

# Extending and integrating with Xray using 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, specially 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 allowing 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](#)
    - [ScripRunner configuration](#)
    - [Jenkins configuration](#)
- [Trigger a Jenkins project build from a Test Plan, for the Tests contained in the Test Plan](#)
  - [Example](#)
    - [ScripRunner 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](#)

## 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 allowing 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);

```

```

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 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.

**Create a custom web section** ?

Creates a new web section, which you can add web-items to

Note

An optional note, used only for your reference.

Location

What location should this go in

Condition

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

Key

Used for nesting

Weight

Placement of the item within its section

Menu text

Text of menu item

**Constrained create issue dialog** ?

Opens the Create Issue dialog with project, issue type, and a behavior set

Key

Functional note, used only for your reference

What section should this go in

The section to place the item

Key

The key of the menu item

Menu text

Text of menu item

Weight

Placement of the item within its section

Condition

Explicit examples

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

Project

Open issue in what project

Issue type

What for is type?

## 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 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 provide 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:	<b>OPEN</b> ( <a href="#">View Workflow</a> )
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	<a href="#">v3.0</a>
Component/s:	None		
Labels:	None		

### Example

#### ScripRunner configuration


## •> Custom web item

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

Note	<input type="text" value="Trigger Jenkins build"/>
	An optional note, used only for your reference.
What section should this go in	<input type="text" value="operations-top-level"/>
	The section to place the item
Key	<input type="text" value="trigger-jenkins-build-button"/>
	The key of the module
Menu text	<input type="text" value="Trigger Jenkins Build"/>
	Text of menu item
Weight	<input type="text" value="1"/>
	Placement of the item within its section
Condition	<div><div>SCRIPT</div><div>FILE</div><div>1 issue.issueType.name == 'Test Plan'</div></div> <div><a href="#">Expand examples</a>    </div> <div>Under what circumstances should this link be displayed. Must be either a plain script or an implementation of <code>com.atlassian.plugin.web.Condition</code></div>
Do what	<input type="text" value="Run code and show a flag"/>
	What do you want to do when the item is clicked
Link	<input type="text" value="/rest/scriptrunner/latest/custom/triggerJenkinsBuild?issueId=\${issue.key}"/>
	The relative URL to direct to. If external, include the scheme, eg <code>http://google.com</code>


## •> Custom endpoint

Define a REST endpoint in a script

Note	<input type="text" value="jenkins build"/>
	An optional note, used only for your reference.
Script file	<input type="text" value="Start typing to search for script files..."/>
	 Path to the script file accessible on the server
Inline script	<pre>1 import com.onresolve.scriptrunner.runner.rest.common 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") { MultivaluedMap 12</pre>


## Jenkins configuration


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





# Jenkins


Jenkins > admin


 People

 Status

 Builds

 **Configure**

 My Views

 Credentials

Full Name

admin

Description

API Token

User ID


admin

API Token

fa02840152aa2e4da3d8db933ec708d6

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.

### Build Triggers

☒ Trigger builds remotely (e.g., from scripts) 

Authentication Token

IFBDOBhNhaxL4T9ass93HRXun2JF161Z

Use the following URL to trigger build remotely: JENKINS\_URL/job/java-junit-calc/build?token=TOKEN\_NAME or /buildWithParameters?token=TOKEN\_NAME  
Optionally append &cause=Cause+Text to provide text that will be included in the recorded build cause.

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.

Jenkins > java-junit-calc >

General

Source Code Management

Build Triggers

Build Environment

Build

Post-build Actions


Project name

java-junit-calc

Description

creates a new Test Execution with the results from 4 junit tests. The revision field is populated with the build #

[Plain text] [Preview](#)

☒ Discard old builds 

Strategy

Log Rotation

Days to keep builds

if not empty, build records are only kept up to this number of days


Max # of builds to keep


3

if not empty, only up to this number of build records are kept

Advanced...

☐ GitHub project

☒ This project is parameterized 

String Parameter 

Name

TESTPLAN

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

## Post-build Actions

Xray: Results Import Task

X

JIRA Instance

xray-vm

⌵

Format

JUnit XML

⌵

Parameters

Execution Report File (file path with file name)

java-junit-calc/target/surefire-reports/TEST-com.xpand.java.CalcTest.xml

Project Key

CALC

Test Execution Key

Test Plan Key

\$(TESTPLAN)

Test Environments

Revision

\$(BUILD\_NUMBER)

Fix Version

v3.0

## 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 provide 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 omitted 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 = "iFBDOhNhaxL4T9ass93HRXun2JF161Z" // 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 { getField(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

### 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=\${issueId}

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.


 **Jenkins**


Jenkins > admin


 People

 Status

 Builds

 **Configure**

 My Views

 Credentials

Full Name

admin

Description

API Token

User ID

admin

API Token

fa02840152aa2e4da3d8db933ec708d6

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.

**Build Triggers**

☒ Trigger builds remotely (e.g., from scripts)

Authentication Token

iFBDOhNhaxL4T9ass93HRXun2JF161Z

Use the following URL to trigger build remotely: JENKINS\_URL/job/java-junit-calc-triggered/build?token=TOKEN\_NAME or /buildWithParameters?token=TOKEN\_NAME  
Optionally append &cause=Cause+Text to provide text that will be included in the recorded build cause.

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.

Jenkins > java-junit-calc

General

Source Code Management

Build Triggers

Build Environment

Build

Post-build Actions

Project name

java-junit-calc

Description

creates a new Test Execution with the results from 4 junit tests. The revision field is populated with the build #

[Plain text] [Preview](#)

☒ Discard old builds

Strategy

Log Rotation

Days to keep builds

if not empty, build records are only kept up to this number of days

Max # of builds to keep

3

if not empty, only up to this number of build records are kept

Advanced...

☐ GitHub project

☒ This project is parameterized

String Parameter

Name

TESTPLAN

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

Invoke top-level Maven targets

X

?

Goals

clean compile test

▼

POM

java-junit-calc/pom.xml

?

Properties

?

JVM Options

-Dtest=\${TESTLIST}

▼

?

Inject build variables

☐

?

Use private Maven repository

☐

?

Settings file

Use default maven settings

⌵

?

Global Settings file

Use default maven global settings

⌵

?

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

## Post-build Actions

Xray: Results Import Task

X

JIRA Instance

xray-vm

⌵

Format

JUnit XML

⌵

Parameters

Execution Report File (file path with file name)

java-junit-calc/target/surefire-reports/TEST-com.xpand.java.CalcTe

Project Key

CALC

Test Execution Key

Test Plan Key

\${TESTPLAN}

Test Environments

Revision

\${BUILD\_NUMBER}

Fix Version

v3.0

## 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 elsewhere
    def password = "admin" //
probably, would need to be stored elsewhere
    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();
    }
}

```



Calculator / CALC-2350

## CLONE - all my relevant tests for v3.0

Edit

Comment

Trigger Bamboo Build

More ▾

Start Progress

Resolve Issue

Close Issue

Admin ▾

### Details

Type:	Test Plan	Status:	<b>OPEN</b> ( <a href="#">View Workflow</a> )
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	<a href="#">v3.0</a>
Component/s:	None		
Labels:	None		

### Example

#### ScripRunner configuration


## •> Custom web item ?

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

Note	<input type="text" value="Trigger Jenkins build with Test List"/>
	An optional note, used only for your reference.
What section should this go in	<input type="text" value="operations-top-level"/>
	The section to place the item
Key	<input type="text" value="trigger-jenkins-build-with-params-button"/>
	The key of the module
Menu text	<input type="text" value="Trigger Jenkins Build with these Tests"/>
	Text of menu item
Weight	<input type="text" value="2"/>
	Placement of the item within its section
Condition	<div><div>SCRIPT</div><div>FILE</div><div>1 issue.issueType.name == 'Test Plan'</div></div> <div>Expand examples ? ? ? ? ?</div> <div>Under what circumstances should this link be displayed. Must be either a plain script or an implementation of com.atlassian.plugin.web.Condition</div>
Do what	<input type="text" value="Run code and show a flag"/>
	What do you want to do when the item is clicked
Link	<input type="text" value="/rest/scriptrunner/latest/custom/triggerJenkinsBuildWithTestList?issueId=\${issueId}"/>
	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	<input type="text" value="Trigger Bamboo build"/>
	An optional note, used only for your reference.
Script file	<input type="text" value="Start typing to search for script files..."/>
	 Path to the script file accessible on the server
Inline script	<pre>1 import com.onresolve.scriptrunner.runner.rest.common 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</pre> <div>Enter your script here ? ? ? ? ?</div>

## 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.



Build projects / Xray Automation Samples / java-junit-calc

Configuration - java-junit-calc

java-junit-calc

Plan Configuration

Stages & jobs 1

Default Stage

default

Branches 0

Plan details Stages Repositories Triggers Branches Dependencies Permissions Notifications Variables Miscellaneous Audit log

### Variables

Variables substitute values in your task configuration and inline scripts. If the key contains the phrase 'password', like 'userpassword' the value will be masked with \*\*\*\*\*.

For task configuration fields, use the syntax \${bamboo.myvariablename}. For inline scripts, variables are exposed as shell environment variables which can be accessed using the syntax \$bamboo\_MY\_VARIABLE\_NAME (Linux/Mac OS X) or %BAMBOO\_MY\_VARIABLE\_NAME% (Windows).

Variable name	Value
TESTPLAN	CALC-1200

Add

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

Stages & jobs 1

Default Stage

default

Branches 0

### Tasks

A task is a piece of work that is being executed as part of the build. The execution of a script, a shell command, an Ant Task or a Maven goal is about tasks.

You can use [runtime](#), [plan](#) and [global variables](#) to parameterize your tasks.

Source Code Checkout

Checkout Default Repository

Maven 3.x

clean compile test

**Final tasks** Are always executed even if a previous task fails

**Xray: Results Import Task**

Import JUnit XML

Add task

#### Xray: Results Import Task configuration

Task description

Import JUnit XML

☐ Disable this task

JIRA instance\*

Local JIRA

Format\*

JUnit XML

Execution Report File (file path with file name)\*

java-junit-calc/target/surefire-reports/TI

Project Key

CALC

Test Execution Key

Test Plan Key

**\${bamboo.TESTPLAN}**

## 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 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 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 to 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=\${issueId}

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.IssueLinkManager
3 import com.atlassian.jira.issue.link.IssueLinkType
4 import com.atlassian.jira.issue.link.IssueLinkTypes
5 import com.atlassian.jira.ComponentManager
6 import com.atlassian.jira.component.ComponentAccessor
7 import com.atlassian.jira.jql.builder.JqlQueryBuilder
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.SearchService
12 import com.atlassian.jira.issue.search.SearchProvider
13 import com.atlassian.jira.issue.search.SearchResults
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.

Build projects / Xray Automation Samples / java-junit-calc-params

## Configuration - java-junit-calc-params

java-junit-calc with TESTLIST param

Plan Configuration

Stages & jobs 1

Default Stage

default

Branches 0

Plan details Stages Repositories Triggers Branches Dependencies Permissions Notifications Variables Miscellaneous Audit log

### Variables

Variables substitute values in your task configuration and inline scripts. If the key contains the phrase 'password', like 'userpassword' the value will be masked with \*\*\*\*\*.

For task configuration fields, use the syntax \${bamboo.myvariablename}. For inline scripts, variables are exposed as shell environment variables which can be accessed using the syntax \$bamboo\_MY\_VARIABLE\_NAME (Linux/Mac OS X) or %BAMBOO\_MY\_VARIABLE\_NAME% (Windows).

Variable name	Value
TESTLIST	
TESTPLAN	CALC-1200

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

Plan Configuration

Stages & jobs 1

Default Stage

default

Branches 0

Job details Tasks Requirements Artifacts Miscellaneous

### Tasks

A task is a piece of work that is being executed as part of the build. The execution of a script, a shell command, an Ant Task or a Maven goal are only about tasks.

You can use [runtime](#), [plan](#) and [global variables](#) to parameterize your tasks.

1 a

Source Code Checkout

Checkout Default Repository

Maven 3.x

clean compile test

Final tasks Are always executed even if a previous task fails

Xray: Results Import Task

Import JUnit XML

Add task

#### Maven 3.x configuration

Task description

clean compile test

☐ Disable this task

Executable

Maven 3 Add new executable

Goal\*

clean compile test -Dtest=\${bamboo.TESTLIST}

The goal you want to execute. You can also define system properties such as -Djava.awt.headless=true.

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

## Tasks

A task is a piece of work that is being executed as part of the build. The execution of a script, a shell command, an Ant Task or a Maven goal are [about tasks](#).

You can use [runtime](#), [plan](#) and [global variables](#) to parameterize your tasks.

Source Code Checkout  
Checkout Default Repository

Maven 3.x  
clean compile test

Final tasks Are always executed even if a previous task fails

Xray: Results Import Task  
Import JUnit XML

Add task

### Xray: Results Import Task configuration

Task description

Import JUnit XML

☐ Disable this task

JIRA instance\*

Local JIRA

Format\*

JUnit XML

Execution Report File (file path with file name)\*

java-junit-calc/target/surefire-reports/TI

Project Key

CALC

Test Execution Key

Test Plan Key

`\${bamboo.TESTPLAN}`

## 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"
```