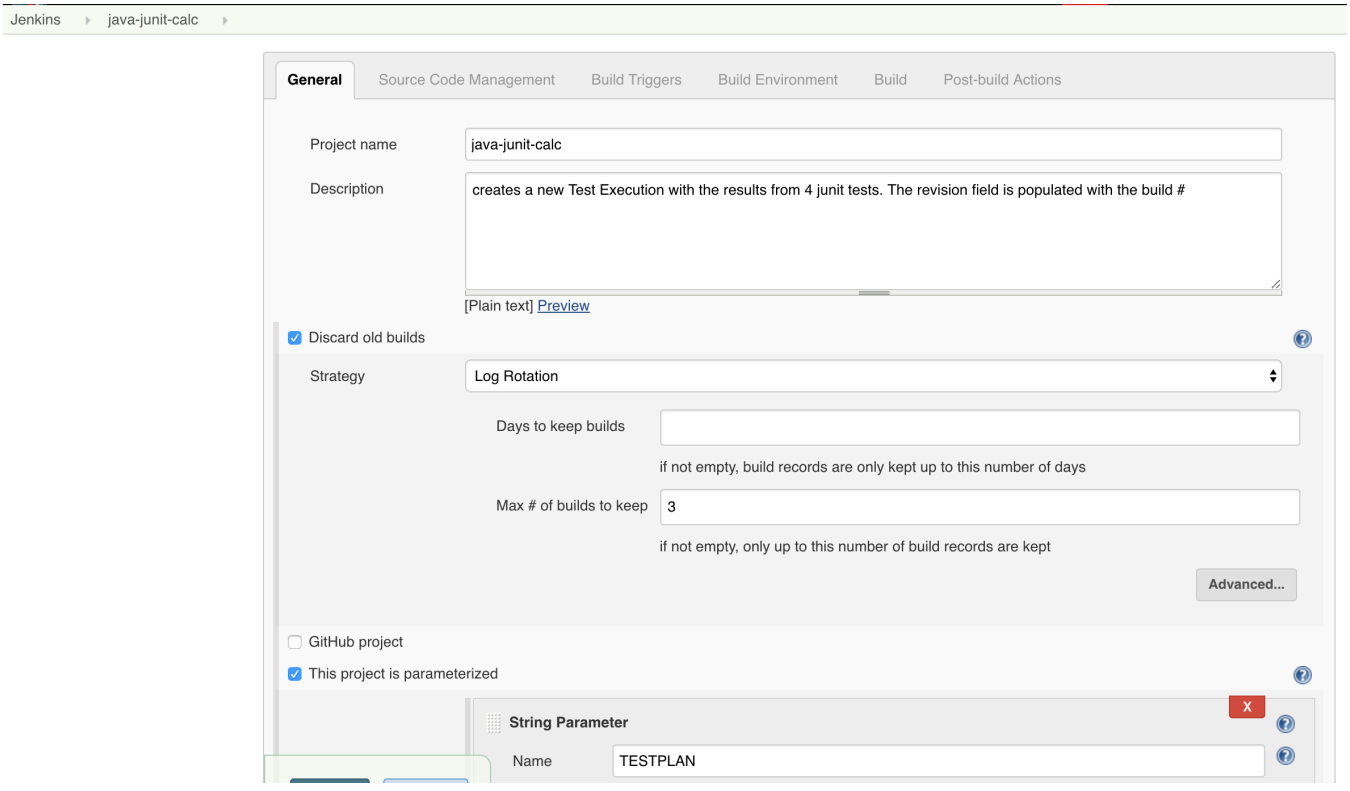
In Jenkins, we need to generate an API token for some user, which can be done from the profile settings page.



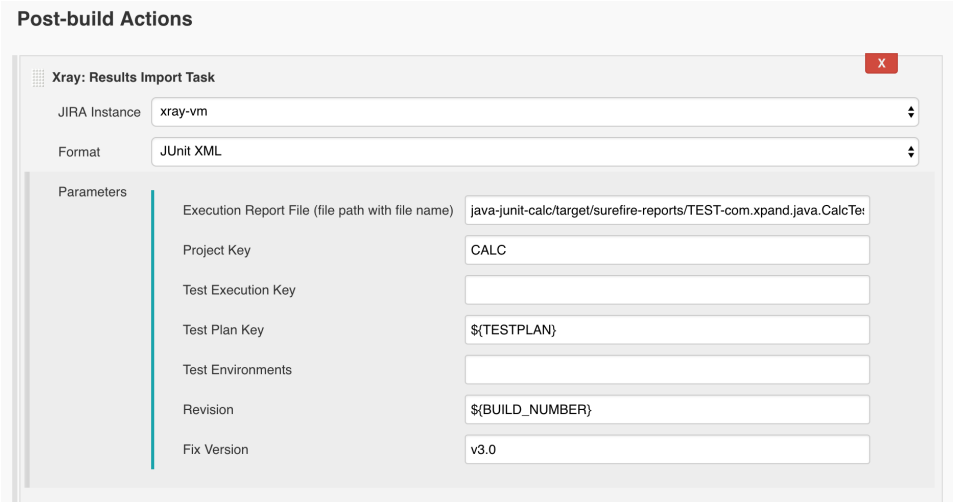
At the project level, we need to enable remote build triggers, so we can obtain an "authentication token" to be used in the HTTP request afterwards.



The project itself is a normal one; the only thing relevant to mention is that this project is a parameterized one, so it receives a TESTPLAN variable, that in our case will be coming from Jira.



The final task submits the results linking the Test Execution to the Test Plan passed as argument.



Trigger a Jenkins project build from a Test Plan, for the Tests contained in the Test Plan

This scenario is somehow similar to the previous one, except that the list of Tests that will be run in the CI side will be based on the Tests contained in the Test Plan.

The results will be submitted back to Xray, if the project is configured to do so in Jenkins.

In order to add this option in Jira's UI, we'll need to add a custom "web item" that provides an action that will interact with a custom ScriptRunner endpoint, which will be the one doing the HTTP request to the Jenkins server, passing the Test Plan issue key. In the ScriptRunner endpoint script, we'll obtain the list of Generic Tests (we're assuming that they will come from Junit, so they have a certain syntax in the Generic Test Definition field.

In order to submit the request to Jenkins, we need to obtain Jenkins username and respective API token along with the project-specific authentication token.

trigger_jenkins_build_restapi_endpoint_with_testlist.groovy

```
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.link.IssueLinkManager
import com.atlassian.jira.issue.link.IssueLinkType
import com.atlassian.jira.issue.link.IssueLinkTypeManager
import com.atlassian.jira.ComponentManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.jql.builder.JqlQueryBuilder
import com.atlassian.jira.user.util.UserUtil
import com.atlassian.jira.user.util.UserManager;
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.search.SearchService;
import com.atlassian.jira.issue.search.SearchProvider
import com.atlassian.jira.issue.search.SearchResults
import com.atlassian.jira.web.bean.PagerFilter;
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.user.UserPropertyManager
import com.atlassian.jira.propertyset.JiraPropertySetFactory;
import com.google.common.collect.ImmutableMap;
import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.module.propertyset.PropertySetManager;
import com.atlassian.jira.util.BuildUtils
import com.atlassian.jira.util.BuildUtilsInfo
import com.atlassian.jira.util.BuildUtilsInfoImpl
import com.atlassian.plugin.PluginAccessor
import com.atlassian.plugin.PluginManager
import com.atlassian.jira.bc.license.JiraLicenseService
import com.atlassian.jira.bc.license.JiraLicenseServiceImpl
import org.apache.log4j.Level
import org.apache.log4j.Logger
import com.atlassian.jira.issue.IssueManager
```

```
import com.opensymphony.workflow.InvalidInputException

import com.onresolve.scriptrunner.runner.rest.common.CustomEndpointDelegate
import groovy.json.JsonOutput
import groovy.transform.BaseScript
import groovy.json.JsonSlurper;
import groovy.json.StreamingJsonBuilder;
import javax.ws.rs.core.MultivaluedMap
import javax.ws.rs.core.Response

@BaseScript CustomEndpointDelegate delegate


issueManager = ComponentAccessor.getIssueManager()
searchService =  ComponentAccessor.getComponent(SearchService.class);
serviceAccount = ComponentAccessor.getJiraAuthenticationContext().getLoggedInUser()
customFieldManager = ComponentAccessor.getCustomFieldManager()

Object getIssues(jqlQuery){
    // A list of GenericValues representing issues
    List<Issue> searchResults = null;

    SearchService.ParseResult parseResult =  searchService.parseQuery(serviceAccount, jqlQuery);

    if (parseResult.isValid()) {
        // throws SearchException
        SearchResults results = searchService.search(serviceAccount, parseResult.getQuery(), PagerFilter.
getUnlimitedFilter());
        searchResults = results.getIssues();
        return searchResults;
    }

     return []
}


Object getFieldValue(issue,customField) {
    //def cField = customFieldManager.getCustomFieldObject(customField)
    def cField = customFieldManager.getCustomFieldObjectByName(customField)
    def cFieldValue = issue.getCustomFieldValue(cField)
    return cFieldValue
}

String replaceLast(String string, String substring, String replacement)
{
  int index = string.lastIndexOf(substring);
  if (index == -1)
    return string;
  return string.substring(0, index) + replacement + string.substring(index+substring.length());
}


triggerJenkinsBuildWithTestList(httpMethod: "GET") { MultivaluedMap queryParams ->

        // the details of getting and modifying the current issue are ommitted for brevity
        def issueId = queryParams.getFirst("issueId") as String // use the issueId to retrieve this issue


        def flag = [
        type : 'success',
        title: "Build scheduled",
        close: 'auto',
        body : "A new build has been scheduled related with "+issueId
        ]


        URL url;
        def jobName = "java-junit-calc-triggered"               // could be defined in a CF in the Test Plan
        def jenkinsHostPort = "192.168.56.102:8081"             // could be defined elsewhere
        def token = "iFBDOBhNhaxL4T9ass93HRXun2JF161Z"                  // could also come from a CF in the
Test Plan
```

```
        def username = "admin"                                                                          //
probably, would need to be stored elsewhere
        def password = "fa02840152aa2e4da3d8db933ec708d6"               // probably, would need to be stored
elsewhere
        //def baseURL = "http://${username}:${password}@${jenkinsHostPort}/job/${jobName}/build?token=${token}"
        //def baseURL = "http://${jenkinsHostPort}/job/${jobName}/build?token=${token}"

        jql = "issue in testPlanTests('${issueId}') and \"Test Type\" = Generic"
        issues = getIssues(jql)
    // we're assuming that we have Junit based Tests.. so we need to do some conversion beforehand, so maven
can process the list of tests to be run
        def testlist = issues.collect { getFieldValue(it,"Generic Test Definition")}
        def testlist2 = testlist.collect { replaceLast(it,".","%23") }

        def baseURL = "http://${jenkinsHostPort}/job/${jobName}/buildWithParameters?token=${token}
&TESTPLAN=${issueId}&TESTLIST=${testlist2.join(',')}"
        url = new URL(baseURL);
        def body_req = []

        def authString = "${username}:${password}".bytes.encodeBase64().toString()

        URLConnection connection = url.openConnection();
        connection.requestMethod = "POST"
        connection.doOutput = true
        connection.addRequestProperty("Authorization", "Basic ${authString}")
        connection.setRequestProperty("Content-Type", "application/json;charset=UTF-8")
        connection.outputStream.withWriter("UTF-8") { new StreamingJsonBuilder(it, body_req) }
        connection.connect();
        //connection.getContent();
        log.debug(connection.getResponseCode())
        log.debug(connection.getResponseMessage())


        if (connection.getResponseCode() == 201) {
                Response.ok(JsonOutput.toJson(flag)).build()
         } else {
                //Response.status(Response.Status.NOT_FOUND).entity("Problem scheduling job!").build();
         }

}
```

Example

ScriptRunner configuration

Custom web item

Creates a web item (a button or link) at the location you specify

Note: Trigger Jenkins build with Test List

An optional note, used only for your reference.

What section should this go in: operations-top-level

The section to place the item

Key: trigger-jenkins-build-with-params-button

The key of the module

Menu text: Trigger Jenkins Build with these Tests

Text of menu item

Weight: 2

Placement of the item within its section

Condition: SCRIPT | FILE

```
1  issue.issueType.name == 'Test Plan'
```

Expand examples

Under what circumstances should this link be displayed. Must be either a plain script or an implementation of com.atlassian.plugin.web.Condition

Do what: Run code and show a flag

What do you want to do when the item is clicked

Link: /rest/scriptrunner/latest/custom/triggerJenkinsBuildWithTestList?issueId=${issu

The relative URL to direct to. If external, include the scheme, eg http://google.com

Custom endpoint

Define a REST endpoint in a script

Note

An optional note, used only for your reference.

Script file: Start typing to search for script files...

Path to the script file accessible on the server

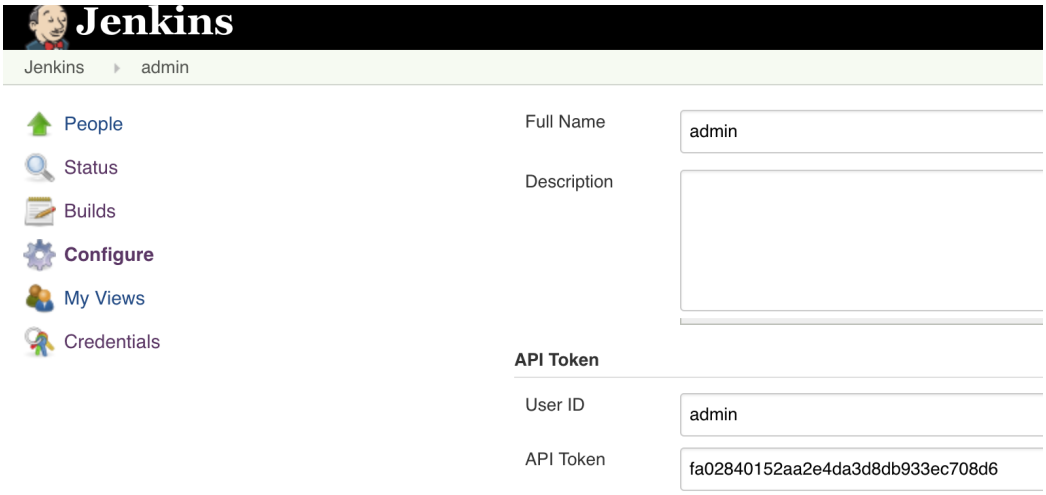
Inline script

```
79  }
80
81
82  triggerJenkinsBuildWithTestList(httpMethod: "GET")
83
84      // the details of getting and modifying the cu
85      def issueId = queryParams.getFirst("issueId")
86
87
88
89
90
91      def flag = [
92
```

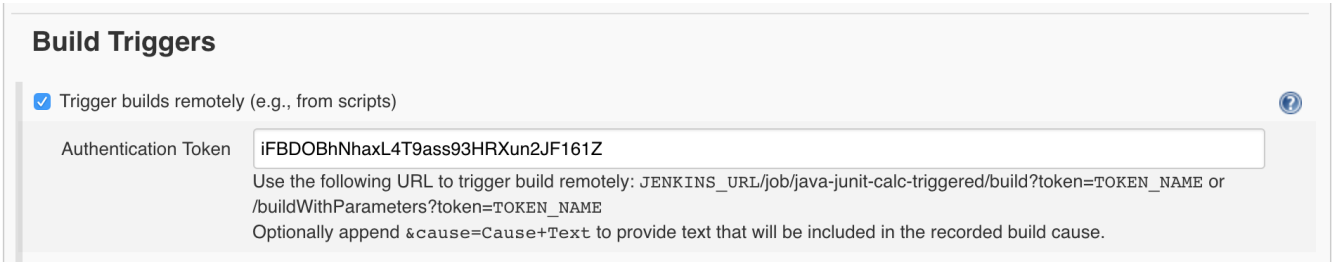
Enter your script here

Jenkins configuration

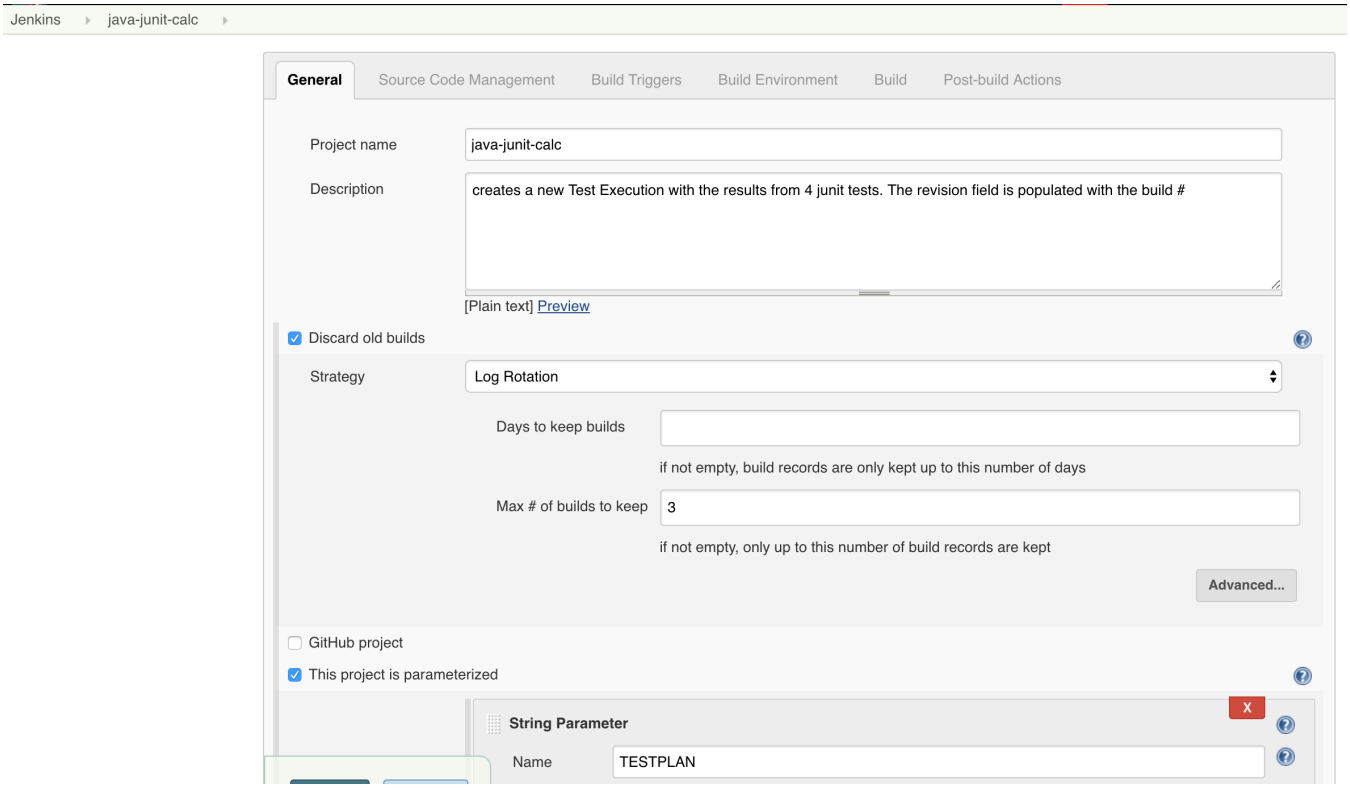
In Jenkins, we need to generate an API token for some user, which can be done from the profile settings page.



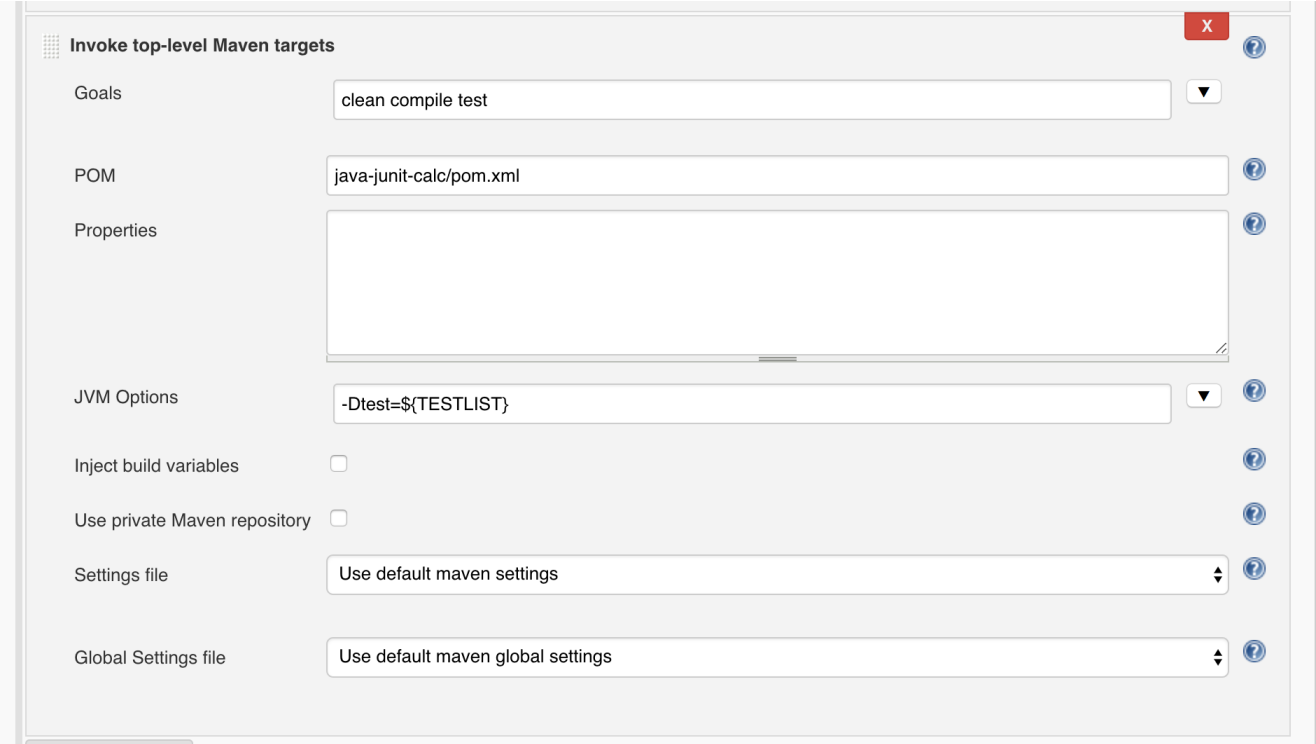
At the project level, we need to enable remote build triggers, so we can obtain an "authentication token" to be used in the HTTP request afterwards.



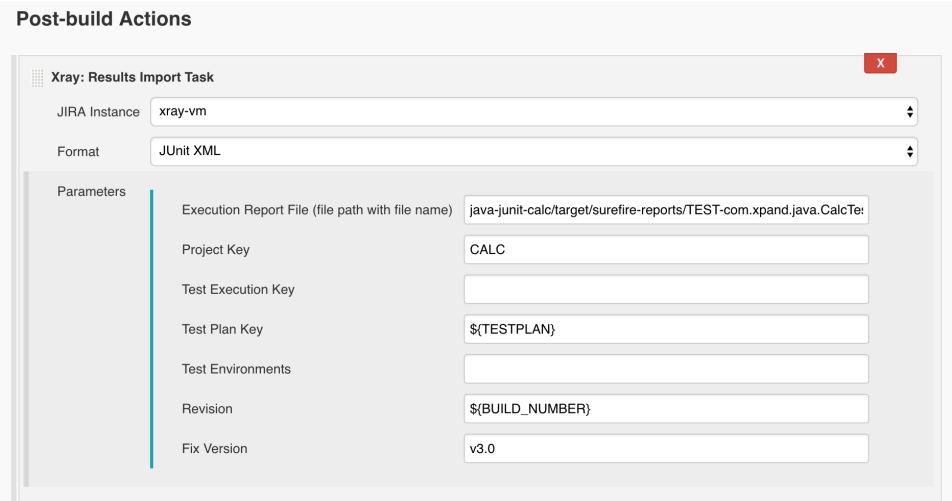
The project itself is a normal one; the only thing relevant to mention is that this project is a parameterized one, so it receives TESTPLAN and TESTLIST variables, that in our case will be coming from Jira.



Maven is configured in order to run just the tests identified in the TESTLIST variable, using the "-Dtest" JVM option.



The final task submits the results linking the Test Execution to the Test Plan passed as argument.



Trigger a Bamboo plan build from a Test Plan

In scenarios with CI implemented, you may want to trigger certain Bamboo "plans" (i.e. builds) from a Test Plan and link back the result to Xray.

In this example, we're assuming that the list of automated tests that will be run is managed in the CI side, depending on the plan configuration.

The results will be submitted back to Xray, if the project is configured to do so in Bamboo.

In order to add this option in Jira's UI, we'll need to add a custom "web item" that provide an action that will interact with a custom ScriptRunner endpoint, which will be the one doing the HTTP request to the Bamboo server, passing the Test Plan issue key. In order to submit the request to Bamboo we just need to use the credentials of some user.

trigger_bamboo_build_restapi_endpoint.groovy

```
import com.onresolve.scriptrunner.runner.rest.common.CustomEndpointDelegate
import groovy.json.JsonOutput
import groovy.transform.BaseScript
import groovy.json.JsonSlurper;
import groovy.json.StreamingJsonBuilder;
import javax.ws.rs.core.MultivaluedMap
import javax.ws.rs.core.Response
import java.nio.charset.StandardCharsets

@BaseScript CustomEndpointDelegate delegate

triggerBambooBuild(httpMethod: "GET") { MultivaluedMap queryParams ->

        def issueId = queryParams.getFirst("issueId") as String // use the issueId to retrieve this issue

        def flag = [
        type : 'success',
        title: "Build scheduled",
        close: 'auto',
        body : "A new build has been scheduled related with "+issueId
        ]


        URL url;
        // curl --user admin:admin -X POST -d "default&ExecuteAllStages=true" http://yourbambooserver/rest/api
/latest/queue/XRAY-JUNITCALC
        def projectKey = "XRAY"                                                                          // could
come from a CF in the Test Plan
    def planKey = "JUNITCALC"                                                                  // could come from
a CF in the Test Plan
        def bambooHostPort = "192.168.56.102:8085"                                 // could be defined elsewhere
        def username = "admin"                                                                           //
probably, would need to be stored elesewhere
        def password = "admin"                                                                           //
probably, would need to be stored elesewhere
        def baseURL = "http://${bambooHostPort}/rest/api/latest/queue/${projectKey}-${planKey}"
    String urlParameters  = "default&ExecuteAllStages=true&bamboo.TESTPLAN=${issueId}";
    byte[] postData       = urlParameters.getBytes( StandardCharsets.UTF_8 );
    int    postDataLength = postData.length;

        url = new URL(baseURL);
        def body_req = []

        def authString = "${username}:${password}".bytes.encodeBase64().toString()

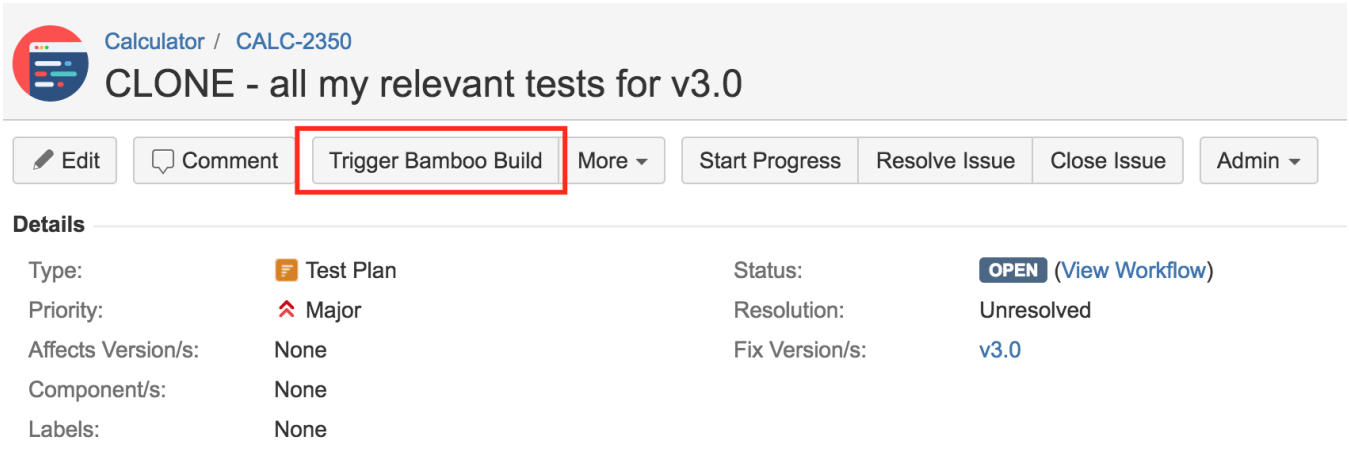
        URLConnection connection = url.openConnection();
        connection.requestMethod = "POST"
        connection.doOutput = true
        connection.addRequestProperty("Authorization", "Basic ${authString}")
    connection.setRequestProperty( "Content-Type", "application/x-www-form-urlencoded");
    connection.setRequestProperty( "charset", "utf-8");
    connection.setRequestProperty( "Content-Length", Integer.toString( postDataLength ));
    connection.setUseCaches( false );

    DataOutputStream wr = new DataOutputStream(connection.getOutputStream());
    wr.write( postData );

        connection.connect();
        //connection.getContent();
        log.debug(connection.getResponseCode())
        log.debug(connection.getResponseMessage())


        if (connection.getResponseCode() == 200) {
                Response.ok(JsonOutput.toJson(flag)).build()
         } else {
                //Response.status(Response.Status.NOT_FOUND).entity("Problem scheduling job!").build();
         }

}
```



Example

ScripRunner configuration

Custom web item

Creates a web item (a button or link) at the location you specify

Note: Trigger Jenkins build with Test List

An optional note, used only for your reference.

What section should this go in: operations-top-level

The section to place the item

Key: trigger-jenkins-build-with-params-button

The key of the module

Menu text: Trigger Jenkins Build with these Tests

Text of menu item

Weight: 2

Placement of the item within its section

Condition: SCRIPT | FILE

```
1 issue.issueType.name == 'Test Plan'
```

Expand examples

Under what circumstances should this link be displayed. Must be either a plain script or an implementation of com.atlassian.plugin.web.Condition

Do what: Run code and show a flag

What do you want to do when the item is clicked

Link: /rest/scriptrunner/latest/custom/triggerJenkinsBuildWithTestList?issueId=${issu

The relative URL to direct to. If external, include the scheme, eg http://google.com

Custom endpoint

Define a REST endpoint in a script

Note: Trigger Bamboo build

An optional note, used only for your reference.

Script file: Start typing to search for script files...

Path to the script file accessible on the server

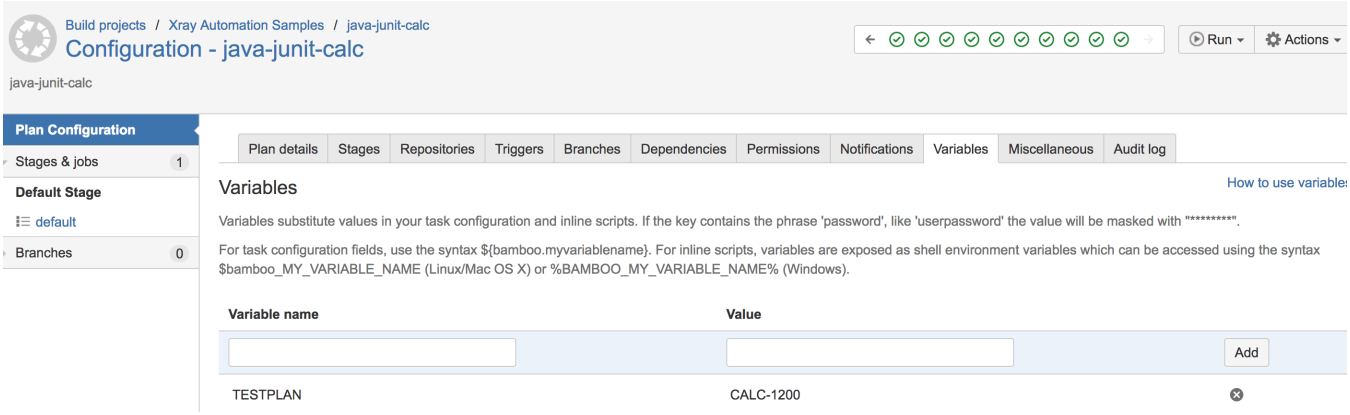
Inline script

```
1  import com.onresolve.scriptrunner.runner.rest.commo
2  import groovy.json.JsonOutput
3  import groovy.transform.BaseScript
4  import groovy.json.JsonSlurper;
5  import groovy.json.StreamingJsonBuilder;
6  import javax.ws.rs.core.MultivaluedMap
7  import javax.ws.rs.core.Response
8  import java.nio.charset.StandardCharsets
9
10 @BaseScript CustomEndpointDelegate delegate
11
12 triggerBambooBuild(httpMethod: "GET") { Multivalued
13
14
```

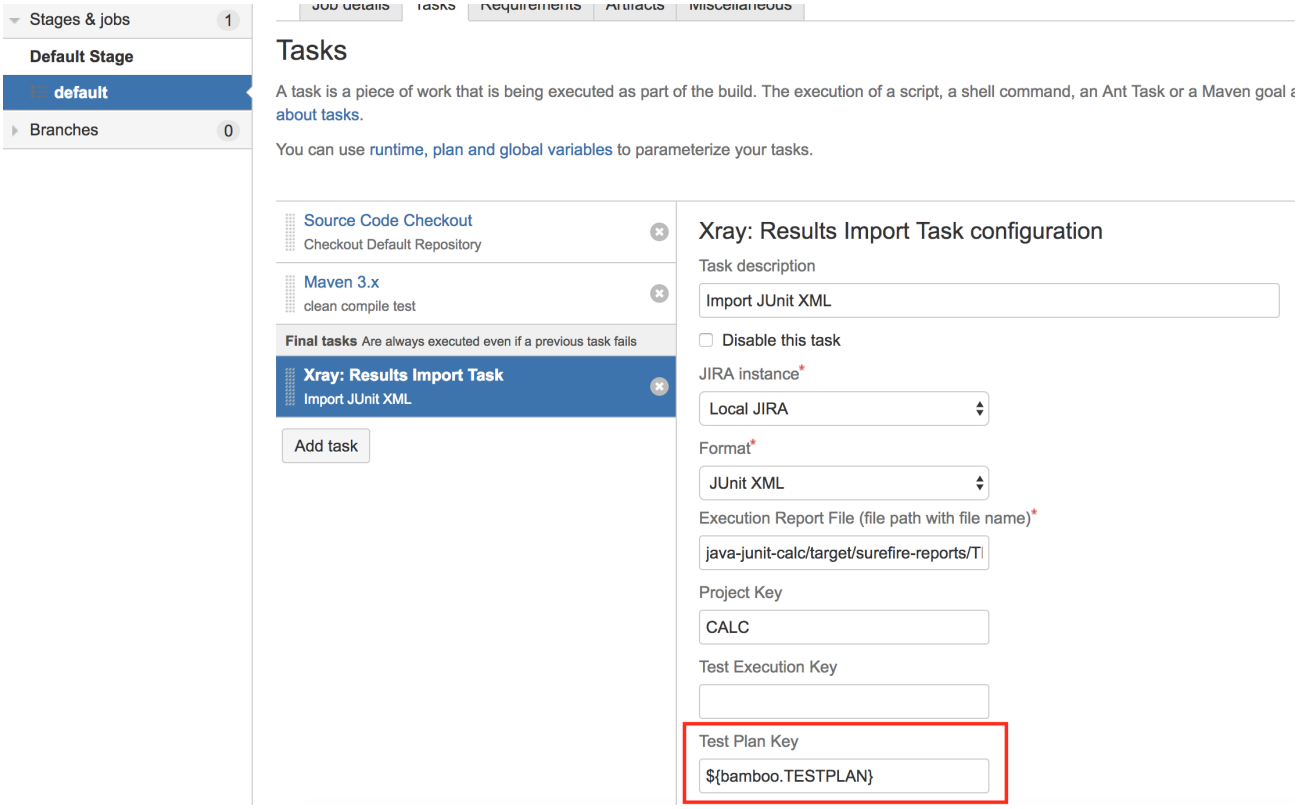
Enter your script here

Bamboo configuration

The project itself is a normal one; the only thing relevant to mention is that this project is a parameterized one, so it receives a TESTPLAN variable, that in our case will be coming from Jira.



The final task submits the results linking the Test Execution to the Test Plan passed as an argument.



Trigger a Bamboo plan/stage build from a Test Plan, for the Tests contained in the Test Plan

This scenario is somehow similar to the previous one, except that the list of Tests that will be run in the CI side will be based on the Tests contained in the Test Plan.

The results will be submitted back to Xray, if the project is configured to do so in Bamboo.

In order to add this option in Jira's UI, we'll need to add a custom "web item" that provides an action that will interact with a custom ScriptRunner endpoint, which will be the one doing the HTTP request to the Bamboo server, passing the Test Plan issue key. In the ScriptRunner endpoint script, we'll obtain the list of Generic Tests (we're assuming that they will come from Junit, so they have a certain syntax in the Generic Test Definition field.

In order to submit the request to Bamboo, we need the credentials of some Bamboo user.

trigger_bamboo_build_restapi_endpoint_with_testlist.groovy

```
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.link.IssueLinkManager
import com.atlassian.jira.issue.link.IssueLinkType
import com.atlassian.jira.issue.link.IssueLinkTypeManager
import com.atlassian.jira.ComponentManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.jql.builder.JqlQueryBuilder
import com.atlassian.jira.user.util.UserUtil
import com.atlassian.jira.user.util.UserManager;
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.search.SearchService;
import com.atlassian.jira.issue.search.SearchProvider
import com.atlassian.jira.issue.search.SearchResults
import com.atlassian.jira.web.bean.PagerFilter;
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.user.UserPropertyManager
import com.atlassian.jira.propertyset.JiraPropertySetFactory;
import com.google.common.collect.ImmutableMap;
import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.module.propertyset.PropertySetManager;
import com.atlassian.jira.util.BuildUtils
import com.atlassian.jira.util.BuildUtilsInfo
import com.atlassian.jira.util.BuildUtilsInfoImpl
import com.atlassian.plugin.PluginAccessor
import com.atlassian.plugin.PluginManager
import com.atlassian.jira.bc.license.JiraLicenseService
import com.atlassian.jira.bc.license.JiraLicenseServiceImpl
import org.apache.log4j.Level
import org.apache.log4j.Logger
import com.atlassian.jira.issue.IssueManager
import com.opensymphony.workflow.InvalidInputException



import com.onresolve.scriptrunner.runner.rest.common.CustomEndpointDelegate
import groovy.json.JsonOutput
import groovy.transform.BaseScript
import groovy.json.JsonSlurper;
import groovy.json.StreamingJsonBuilder;
import javax.ws.rs.core.MultivaluedMap
import javax.ws.rs.core.Response
import java.nio.charset.StandardCharsets

@BaseScript CustomEndpointDelegate delegate

issueManager = ComponentAccessor.getIssueManager()
searchService =  ComponentAccessor.getComponent(SearchService.class);
serviceAccount = ComponentAccessor.getJiraAuthenticationContext().getLoggedInUser()
customFieldManager = ComponentAccessor.getCustomFieldManager()


Object getIssues(jqlQuery){
    // A list of GenericValues representing issues
    List<Issue> searchResults = null;

    SearchService.ParseResult parseResult =  searchService.parseQuery(serviceAccount, jqlQuery);

    if (parseResult.isValid()) {
        // throws SearchException
        SearchResults results = searchService.search(serviceAccount, parseResult.getQuery(), PagerFilter.
getUnlimitedFilter());
        searchResults = results.getIssues();
        return searchResults;
    }

     return []
}
```

```
Object getFieldValue(issue,customField) {
    def cField = customFieldManager.getCustomFieldObjectByName(customField)
    def cFieldValue = issue.getCustomFieldValue(cField)
    return cFieldValue
}

String replaceLast(String string, String substring, String replacement)
{
  int index = string.lastIndexOf(substring);
  if (index == -1)
    return string;
  return string.substring(0, index) + replacement + string.substring(index+substring.length());
}

triggerBambooBuildWithTestList(httpMethod: "GET") { MultivaluedMap queryParams ->

        def issueId = queryParams.getFirst("issueId") as String // use the issueId to retrieve this issue

        def flag = [
        type : 'success',
        title: "Build scheduled",
        close: 'auto',
        body : "A new build has been scheduled related with "+issueId
        ]


        jql = "issue in testPlanTests('${issueId}') and \"Test Type\" = Generic"
        issues = getIssues(jql)
    // // we're assuming that we have Junit based Tests.. so we need to do some conversion beforehand, so maven
can process the list of tests to be run
        def testlist = issues.collect { getFieldValue(it,"Generic Test Definition")}
    def testlist2 = testlist.collect { replaceLast(it,".","%23") }

        URL url;
        // curl --user admin:admin -X POST -d "default&ExecuteAllStages=true&bamboo.TESTLIST=com.xpand.java.
CalcTest#CanAddNumbers" http://yourbambooserver/rest/api/latest/queue/XRAY-JUNITCALCPARAMS

        def projectKey = "XRAY"                                                                          // could
come from a CF in the Test Plan
    def planKey = "JUNITCALCPARAMS"                                                                  // could come from a
CF in the Test Plan
    def stage = "default"                                    // could be hardcoded or come from a CF in the Test
Plan
        def bambooHostPort = "192.168.56.102:8085"                                  // could be defined elsewhere
        def username = "admin"                                                                           //
probably, would need to be stored elsewhere
        def password = "admin"                                                                           //
probably, would need to be stored elsewhere
        def baseURL = "http://${bambooHostPort}/rest/api/latest/queue/${projectKey}-${planKey}"
    String urlParameters  = "${stage}&ExecuteAllStages=true&bamboo.TESTPLAN=${issueId}&bamboo.
TESTLIST=${testlist2.join(',')}";
    byte[] postData       = urlParameters.getBytes( StandardCharsets.UTF_8 );
    int    postDataLength = postData.length;

        url = new URL(baseURL);
        def body_req = []

        def authString = "${username}:${password}".bytes.encodeBase64().toString()

        URLConnection connection = url.openConnection();
        connection.requestMethod = "POST"
        connection.doOutput = true
        connection.addRequestProperty("Authorization", "Basic ${authString}")
    connection.setRequestProperty( "Content-Type", "application/x-www-form-urlencoded");
    connection.setRequestProperty( "charset", "utf-8");
    connection.setRequestProperty( "Content-Length", Integer.toString( postDataLength ));
    connection.setUseCaches( false );

    DataOutputStream wr = new DataOutputStream(connection.getOutputStream());
    wr.write( postData );
```

```
        connection.connect();
        //connection.getContent();
        log.debug(connection.getResponseCode())
        log.debug(connection.getResponseMessage())


        if (connection.getResponseCode() == 200) {
                Response.ok(JsonOutput.toJson(flag)).build()
         } else {
                //Response.status(Response.Status.NOT_FOUND).entity("Problem scheduling job!").build();
         }

}
```

Example

ScripRunner configuration

Custom web item

Creates a web item (a button or link) at the location you specify

Note: Trigger Bamboo build with Test List

An optional note, used only for your reference.

What section should this go in: operations-top-level

The section to place the item

Key: trigger-bamboo-build-with-params-button

The key of the module

Menu text: Trigger Bamboo Build with these Tests

Text of menu item

Weight: 2

Placement of the item within its section

Condition: SCRIPT | FILE

```
1 issue.issueType.name == 'Test Plan'
```

Expand examples

Under what circumstances should this link be displayed. Must be either a plain script or an implementation of com.atlassian.plugin.web.Condition

Do what: Run code and show a flag

What do you want to do when the item is clicked

Link: /rest/scriptrunner/latest/custom/triggerBambooBuildWithTestList?issueId=${issu

The relative URL to direct to. If external, include the scheme, eg http://google.com

Custom endpoint

Define a REST endpoint in a script

Note: Trigger Bamboo build with Test List

An optional note, used only for your reference.

Script file: Start typing to search for script files...

Path to the script file accessible on the server

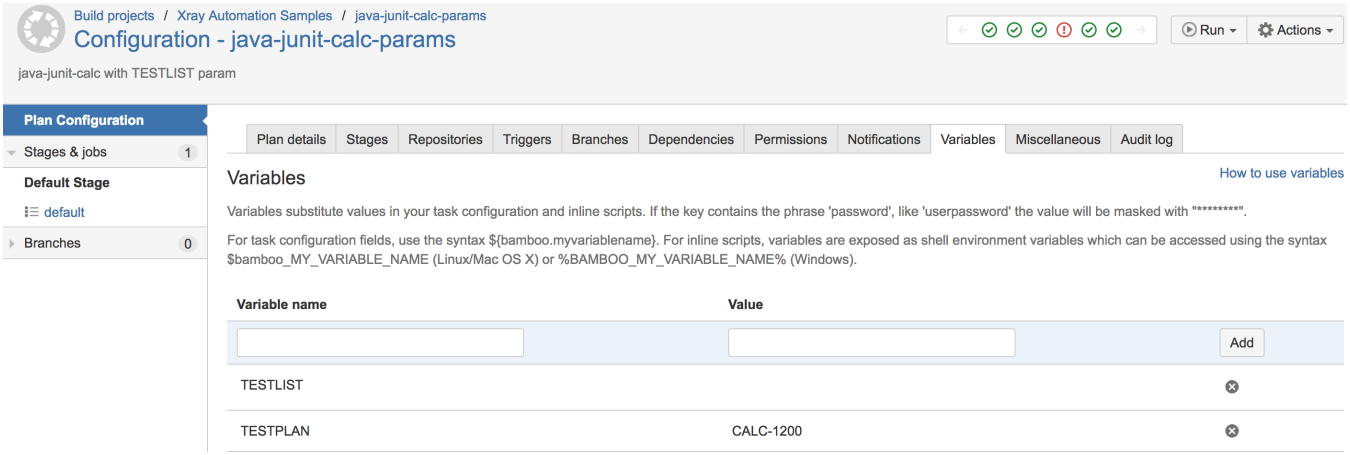
Inline script

```
1  import com.atlassian.jira.issue.Issue
2  import com.atlassian.jira.issue.link.IssueLinkMana
3  import com.atlassian.jira.issue.link.IssueLinkType
4  import com.atlassian.jira.issue.link.IssueLinkTypeI
5  import com.atlassian.jira.ComponentManager
6  import com.atlassian.jira.component.ComponentAcces
7  import com.atlassian.jira.jql.builder.JqlQueryBuil
8  import com.atlassian.jira.user.util.UserUtil
9  import com.atlassian.jira.user.util.UserManager;
10 import com.atlassian.jira.bc.issue.IssueService
11 import com.atlassian.jira.bc.issue.search.SearchSe
12 import com.atlassian.jira.issue.search.SearchProvi
13 import com.atlassian.jira.issue.search.SearchResul
14
```

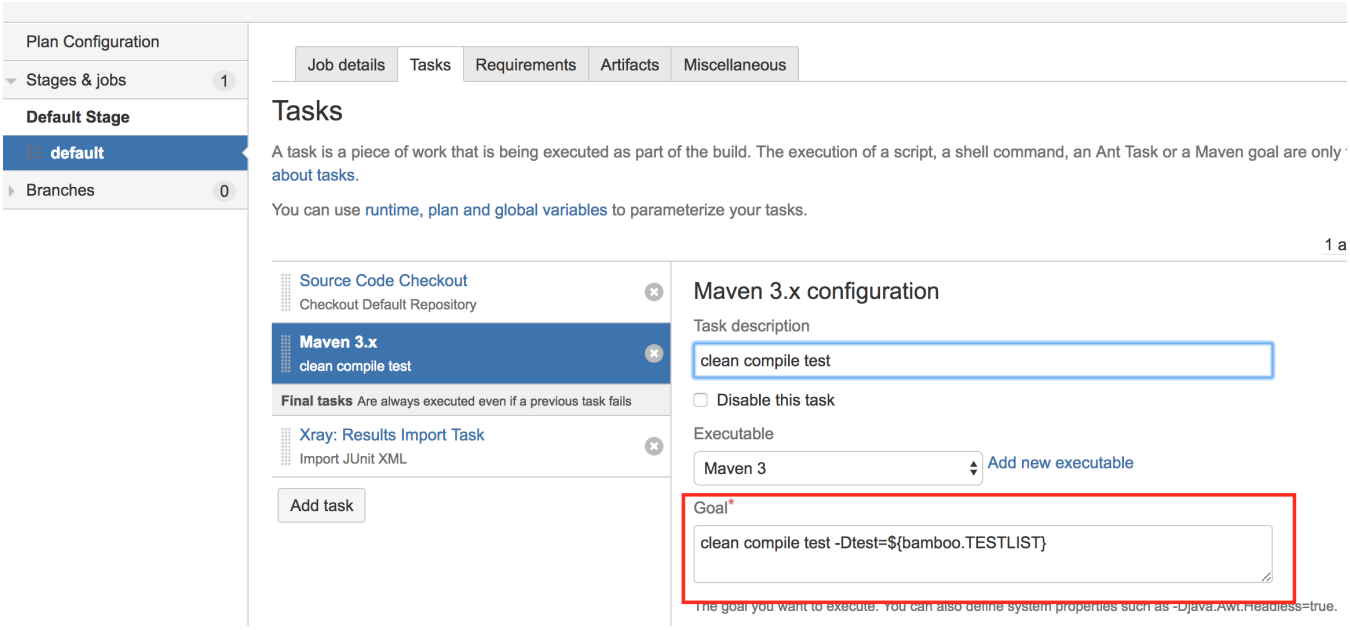
Enter your script here

Bamboo configuration

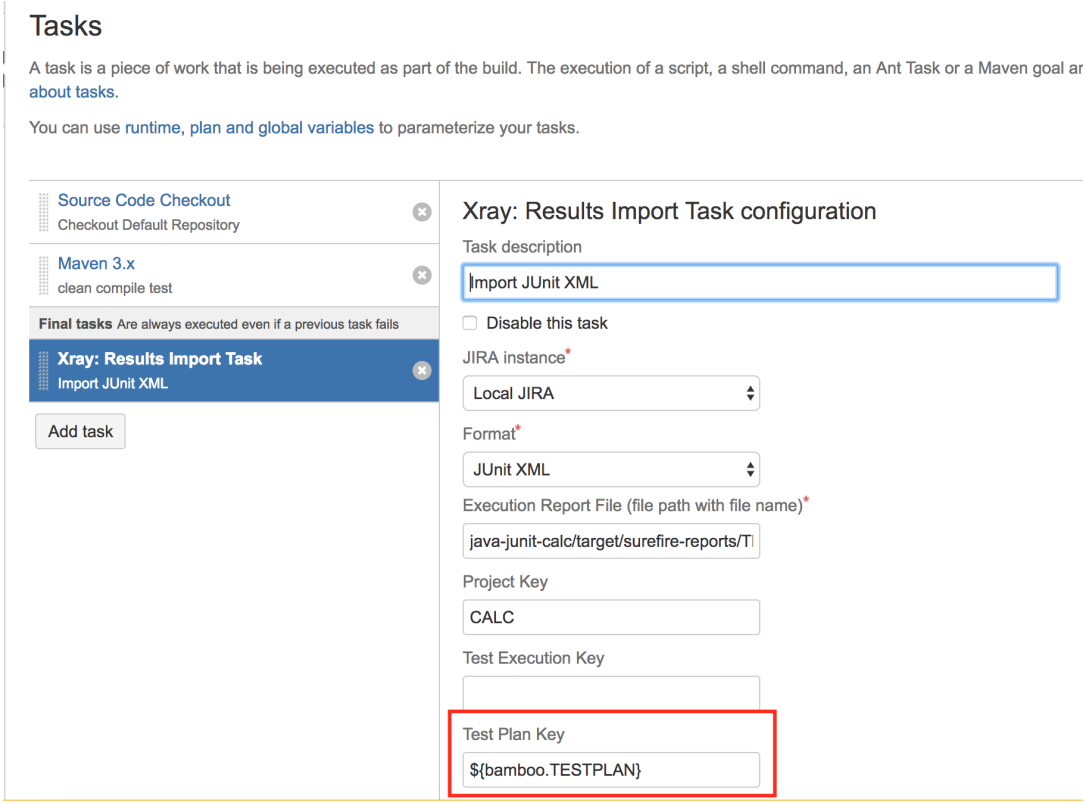
The project itself is a normal one; the only thing relevant to mention is that this project is a parameterized one, so it receives TESTPLAN and TESTLIST variables, that in our case will be coming from Jira.



Maven is configured in order to run just the tests identified in the TESTLIST variable, using the "-Dtest" JVM option.



The final task submits the results linking the Test Execution to the Test Plan passed as argument.



Extending REST API for interacting with requirement projects

In this example, we'll be creating some endpoints for obtaining the requirement projects and also for enabling or disabling requirement coverage for a certain project.

This makes use of ScriptRunner's custom REST API capabilities.

xray_custom_rest_api.groovy

```
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.link.IssueLinkManager
import com.atlassian.jira.issue.link.IssueLinkType
import com.atlassian.jira.issue.link.IssueLinkTypeManager
import com.atlassian.jira.ComponentManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.jql.builder.JqlQueryBuilder
import com.atlassian.jira.user.util.UserUtil
import com.atlassian.jira.user.util.UserManager;
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.search.SearchService;
import com.atlassian.jira.issue.search.SearchProvider
import com.atlassian.jira.issue.search.SearchResults
import com.atlassian.jira.web.bean.PagerFilter;
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.user.UserPropertyManager
import com.atlassian.jira.propertyset.JiraPropertySetFactory;
import com.google.common.collect.ImmutableMap;
import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.module.propertyset.PropertySetManager;
import com.onresolve.scriptrunner.runner.rest.common.CustomEndpointDelegate
import groovy.json.JsonBuilder
import groovy.json.JsonSlurper;
import groovy.transform.BaseScript
```

```
import javax.servlet.http.HttpServletRequest
import javax.ws.rs.core.MultivaluedMap
import javax.ws.rs.core.Response

@BaseScript CustomEndpointDelegate delegate


ENTITY_NAME = "com.xpandit.raven";
ENTITY_ID = 12345678987654321L;
REQUIREMENT_PROJECTS_SETTING = "requirement-coverage.projects";

projectManager = ComponentAccessor.getProjectManager()

Object obtainRequirementProjectsIds() {
    JiraPropertySetFactory jiraPropertySetFactory = ComponentAccessor.getComponent(JiraPropertySetFactory.
class);
    def setting = jiraPropertySetFactory.buildCachingPropertySet(ENTITY_NAME, ENTITY_ID, true);
    def requirementProjects = Eval.me(setting.getText(REQUIREMENT_PROJECTS_SETTING))
}


Object obtainRequirementProjects() {
    def requirementProjects = obtainRequirementProjectsIds()
    log.debug("requirementProjects: "+requirementProjects)
    def availableProjects = projectManager.getProjectObjects()
    availableProjects.findAll { it.id.toInteger() in requirementProjects}
}

boolean enableRequirementCoverageForProject(project){
    JiraPropertySetFactory jiraPropertySetFactory = ComponentAccessor.getComponent(JiraPropertySetFactory.
class);
    def setting = jiraPropertySetFactory.buildCachingPropertySet(ENTITY_NAME, ENTITY_ID, true);
    projectList = obtainRequirementProjectsIds()
    if (!projectList.contains(project.id.toInteger())){
        setting.setText(REQUIREMENT_PROJECTS_SETTING,(projectList << project.id).toString())
    }
}

boolean disableRequirementCoverageForProject(project){
    JiraPropertySetFactory jiraPropertySetFactory = ComponentAccessor.getComponent(JiraPropertySetFactory.
class);
    def setting = jiraPropertySetFactory.buildCachingPropertySet(ENTITY_NAME, ENTITY_ID, true);
    projectList = obtainRequirementProjectsIds()
    if (projectList.contains(project.id.toInteger())){
        projectList.removeAll{it == project.id.toInteger()}
        setting.setText(REQUIREMENT_PROJECTS_SETTING,projectList.toString())
    }
}



boolean requirementCoverageEnabledForProject(project){
    JiraPropertySetFactory jiraPropertySetFactory = ComponentAccessor.getComponent(JiraPropertySetFactory.
class);
    def setting = jiraPropertySetFactory.buildCachingPropertySet(ENTITY_NAME, ENTITY_ID, true);
    def requirementProjects = Eval.me(setting.getText(REQUIREMENT_PROJECTS_SETTING))
    (project.id.toInteger() in requirementProjects)
}




// curl -u admin:admin "http://yourjiraserver/rest/scriptrunner/latest/custom/getRequirementProjects"
requirementProjects(
    httpMethod: "GET", groups: ["jira-administrators"]
) { MultivaluedMap queryParams, String body ->
    return Response.ok(new JsonBuilder(obtainRequirementProjects().collect{ [id: it.id, key: it.key, name: it.
name] } ).toString() ).build()
}
```

```
// curl -u admin:admin -X DELETE "http://yourjiraserver/rest/scriptrunner/latest/custom/requirementProjects
/CALC"
requirementProjects(
    httpMethod: "DELETE", groups: ["jira-administrators"]
) {MultivaluedMap queryParams, String body, HttpServletRequest request ->
    try{
        def extraPath = getAdditionalPath(request)
        projectKey = extraPath.split("/")[1]
        log.debug("projectKey: "+projectKey)
        project = projectManager.getProjectObjByKey(projectKey)
        disableRequirementCoverageForProject(project)
    } catch (e) {
        return Response.serverError().entity([error: e.message]).build()
    }
    return Response.ok().build()
}


//curl -u admin:admin -X POST -d "@data.json" -H "Content-Type: application/json" "http://yourjiraserver/rest
/scriptrunner/latest/custom/requirementProjects"
requirementProjects(
    httpMethod: "POST", groups: ["jira-administrators"]
) {MultivaluedMap queryParams, String body ->
    try{
        def jsonSlurper = new JsonSlurper()
        def object = jsonSlurper.parseText(body)
        log.debug("projectKey: "+object.key)
        def project
        if (object.key) {
            project = projectManager.getProjectObjByKey(object.key)
        } else if (project.id ){
            project = projectManager.getProjectObjById(object.id)
        }
        enableRequirementCoverageForProject(project)
    } catch (e) {
        return Response.serverError().entity([error: e.message]).build()
    }
    return Response.ok().build()
}
```

Example of requests

Obtaining all projects with requirement coverage enabled

```
curl -u admin:admin "http://yourjiraserver/rest/scriptrunner/latest/custom/requirementProjects"
```

Response

```
[
   {
      "id":10300,
      "key": "CALC",
      "name":"Calculator"
   },
   {
      "id":10501,
      "key": "DEMO",
      "name":"Demonstration"
   }
]
```

Enabling requirement coverage for a project

```
curl -u admin:admin -X POST -d "@data.json" -H "Content-Type: application/json" "http://yourjiraserver/rest
/scriptrunner/latest/custom/requirementProjects"
```

data.json

```
{
 "key": "CALC"
}
```

Disabling requirement coverage for a project

```
curl -u admin:admin -X DELETE "http://yourjiraserver/rest/scriptrunner/latest/custom/requirementProjects/CALC"
```

Synchronize Tests from "related" Test Sets to a Test Execution or to a Test Plan

Test Plans and (Sub) Test Executions contain a list of Tests. Although you can add Tests using Test Sets, you're not actually adding the Test Set itself; you're adding the Tests that belong to that Test Set(s) at that given moment. Thus, there is no relation whatsoever between Test Plans<=>Test Sets or between Test Executions<=>Test Sets.

However, and taking the Test Plan as an example, you may find handy to have some sort of "dynamic Test Plan" that will contain the Tests of the Test Sets that would be related to that Test Plan.

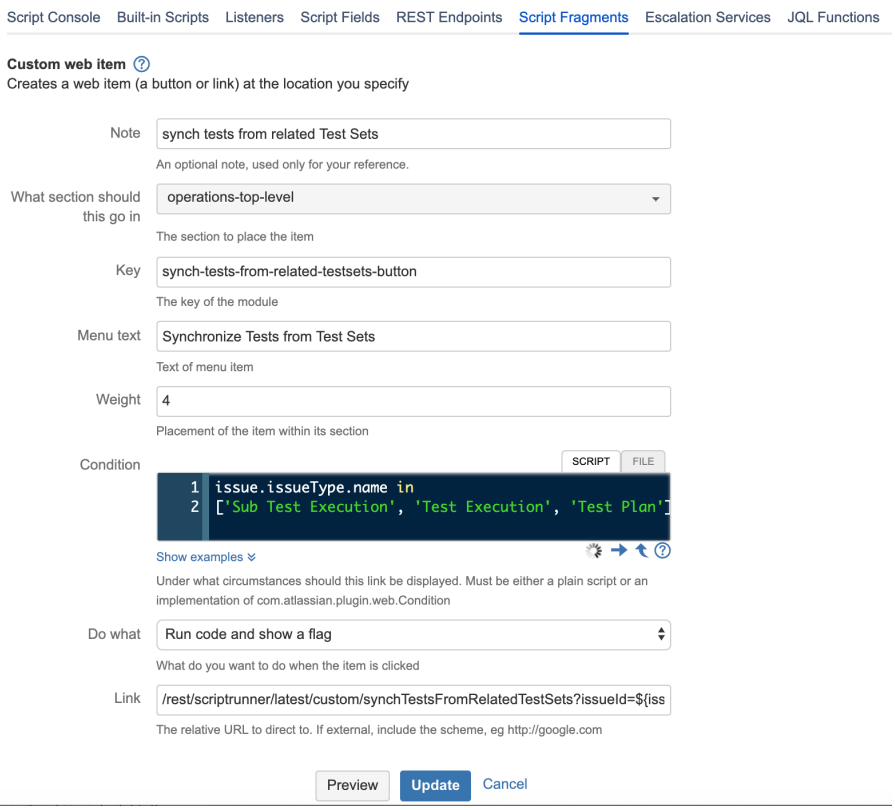
Since there is no relation between Test Plans<=>Test Sets, you have to define a convention for that. One option would be to use issue links (e.g. "relates to" or "includes"); in this case, you would need to manually create these links between Test Sets and Test Plans (or Test Executions).

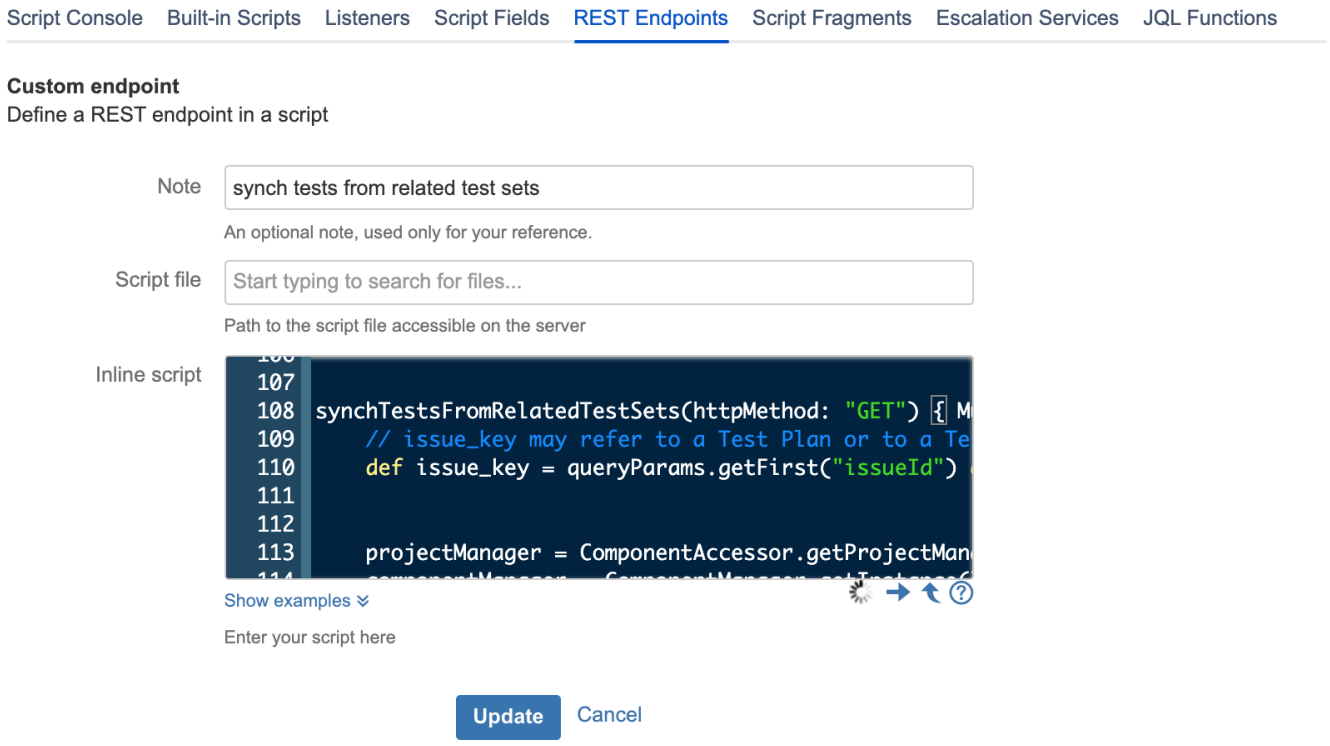
After this, you can create a custom script that will obtain the Tests from the related (e.g. the linked) Test Sets using JQL, followed by REST API requests to specific endpoints to add these Tests to the destination entity (e.g. Test Plan).

We start by adding a button in Jira's UI using a custom "web item", providing an action that will interact with a custom ScriptRunner endpoint containing the following logic:

- receive the source issue key that triggered the synchronization request (e.g. Test Plan, Test Execution, Sub Test Execution)
- obtain the linked Test Sets, using JQL
    - obtain the Tests on each Test Set, using JQL
    - submit a REST API request to Xray specific endpoints to add the Test to the entity (e.g Test Plan)

ScripRunner configuration





synchTestsFromRelatedTestSets_restapi_endpoint.groovy

```
import com.onresolve.scriptrunner.runner.rest.common.CustomEndpointDelegate
import groovy.json.JsonOutput
import groovy.transform.BaseScript
import groovy.json.JsonSlurper;
import groovy.json.StreamingJsonBuilder;
import javax.ws.rs.core.MultivaluedMap
import javax.ws.rs.core.Response
import java.nio.charset.StandardCharsets
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.link.IssueLinkManager
import com.atlassian.jira.issue.link.IssueLinkType
import com.atlassian.jira.issue.link.IssueLinkTypeManager
import com.atlassian.jira.ComponentManager
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.jql.builder.JqlQueryBuilder
import com.atlassian.jira.user.util.UserUtil
import com.atlassian.jira.user.util.UserManager;
import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.bc.issue.search.SearchService;
import com.atlassian.jira.issue.search.SearchProvider
import com.atlassian.jira.issue.search.SearchResults
import com.atlassian.jira.web.bean.PagerFilter;
import com.atlassian.jira.issue.MutableIssue
import com.atlassian.jira.user.UserPropertyManager
import com.atlassian.jira.propertyset.JiraPropertySetFactory;
import com.google.common.collect.ImmutableMap;
import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.module.propertyset.PropertySetManager;
import com.atlassian.jira.util.BuildUtils
import com.atlassian.jira.util.BuildUtilsInfo
import com.atlassian.jira.util.BuildUtilsInfoImpl
import com.atlassian.plugin.PluginAccessor
import com.atlassian.plugin.PluginManager
import com.atlassian.jira.bc.license.JiraLicenseService
import com.atlassian.jira.bc.license.JiraLicenseServiceImpl
import org.apache.log4j.Level
import org.apache.log4j.Logger
import com.atlassian.jira.issue.IssueManager
import groovy.json.StreamingJsonBuilder



@BaseScript CustomEndpointDelegate delegate


boolean associateTestsToXrayIssue(endpoint, issueKey,listOfTestKeys){
    def jiraBaseUrl = com.atlassian.jira.component.ComponentAccessor.getApplicationProperties().getString("jira.
baseurl")
    def endpointUrl = "${jiraBaseUrl}/rest/raven/1.0/api/${endpoint}/${issueKey}/test"

    log.debug("issueKey: "+issueKey)
    log.debug("listOfTestKeys: "+listOfTestKeys)
    log.debug("jirabaseurl: "+jiraBaseUrl)
    log.debug("endpoint: "+endpointUrl)
    url = new URL(endpointUrl);
    def body_req = [ "add": listOfTestKeys ]

    // you should use a specific user for this purpose
    username = "admin"
    password = "admin"
    def authString = "${username}:${password}".bytes.encodeBase64().toString()

    URLConnection connection = url.openConnection();
    connection.requestMethod = "POST"
    connection.doOutput = true
    connection.addRequestProperty("Authorization", "Basic ${authString}")
    connection.setRequestProperty("Content-Type", "application/json;charset=UTF-8")
    connection.outputStream.withWriter("UTF-8") { new StreamingJsonBuilder(it, body_req) }
    connection.connect();
    log.debug(connection.getResponseCode())
```

```
    log.debug(connection.getResponseMessage())

    if (connection.getResponseCode() == 200) {
        // OK
        return true;
    } else {
        // error
        return false;
    }
}

boolean associateTestsToTestPlan(testPlanKey, listOfTestKeys){
 return associateTestsToXrayIssue("testplan", testPlanKey, listOfTestKeys)
}

boolean associateTestsToTestExecution(testExecutionKey, listOfTestKeys){
 return associateTestsToXrayIssue("testexec", testExecutionKey, listOfTestKeys)
}



Object getIssues(jqlQuery){
    // A list of GenericValues representing issues
    List<Issue> searchResults = null;
    SearchService.ParseResult parseResult =  searchService.parseQuery(serviceAccount, jqlQuery);

    if (parseResult.isValid()) {
        // throws SearchException
        SearchResults results = searchService.search(serviceAccount, parseResult.getQuery(), PagerFilter.
getUnlimitedFilter());
        searchResults = results.getIssues();
        return searchResults;
    }

     return []
}


synchTestsFromRelatedTestSets(httpMethod: "GET") { MultivaluedMap queryParams ->
    // issue_key may refer to a Test Plan or to a Test Execution
    def issue_key = queryParams.getFirst("issueId") as String // use the issueId to retrieve this issue


    projectManager = ComponentAccessor.getProjectManager()
    componentManager = ComponentManager.getInstance();
    searchService =  ComponentAccessor.getComponent(SearchService.class);
    issueManager = ComponentAccessor.getIssueManager()
    customFieldManager = ComponentAccessor.getCustomFieldManager()
    userUtil = ComponentAccessor.getUserUtil();
    serviceAccount = ComponentAccessor.getJiraAuthenticationContext().getLoggedInUser()
    pluginAccessor = componentManager.getPluginAccessor();

    MutableIssue issue = issueManager.getIssueObject(issue_key)

    Logger.getLogger("com.onresolve").setLevel(Level.DEBUG)

    // assume that Test Plan/Execution is linked to Test Sets using the issue link "relates to"; customize if
needed
    jql = "issue in linkedIssues('${issue_key}', 'relates to')"
    testset_issues = getIssues(jql)
    def had_errors = false
    def success = false

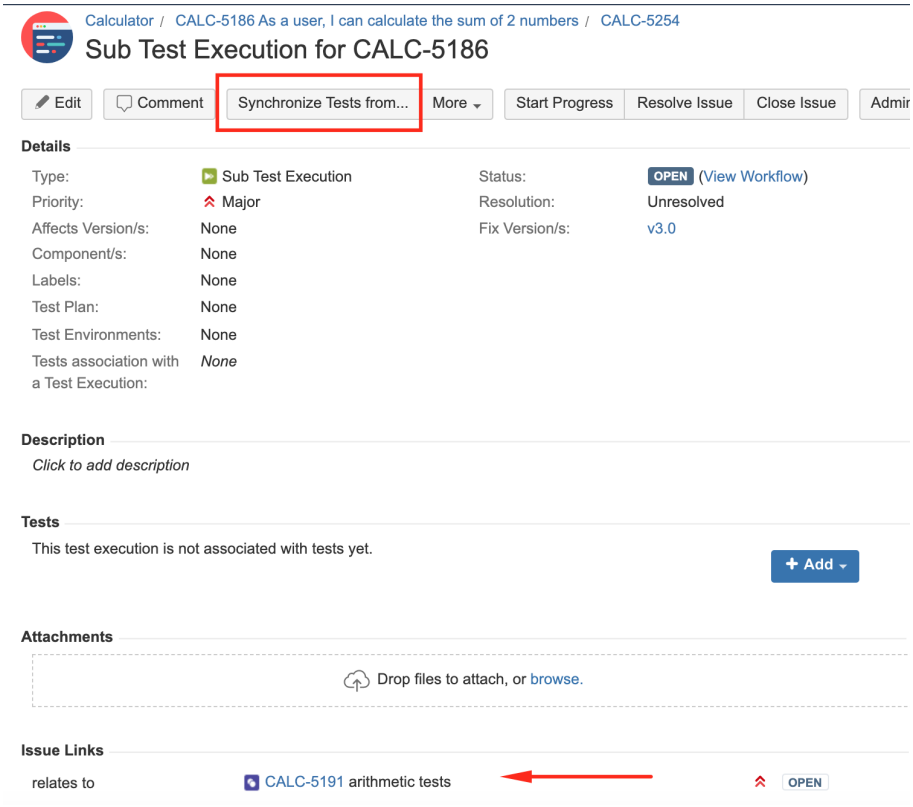
    testset_issues.each {
        // process only "Test Set" issues
        if (it.issueType.name == "Test Set") {
            jql = "issue in testSetTests('${it.key}')"
            issues = getIssues(jql)
            test_keys = issues.collect{ it.key }
            //log.debug(test_keys)
            if (issue.issueType.name == "Test Plan") {
```

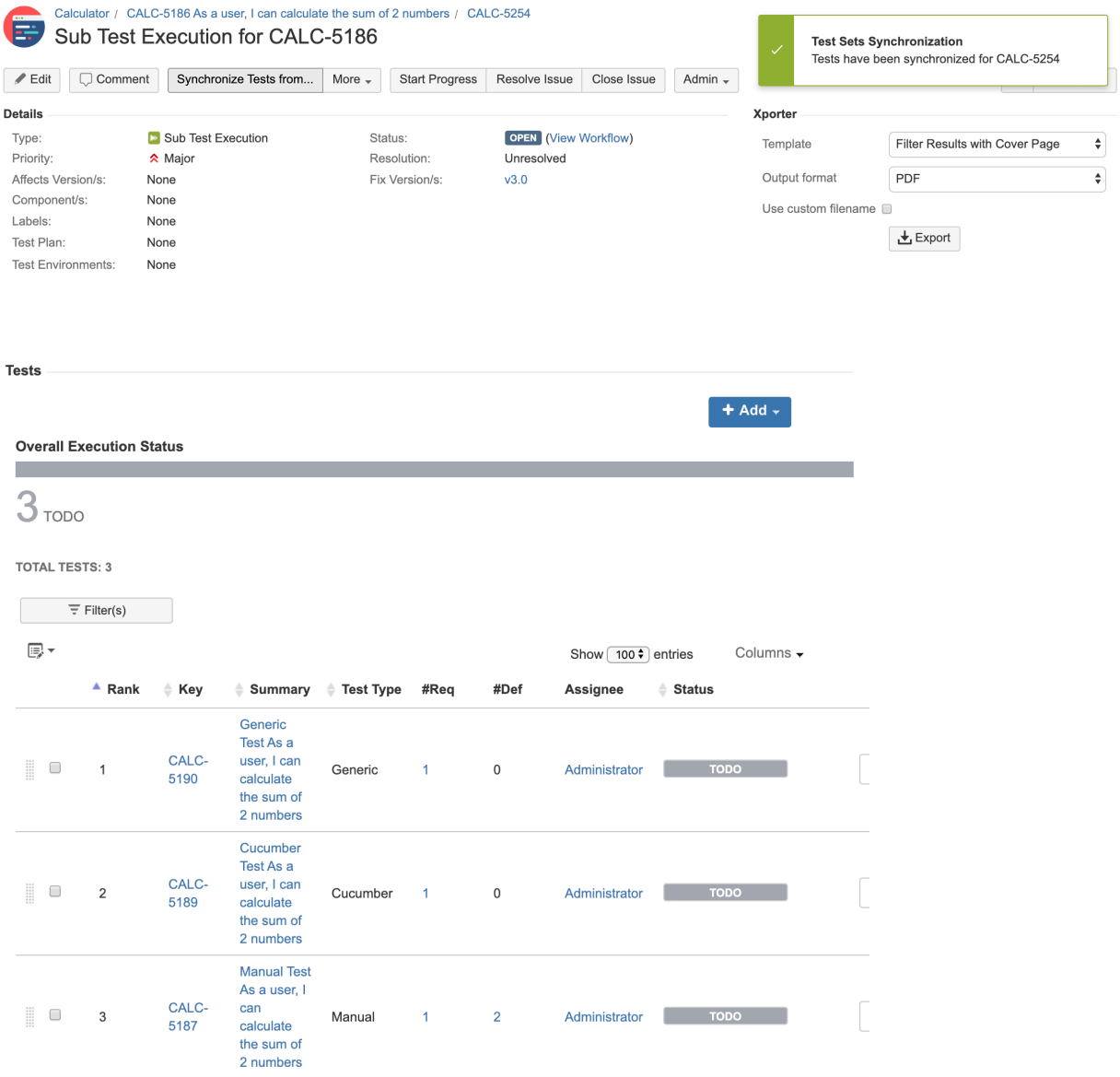
```
                success = associateTestsToTestPlan(issue_key, test_keys)
            } else if ((issue.issueType.name == "Sub Test Execution") || (issue.issueType.name == "Test
Execution")) {
                    success = associateTestsToTestExecution(issue_key, test_keys)
            }
            if (!success) {
                had_errors = true
            }
        }
    }

    def flag = []
    if (success) {
            flag = [
            type : 'success',
            title: "Test Sets Synchronization",
            close: 'auto',
            body : "Tests have been synchronized for " + issue_key
            ]
        Response.ok(JsonOutput.toJson(flag)).build()
    } else {
            flag = [
            type : 'success',
            title: "Test Sets Synchronization",
            close: 'auto',
            body : "Tests have been synchronized for " + issue_key
            ]
        Response.serverError().entity([error: "some Tests were no synchronized"]).build()
    }

}
```

Example





Add a custom link to the Tests top menu

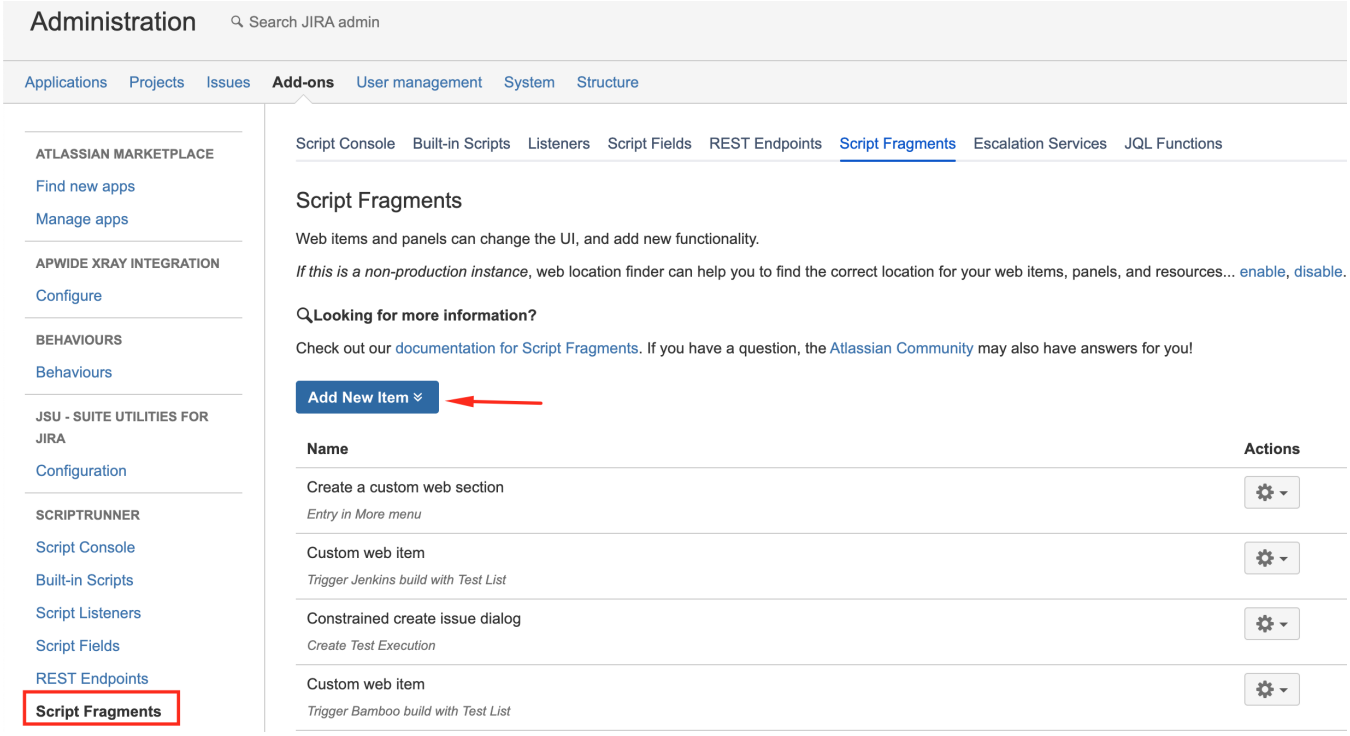
Sometimes it may be useful to add a custom link to the Tests top menu.

As an example, you may have an internal documentation/Confluence space with some valuable information concerning Xray usage in your organization.

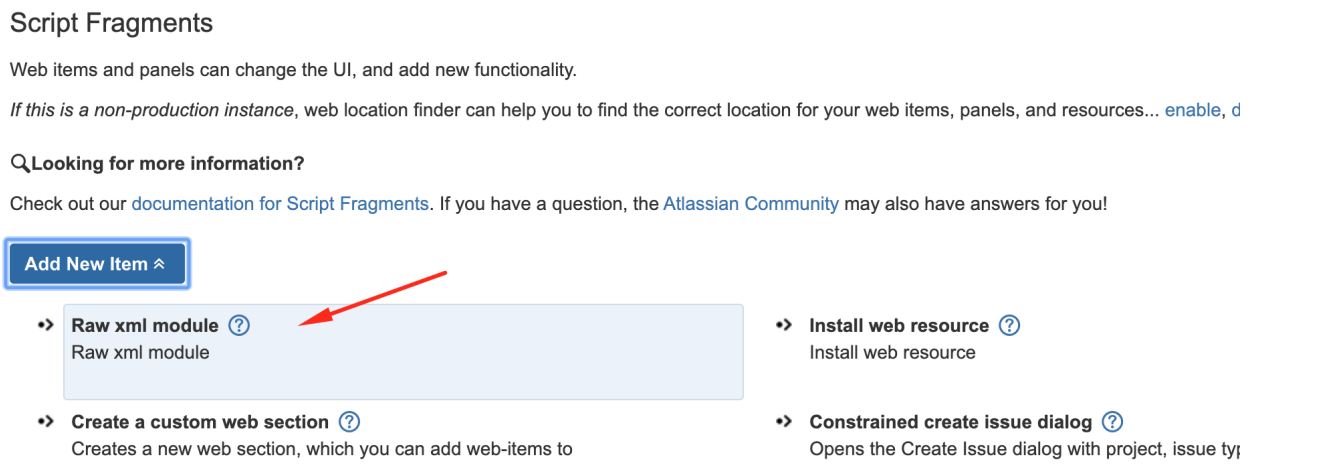
ScripRunner configuration

Adding an entry to the top menu is easy.

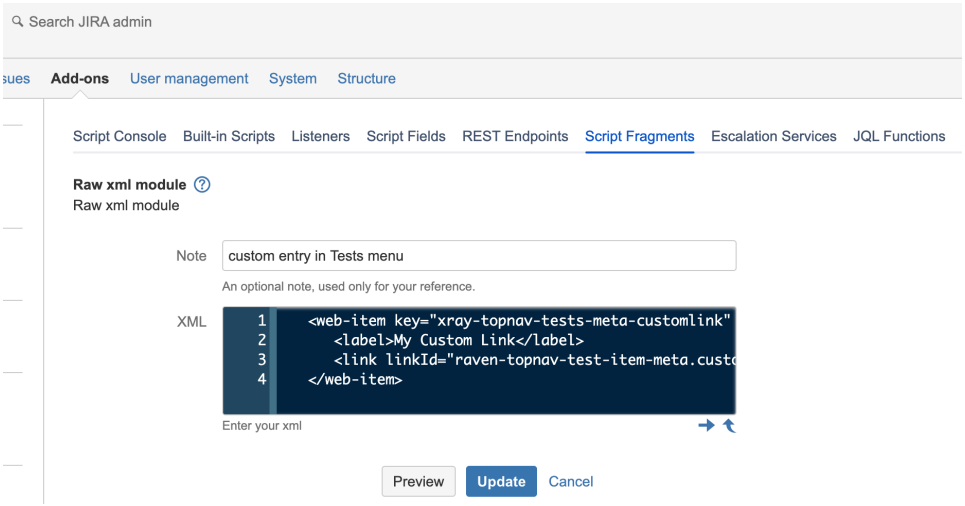
Just go the Add-ons section in your Jira administration and then "Script Fragments".



Add a new "Raw xml module".



And add the configuration for the link; this configuration is exactly the same as if you were going to develop your own app, so it follows Atlassian developer documentation guidelines for the "web-item" element.



Make sure the "web-item" as a "weight" value higher than 120, to append your link to the end of the options available from the dropdown menu.

Example

Let's say that we want to add a link to "https://getxray.app" in the top Test menu, having the name "My Custom Link".

The "raw xml module" configuration would be something similar to the following snippet.

```
   <web-item key="xray-topnav-tests-meta-customlink" name="Custom link" section="raven-menu/xray.topnav.meta.
section" weight="130">
      <label>My Custom Link</label>
      <link linkId="raven-topnav-test-item-meta.customlink">https://getxray.app</link>
   </web-item>
```

