How to manage attachments with public Java API

This section of our documentation refers to the public Java API available in Smart Attachments for Jira. Java API is available starting from **Smart Attachments for Jira 2.1.0.**

Please, note that Java API methods are not exposed in Smart Attachments versions older than 2.1.0. For **older versions** of the app, use **REST API** instea d.

In this section, you will find examples of the most common cases of usage of the public Java API available in Smart Attachments for Jira. If you need the whole Java API documentation for the app, click the button below.

Please note that we provide this public Java API to solve use cases not included in the app functionality available. Our support doesn't include help in writing scripts with the app public Java API. Our Support Team may assist with particular requests regarding writing scripts at our sole consent.

View Java API Documentation

Create a new category scheme with categories

This example illustrates creating a new category scheme with a pre-defined set of categories.

```
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.user.util.UserManager
import com.onresolve.scriptrunner.runner.ScriptRunnerImpl
import com.onresolve.scriptrunner.runner.customisers.WithPlugin
import com.stiltsoft.jira.attachcategory.facade.SmartAttachmentsFacade
import com.atlassian.jira.user.util.UserUtil

// Initializing the app components

@WithPlugin("com.stiltsoft.jira.smart-attachments")
SmartAttachmentsFacade facade = ScriptRunnerImpl.getPluginComponent(SmartAttachmentsFacade)
UserManager userManager = ComponentAccessor.getUserManager()

// Specifying the user on the behalf of who the scheme will be created
def currentUser = userManager.getUserByName("admin")

// Specifying the name of a new category scheme
def scheme = facade.createScheme("New Scheme", currentUser)

// Specifying names of categories within the category
def categoryDocuments = facade.generateNewConfigCategory("Documents")
def categoryScreenshots = facade.generateNewConfigCategory("Screenshots")
def categoryLogs = facade.generateNewConfigCategory("Logs")
def categoryPrototypes = facade.generateNewConfigCategory("Prototypes")

// Creating new categories within the category scheme
scheme.categories.add(categoryDocuments)
scheme.categories.add(categoryScreenshots)
scheme.categories.add(categoryLogs)
scheme.categories.add(categoryPrototypes)

// Updating the category scheme and saving modifications
facade.updateScheme(scheme, currentUser)
```

Update the configuration of the category scheme

This example illustrates the way to update the configuration of the category scheme (issue type visibility, access restrictions. auto-distribution rules).

```
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.config.IssueTypeManager
import com.atlassian.jira.security.groups.GroupManager
import com.atlassian.jira.user.util.UserManager
import com.atlassian.jira.security.roles.ProjectRoleManager
import com.onresolve.scriptrunner.runner.ScriptRunnerImpl
import com.onresolve.scriptrunner.runner.customisers.WithPlugin
import com.stiltsoft.jira.attachcategory.facade.SmartAttachmentsFacade
import com.stiltsoft.jira.attachcategory.facade.entity.config.AutoDistributionType

// Initializing the app components
@WithPlugin("com.stiltsoft.jira.smart-attachments")
SmartAttachmentsFacade facade = ScriptRunnerImpl.getPluginComponent(SmartAttachmentsFacade)
UserManager userManager = ComponentAccessor.getUserManager()
IssueTypeManager issueTypeManager = ComponentAccessor.getComponent(IssueTypeManager.class)
GroupManager groupManager = ComponentAccessor.getGroupManager()
ProjectRoleManager projectRoleManager = ComponentAccessor.getComponent(ProjectRoleManager.class)

// Specifying the user name and name of the scheme that will be updated
def currentUser = userManager.getUserByName("admin")
def optionalScheme = facade.getSchemeByName("New Scheme")

// Interrupting the script execution if the scheme with such name does not exist
if (!optionalScheme.isPresent()) {
    return
}

def scheme = optionalScheme.get()

// Specifying the category which configuration should be updated
def categoryDocuments = scheme.categories.find { category -> category.name.equalsIgnoreCase("Documents") }

// Specifying the issue type in which the category should be visible only
def taskIssueType = issueTypeManager.getIssueType("10100")
categoryDocuments.issueTypes.add(taskIssueType)

// Specifying user groups for access restrictions of the category
def softwareUserGroup = groupManager.getGroup("jira-software-users")
categoryDocuments.accessRestrictions.groups.add(softwareUserGroup)
categoryDocuments.accessRestrictions.users.add(currentUser)

// Specifying project roles for access restrictions of the category
def projectRole = projectRoleManager.getProjectRole("Administrators")
categoryDocuments.accessRestrictions.projectRoles.add(projectRole)

// Specifying the autodistribution rule that should be applied to the category
categoryDocuments.autoDistribution.type = AutoDistributionType.NAME_PATTERN
categoryDocuments.autoDistribution.value = ".doc, .pdf"

facade.updateScheme(scheme, currentUser)
```

Associate the category scheme with the project

This example illustrates the way to associate the project to the specific category scheme.

```
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.project.ProjectManager
import com.atlassian.jira.user.util.UserManager
import com.onresolve.scriptrunner.runner.ScriptRunnerImpl
import com.onresolve.scriptrunner.runner.customisers.WithPlugin
import com.stiltsoft.jira.attachcategory.facade.SmartAttachmentsFacade

// Initializing the app components
@WithPlugin("com.stiltsoft.jira.smart-attachments")
SmartAttachmentsFacade facade = ScriptRunnerImpl.getPluginComponent(SmartAttachmentsFacade)
UserManager userManager = ComponentAccessor.getUserManager()
ProjectManager projectManager = ComponentAccessor.getProjectManager()

// Specifyng the category scheme for association with the project
def currentUser = userManager.getUserByName("admin")
def optionalScheme = facade.getSchemeByName("New Scheme")

// Interrupting the script execution if the scheme with such name does not exist
if (!optionalScheme.isPresent()) {
    return
}

def scheme = optionalScheme.get()

// Specifying the project key which the category scheme is associated with
def project = projectManager.getProjectByCurrentKey("ONE")
scheme.projects.add(project)

facade.updateScheme(scheme, currentUser)
```

Deassociat the category scheme from the project

This example illustrates the way to remove the association of the project to the specific category scheme.

```
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.project.ProjectManager
import com.atlassian.jira.user.util.UserManager
import com.onresolve.scriptrunner.runner.ScriptRunnerImpl
import com.onresolve.scriptrunner.runner.customisers.WithPlugin
import com.stiltsoft.jira.attachcategory.facade.SmartAttachmentsFacade

// Initializing the app components
@WithPlugin("com.stiltsoft.jira.smart-attachments")
SmartAttachmentsFacade facade = ScriptRunnerImpl.getPluginComponent(SmartAttachmentsFacade)
UserManager userManager = ComponentAccessor.getUserManager()
ProjectManager projectManager = ComponentAccessor.getProjectManager()

// Specifying the category scheme for association with the project
def currentUser = userManager.getUserByName("admin")
def optionalScheme = facade.getSchemeByName("New Scheme")

// Interrupting the script execution if the scheme with such name does not exist
if (!optionalScheme.isPresent()) {
    return
}

def scheme = optionalScheme.get()

// Specifying the project key which the category scheme is associated with
def project = projectManager.getProjectByCurrentKey("ONE")
scheme.projects.remove(project)

facade.updateScheme(scheme, currentUser)
```

Move attachments between categories

This example illustrates moving attachments from one category to another within the same issue.

```
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.issue.IssueManager
import com.atlassian.jira.user.util.UserManager
import com.onresolve.scriptrunner.runner.ScriptRunnerImpl
import com.onresolve.scriptrunner.runner.customisers.WithPlugin
import com.stiltsoft.jira.attachcategory.facade.SmartAttachmentsFacade
import com.stiltsoft.jira.attachcategory.facade.entity.issue.AttachmentCategories

// Initializing the app components
@WithPlugin("com.stiltsoft.jira.smart-attachments")
SmartAttachmentsFacade facade = ScriptRunnerImpl.getPluginComponent(SmartAttachmentsFacade)
UserManager userManager = ComponentAccessor.getUserManager()
IssueManager issueManager = ComponentAccessor.getIssueManager()

def currentUser = userManager.getUserByName("admin")

// Specifying the issue which contains files for moving
def issue = issueManager.getIssueObject("ONE-1")

AttachmentCategories attachmentCategories = facade.getAttachmentCategories(issue)

// Specifying the source and target categories for moving attachments
def categoryDocuments = attachmentCategories.categories.find { category -> category.name.equalsIgnoreCase
("Documents") }
def categoryScreenshots = attachmentCategories.categories.find { category -> category.name.equalsIgnoreCase
("Screenshots") }

// Specifying the attachment name
def attachments = new ArrayList()
def attachment = categoryDocuments.getAttachments(false).find { attachment -> attachment.filename.
equalsIgnoreCase("image.png") }
attachments.add(attachment)

facade.addAttachmentsToCategory(issue, categoryScreenshots.id, attachments, currentUser)
```

Copy an issue with attachments in categories

This example illustrates the way to copy an issue with attachments and preserve attachments in their categories.

```
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.issue.AttachmentManager
import com.atlassian.jira.issue.IssueFactory
import com.atlassian.jira.issue.IssueManager
import com.atlassian.jira.issue.attachment.Attachment
import com.atlassian.jira.user.util.UserManager
import com.onresolve.scriptrunner.runner.ScriptRunnerImpl
import com.onresolve.scriptrunner.runner.customisers.WithPlugin
import com.stiltsoft.jira.attachcategory.facade.SmartAttachmentsFacade

// Initializing the app components
@WithPlugin("com.stiltsoft.jira.smart-attachments")
SmartAttachmentsFacade facade = ScriptRunnerImpl.getPluginComponent(SmartAttachmentsFacade)
UserManager userManager = ComponentAccessor.getUserManager()
IssueFactory issueFactory = ComponentAccessor.getIssueFactory()
IssueManager issueManager = ComponentAccessor.getIssueManager()
AttachmentManager attachmentManager = ComponentAccessor.getAttachmentManager()

// Specifying the user and issue key
def currentUser = userManager.getUserByName("admin")
def sourceIssue = issueManager.getIssueObject("ONE-1")

// Copying an issue and attachments within the same project
def issueCopy = issueFactory.cloneIssue(sourceIssue)
Map<String, Object> newIssueParams = ["issue": issueCopy] as Map<String, Object>
def clonedIssue = issueManager.createIssueObject(currentUser, newIssueParams)
attachmentManager.copyAttachments(sourceIssue, currentUser, clonedIssue.key)


def sourceAttachmentCategories = facade.getAttachmentCategories(sourceIssue)

List<Attachment> attachmentsFromClonedIssue = new ArrayList<>(attachmentManager.getAttachments(clonedIssue))

sourceAttachmentCategories.categories.each({
    category -> category.getAttachments(false).each({
        attachment ->
            def attachmentCopy = attachmentsFromClonedIssue.find({clonedAttachment -> clonedAttachment.filename.
equalsIgnoreCase(attachment.filename)})
            facade.addAttachmentsToCategory(clonedIssue, category.id, Collections.singletonList
(attachmentCopy), currentUser)
            attachmentsFromClonedIssue.remove(attachmentCopy)
    })
})
```

Copy an issue with attachments in categories to another project

This example illustrates the way to copy an issue with attachments in categories to another project with preserving files in categories as in the source issue.

This script will work only in projects that are associated with the same scheme.

```
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.issue.AttachmentManager
import com.atlassian.jira.issue.IssueFactory
import com.atlassian.jira.issue.IssueManager
import com.atlassian.jira.issue.attachment.Attachment
import com.atlassian.jira.project.ProjectManager
import com.atlassian.jira.user.util.UserManager
import com.onresolve.scriptrunner.runner.ScriptRunnerImpl
import com.onresolve.scriptrunner.runner.customisers.WithPlugin
import com.stiltsoft.jira.attachcategory.facade.SmartAttachmentsFacade

// Initializing the app components
@WithPlugin("com.stiltsoft.jira.smart-attachments")
SmartAttachmentsFacade facade = ScriptRunnerImpl.getPluginComponent(SmartAttachmentsFacade)
UserManager userManager = ComponentAccessor.getUserManager()
IssueFactory issueFactory = ComponentAccessor.getIssueFactory()
IssueManager issueManager = ComponentAccessor.getIssueManager()
AttachmentManager attachmentManager = ComponentAccessor.getAttachmentManager()
ProjectManager projectManager = ComponentAccessor.getProjectManager()

// Specifying the user, issue key and key of the target project
def currentUser = userManager.getUserByName("admin")
def sourceIssue = issueManager.getIssueObject("ONE-1")
def targetProject = projectManager.getProjectByCurrentKey("TWO")

// Copying an issue with attachments to another project
def issueCopy = issueFactory.cloneIssue(sourceIssue)
issueCopy.setProjectObject(targetProject)
Map<String, Object> newIssueParams = ["issue": issueCopy] as Map<String, Object>
def clonedIssue = issueManager.createIssueObject(currentUser, newIssueParams)
attachmentManager.copyAttachments(sourceIssue, currentUser, clonedIssue.key)


def sourceAttachmentCategories = facade.getAttachmentCategories(sourceIssue)

List<Attachment> attachmentsFromClonedIssue = new ArrayList<>(attachmentManager.getAttachments(clonedIssue))

sourceAttachmentCategories.categories.each({
    category -> category.getAttachments(false).each({
        attachment ->
            def attachmentCopy = attachmentsFromClonedIssue.find({clonedAttachment -> clonedAttachment.filename.
equalsIgnoreCase(attachment.filename)})
            facade.addAttachmentsToCategory(clonedIssue, category.id, Collections.singletonList
(attachmentCopy), currentUser)
            attachmentsFromClonedIssue.remove(attachmentCopy)
    })
})
```

Check for the presence of attachments in a certain category depending on the custom field value

This example illustrates how to check for the presence of attachments in a certain category depending on the custom field value.

```
import com.opensymphony.workflow.InvalidInputException
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.issue.CustomFieldManager
import com.atlassian.jira.issue.customfields.option.Option
import com.atlassian.jira.issue.fields.CustomField
import com.onresolve.scriptrunner.runner.ScriptRunnerImpl
import com.onresolve.scriptrunner.runner.customisers.WithPlugin
import com.stiltsoft.jira.attachcategory.facade.SmartAttachmentsFacade
import com.stiltsoft.jira.attachcategory.facade.entity.issue.AttachmentCategories
import com.stiltsoft.jira.attachcategory.facade.entity.issue.AttachmentCategory

@WithPlugin("com.stiltsoft.jira.smart-attachments")
SmartAttachmentsFacade facade = ScriptRunnerImpl.getPluginComponent(SmartAttachmentsFacade)

CustomFieldManager customFieldManager = ComponentAccessor.getCustomFieldManager()

CustomField customfield_10600 = customFieldManager.getCustomFieldObject("customfield_10600")
Option customFieldValue = (Option) issue.getCustomFieldValue(customfield_10600)
String customFieldValueRaw = customFieldValue.getValue()

if (customFieldValueRaw.equals("Server")) {

    AttachmentCategories attachmentCategories = facade.getAttachmentCategories(issue)
    AttachmentCategory serverCategory = attachmentCategories.getCategories().find { category -> category.
getName() == "Server" }

    def tempAttachmentsCount = serverCategory.getTempAttachments().size()
    def actualAttachmentsCount = serverCategory.getAttachments(true).size()

    if (tempAttachmentsCount == 0 && actualAttachmentsCount == 0) {
        throw new InvalidInputException("customfield_10600", "The Server category is empty")
    }
}
```

Get the attachment comment information



The attachment commenting feature has been deprecated in Smart Attachments 2.15.2.

This example illustrates how to get the information about the attachment comments (including the author, comment text, and creation time).

```
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.issue.IssueManager
import com.onresolve.scriptrunner.runner.ScriptRunnerImpl
import com.onresolve.scriptrunner.runner.customisers.WithPlugin
import com.stiltsoft.jira.attachcategory.facade.SmartAttachmentsFacade
import com.stiltsoft.jira.attachcategory.facade.entity.issue.AttachmentCategories
import com.atlassian.jira.issue.Issue

@WithPlugin("com.stiltsoft.jira.smart-attachments")
SmartAttachmentsFacade facade = ScriptRunnerImpl.getPluginComponent(SmartAttachmentsFacade)
IssueManager issueManager = ComponentAccessor.getIssueManager()

Issue issue = issueManager.getIssueObject("CVT-1")

issue.attachments.each{ attachment ->
    facade.getCommentThreadsForAttachment(attachment).each{ thread ->
            log.info "Thread ID: $thread.threadId"
        thread.attachmentComments.each{ attachmentComment ->
            log.info "Comment ID: $attachmentComment.commentId, Author: $attachmentComment.author.displayName,
Content: $attachmentComment.content, Created: $attachmentComment.createTime"
        }
    }
}
```

Get the list of projects with Smart Attachments for Jira

This example illustrates getting information about the Jira projects with the Smart Attachments category scheme activated. The result will bring you the list of all projects in your instance, including the archived ones where the category scheme has been activated.

```
import com.onresolve.scriptrunner.runner.ScriptRunnerImpl
import com.onresolve.scriptrunner.runner.customisers.WithPlugin
import com.stiltsoft.jira.attachcategory.facade.SmartAttachmentsFacade

// Initializing the app components

@WithPlugin("com.stiltsoft.jira.smart-attachments")
SmartAttachmentsFacade facade = ScriptRunnerImpl.getPluginComponent(SmartAttachmentsFacade)

// Getting list of projects
def projectsList = facade.getSchemes().collect { it.getProjects() }.flatten()
def keys = projectsList.collect { it.getKey() }

log.warn(keys)
projectsList.forEach { log.warn(it.getKey() +  " - " + it.getName()) }
```

Copy all the attachments from one issue and add them to a linked issue

This example illustrates copying all the attachments from one issue and adding them to a linked issue with preserving the attachment category and revisions.

Please note that this script has two variables. These are "MIG-789" and "MIG-4005". You will need to replace these values with the following issue keys from your Jira projects:

- use the issue key of the source issue (from which you are copying) instead of "MIG-789"
- use the issue key of the target issue (to which you are copying) instead of "MIG-4005"

```
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.issue.AttachmentManager
import com.atlassian.jira.issue.IssueFactory
import com.atlassian.jira.issue.IssueManager
import com.atlassian.jira.issue.attachment.Attachment
import com.atlassian.jira.user.util.UserManager
import com.onresolve.scriptrunner.runner.ScriptRunnerImpl
import com.onresolve.scriptrunner.runner.customisers.WithPlugin
import com.stiltsoft.jira.attachcategory.facade.SmartAttachmentsFacade

// Initializing the app components
```

```
@WithPlugin("com.stiltsoft.jira.smart-attachments")
SmartAttachmentsFacade facade = ScriptRunnerImpl.getPluginComponent(SmartAttachmentsFacade)
UserManager userManager = ComponentAccessor.getUserManager()
IssueFactory issueFactory = ComponentAccessor.getIssueFactory()
IssueManager issueManager = ComponentAccessor.getIssueManager()
AttachmentManager attachmentManager = ComponentAccessor.getAttachmentManager()

// Specifying the user and source issue key, target issue key
def currentUser = userManager.getUserByName("admin")
def sourceIssue = issueManager.getIssueObject("MIG-789")
def targetIssue = issueManager.getIssueObject("MIG-4005")

// Copying the source issue attachments to the target issue within the same project
List<Attachment> attachmentsFromSourceIssue = new ArrayList<>(attachmentManager.getAttachments(sourceIssue))
List<Attachment> attachmentsFromTargetIssue = new ArrayList<>(attachmentManager.getAttachments(targetIssue))

attachmentsFromSourceIssue.each {
        sourceAttachment ->
            def attachmentTarget = attachmentsFromTargetIssue.find { it.filename.equalsIgnoreCase
(sourceAttachment.filename) }
            if (null == attachmentTarget) {
                attachmentManager.copyAttachment(sourceAttachment, currentUser, targetIssue.key)
                attachmentsFromTargetIssue.add(sourceAttachment)
            }
    }

def sourceAttachmentCategories = facade.getAttachmentCategories(sourceIssue)
def targetAttachmentCategories = facade.getAttachmentCategories(targetIssue)

attachmentsFromTargetIssue = new ArrayList<>(attachmentManager.getAttachments(targetIssue))

sourceAttachmentCategories.categories.each { category ->

    // Copying all attachments, including existing revisions within documents
    category.getAttachments(true).each {
        sourceAttachment ->
            def attachmentCopy = attachmentsFromTargetIssue.find({ it.filename.equalsIgnoreCase
(sourceAttachment.filename) })
            facade.addAttachmentsToCategory(targetIssue, category.id, Collections.singletonList
(attachmentCopy), currentUser)
            attachmentsFromTargetIssue.remove(attachmentCopy)
    }

    // Adding revisions to existing documents or to newly created ones
    category.getDocuments().each { sourceDocument ->
        attachmentsFromTargetIssue = new ArrayList<>(attachmentManager.getAttachments(targetIssue))
        if (!targetAttachmentCategories.categories.any { it.documents.any { it.name.equals(sourceDocument.name)
} }) {
            def sourceDocumentRevisions = sourceDocument.getAttachments()
            def targetDocumentRevisions = attachmentsFromTargetIssue.findAll { targetAttachment ->
                sourceDocumentRevisions.any { it.filename.equalsIgnoreCase(targetAttachment.filename) }
            }
            facade.createDocument(targetIssue, category.id, sourceDocument.name, targetDocumentRevisions,
currentUser)
        }
        else {
            sourceDocument.getAttachments().each { sourceAttachment ->
                def targetCategory = targetAttachmentCategories.categories.find({it.name.equalsIgnoreCase
(category.name)})
                def targetDocument = targetCategory?.documents.find({ it.name.equalsIgnoreCase(sourceDocument.
name) })
                if (!targetDocument?.getAttachments().any( {it.filename.equalsIgnoreCase(sourceAttachment.
filename)})) {
                    def attachmentCopy = attachmentsFromTargetIssue.find({ it.filename.equalsIgnoreCase
(sourceAttachment.filename) })
                    facade.addDocumentRevision(targetIssue, targetDocument.id, attachmentCopy, currentUser)
                    attachmentsFromTargetIssue.remove(attachmentCopy)
                }
            }
        }
    }
```

```
}
```