ScriptRunner IT Service Management (ITSM)

Lab 3

October 2018

Table of Contents

ScriptRunner IT Service Management (ITSM)

Overview

This lab brings together ScriptRunner features that allow ITSM practices such as updating all Linked incidents, count alerts using Scripted Fields, Using PostFunctions to set Remedial Action Priority, Optimising the Jira Service Desk behaviour, and constraining the Create Issue dialog with customised rules.

What do these labs showcase?

- How ScriptRunner enables a one-stop shop for updating all linked incidents in one go, saving time and cost for IT teams by automating these tasks and provide time to focus on more critical issues
- Using ScriptRunner Behaviours, you can now control the user interaction in a Jira Service Desk form, enabling IT teams to decipher key information by controlling what input the user must provide in a support desk form
- Using the power of Behaviours again, we see how to constrain the Create Issue dialog in Jira, providing more control over how your Jira users interact with the form
- Supercharging custom fields to show the number of alerts in an issue, users will learn how powerful Scripted Fields is with this example
- Tailor your Jira workflow to put in place a post function that sets a remedial action priority, for example, automatically set the remedial action issue priority to 'Highest' if it has 2 or more linked alerts.

Technology used

The following technology components are used in this solution:

- ScriptRunner for Jira Server - 5.4.12
    - Scripted Fields
    - Post Functions
    - Behaviours
    - Script Listener
- Jira Service Desk - 3.8.1
- Jira Software - 7.5.0
- Jira Core - 7.5.0

Prerequisites

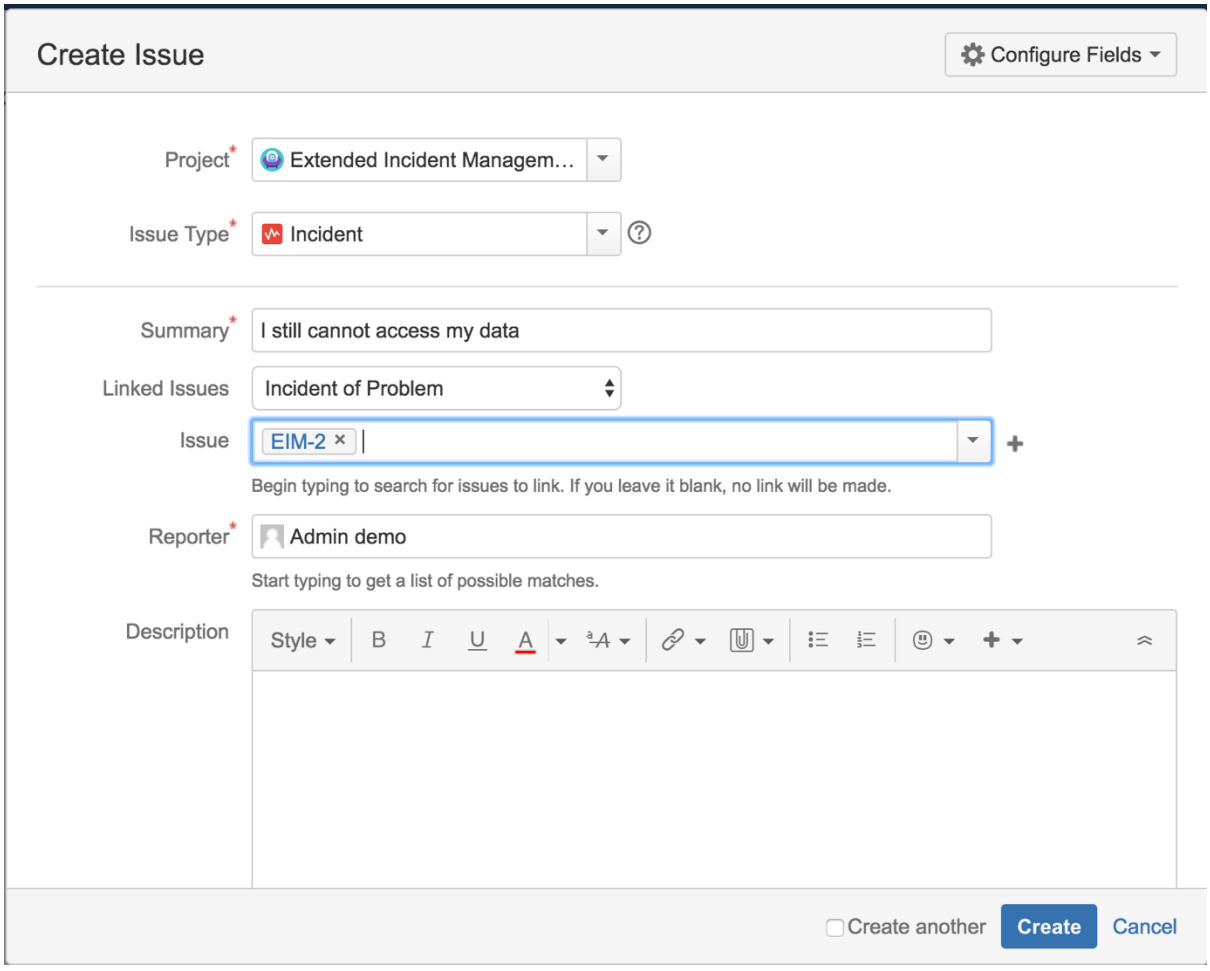
- A demo/magic button Jira with Service Desk and Software license
- A demo/magic administration account
- A ScriptRunner for Jira Server license (evaluation or commercial):
- A basic understanding of Jira

Lab 3 – Script Listener for Updating All Linked Incidents

Before you start

Use the normal Create button in the banner menu to create another Incident issue and link it to the Problem issue created in Lab 2.

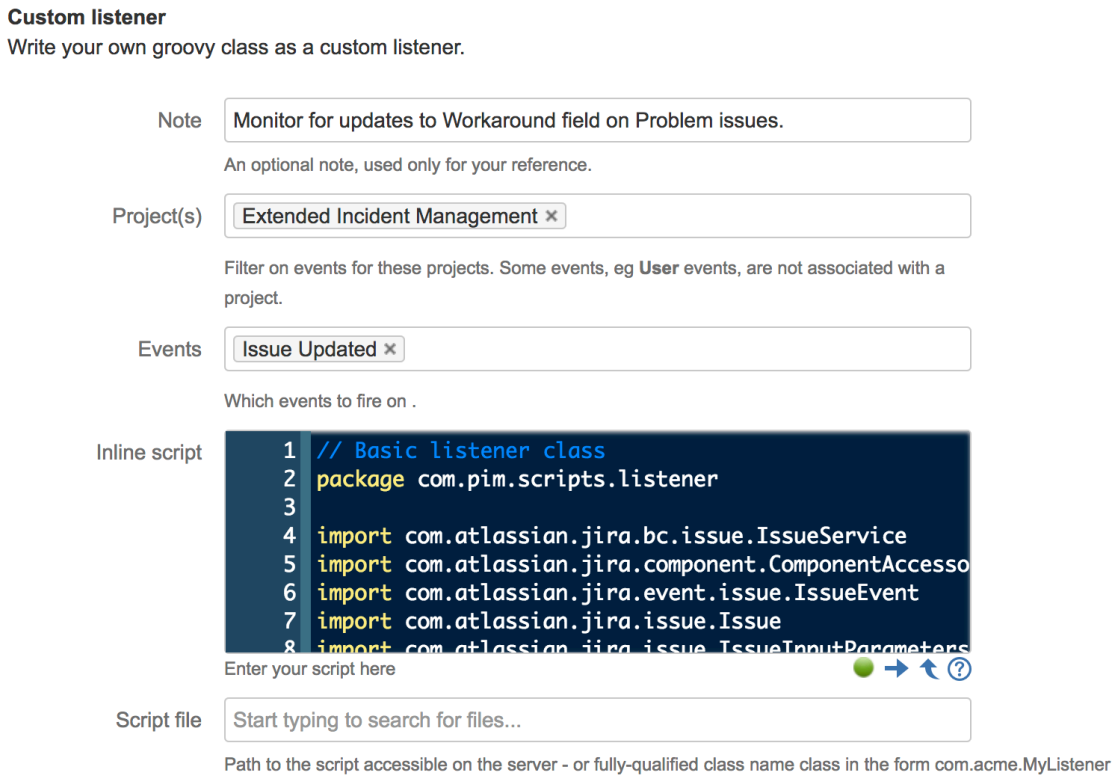
- Issue Type → Incident
- Summary → 'I still cant access my data' (or something similar to whatever you chose to put)
- Linked Issues - Incident of Problem
- Issue → EIM-2 (or whatever the Problem issue is that you created earlier)



You will now have one Problem issue with two Incident issues linked to it.

Step 1

- Go to Add-ons → Script Listeners using the cog in the top right corner
- Then click the Add New Item button and choose Custom listener
- Configure the screen to look like the below:



- Here is the code snippet:

```
// Basic listener class
package com.pim.scripts.listener

import com.atlassian.jira.bc.issue.IssueService
import com.atlassian.jira.component.ComponentAccessor
import com.atlassian.jira.event.issue.IssueEvent
import com.atlassian.jira.issue.Issue
import com.atlassian.jira.issue.IssueInputParameters
import com.atlassian.jira.issue.changehistory.ChangeHistoryManager
import com.atlassian.jira.issue.link.IssueLink
import com.atlassian.jira.user.ApplicationUser

// Imports
import org.apache.log4j.Level
import org.apache.log4j.Logger
import org.ofbiz.core.entity.GenericValue

// Wrapper class for script
class WorkaroundUpdate extends Script {

    // Script code for easy log identification
    public static String scriptCode = "Workaround Update Listener:"
```

```
    // The name of the workaround field
    public static String workaroundFieldName = "Workaround"

    // List of link types we are counting for Incidents
    public static List<String> linkNamesForIncident = [ "Incident Link" ]

    // List of issue types we are counting for Incidents
    public static List<String> IncidentIssueNames = ["Incident"]

    // Username of user to execute automated field update
    public static String adminUsername = "admin"

    // Method to get change items from an event
    List<GenericValue> getEventChangeHistory( IssueEvent event, Logger logger ) {

        // Get the things we need
        ChangeHistoryManager changeHistoryManager =
ComponentAccessor.getChangeHistoryManager()
        List<GenericValue> changeItems = []

        // Try to get the history
        try {
            Long eventID = event.getChangeLog().getLong( "id" )
            changeItems =
changeHistoryManager.getChangeHistoryById(eventID).getChangeItems()
            logger.debug( "$scriptCode Getting change history for event ID
$eventID" )

        }
        catch ( all ) {
            changeItems = null
            logger.debug( "$scriptCode Failed to get change history passed object"
)
        }

        // Return the result
        return changeItems
    }

    // Update a single custom field value for a given issue ID
    def singleCustomFieldValueUpdate( Issue issue, String newValue, String
customFieldName, ApplicationUser user, Logger logger ) {

        // Managers needed
        IssueService issueService = ComponentAccessor.getIssueService()
        logger.setLevel( Level.ALL )
        // Get some issue details
        String customFieldRef = ComponentAccessor.getCustomFieldManager(
).getCustomFieldObjectByName( customFieldName ).getId(  )
        logger.debug( "$scriptCode got custom field ref $customFieldRef" )
        IssueService.IssueResult issueResult = issueService.getIssue( user,
issue.getId() )

        // Check if the issue result is valid
        if ( issueResult == null ) {

            // Some sort of user error, cannot update issue
            logger.debug( "$scriptCode Update attempt on issue ${issue.getKey()}
```

```
        resulted in error from null issue result" )

                } else {

                    // Result is valid, get the issue input parameters
                    def IssueInputParameters issueInputParameters =
        issueService.newIssueInputParameters()
                    issueInputParameters.setRetainExistingValuesWhenParameterNotProvided(
        true, true )
                    issueInputParameters.addCustomFieldValue( customFieldRef, newValue )

                    // Make the updates
                    def IssueService.UpdateValidationResult result =
        issueService.validateUpdate( user, issue.getId(), issueInputParameters )

                    // Check for any errors in the result
                    if ( result.getErrorCollection().hasAnyErrors() ) {

                        // Deal with errors
                        logger.debug( "$scriptCode issue result had errors, stop update,
        errors are" + result.errorCollection )
                        return

                    } else {

                        // No errors, make the update
                        issueService.update( user, result )
                        logger.debug( "$scriptCode Issue ${issue.getKey()} value of custom
        field $customFieldName updated to include $newValue" )
                        return

                    }
                }
            }

            // Run method of script
            def run() {

                // Setup the log and leave a message to show what we're doing
                Logger logger = log
                logger.setLevel( Level.ALL )

                // Get the event and issue
                IssueEvent triggerEvent = event
                Issue triggerIssue = triggerEvent.getIssue()
                logger.debug( "$scriptCode Script triggered on issue
        ${triggerIssue.getKey()}" )

                // Get the issue change history
                List<GenericValue> changeHistory = getEventChangeHistory( triggerEvent,
        logger )

                // String to hold our potential new value
                String newWorkaround = null

                // Iterate over the change items to see if they are relevant
                for ( GenericValue changeItem in changeHistory ) {

                    // Is this change item related to the workaround field?
```

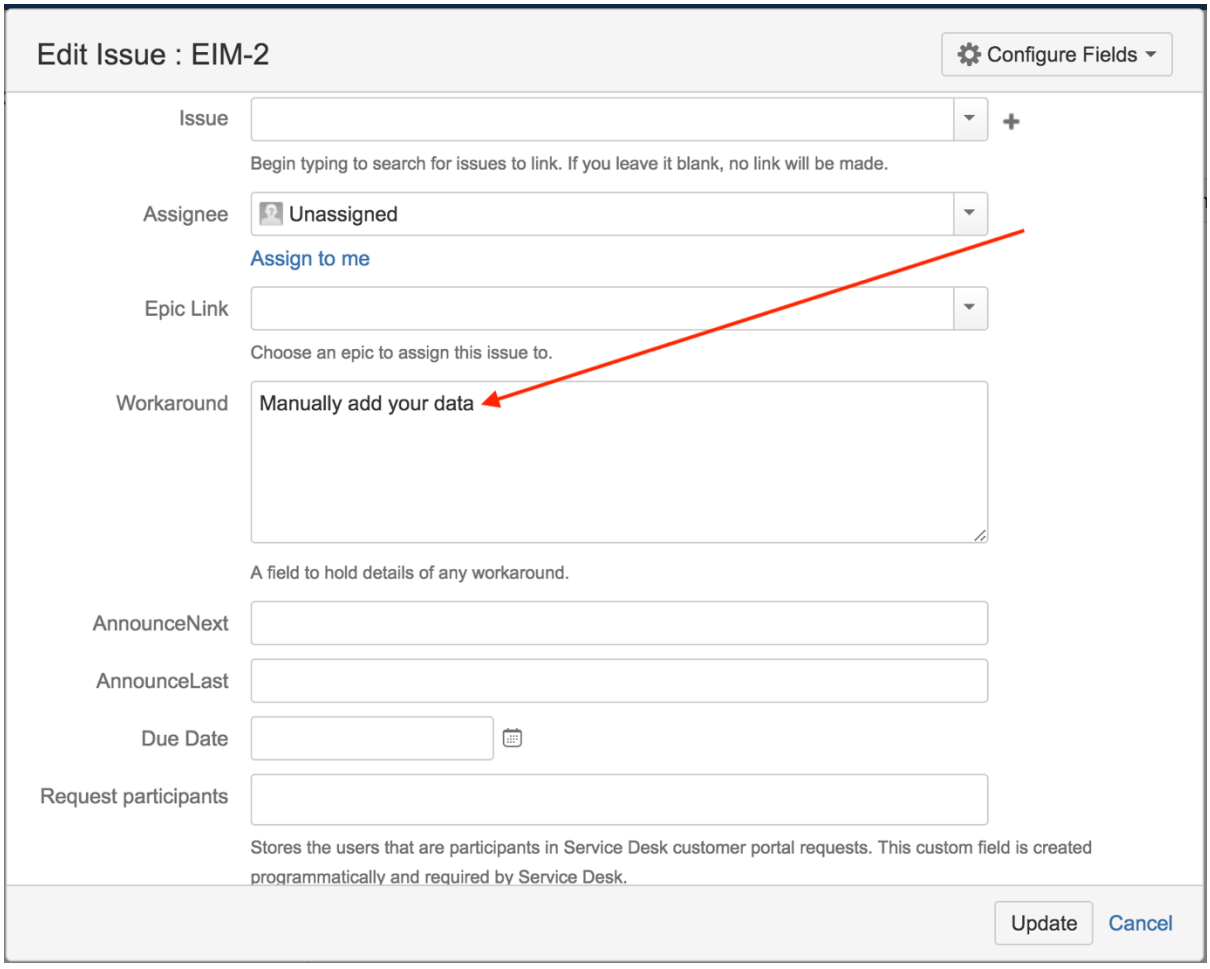
```
                if (changeItem["field"].toString(  ) == workaroundFieldName) {

                    // Yes, so get the new field value
                    newWorkaround = changeItem["newstring"]
                }
            }

            // Did we find a new value of the workaround field?
            if ( newWorkaround ) {

                // Yes, update the linked Incidents
                logger.debug( "$scriptCode New value of workaround field on issue
${triggerIssue.getKey(  )} is '$newWorkaround', updating linked Incidents" )

                // Get all the trigger issue links
                List<IssueLink> triggerIssueInwardLinks =
ComponentAccessor.getIssueLinkManager().getInwardLinks( triggerIssue.getId() )
                logger.debug( "$scriptCode Got ${triggerIssueInwardLinks.size(  )} all
linked issues" )
                 for ( IssueLink currentLink in triggerIssueInwardLinks ) {
                         logger.debug( "$scriptCode Found link
${currentLink.getIssueLinkType().getName()} link" )
                     }
                // Get all the link types that we are looking for
                List<IssueLink> filteredLinks = triggerIssueInwardLinks.findAll {
it.getIssueLinkType().getName() in linkNamesForIncident }
                logger.debug( "$scriptCode Got ${filteredLinks.size(  )} linked issues
where link is in $linkNamesForIncident" )

                // Iterate over those links
                for ( IssueLink currentLink in filteredLinks ) {

                    // Get the source issue
                    Issue currentIssue = currentLink.getSourceObject(  )
                    logger.debug( "$scriptCode Checking that linked issue
${currentIssue.getKey(  )} is one of the following issue types:
$IncidentIssueNames" )

                    // Check that the issue type is correct
                    if ( currentIssue.getIssueType(  ).getName() in IncidentIssueNames
) {

                        // Yes, correct issue type, update the issue in question

                       String oldValue = currentIssue.getCustomFieldValue(
ComponentAccessor.getCustomFieldManager().getCustomFieldObjectByName(workaroundFie
ldName))?:""
                      String newValue = triggerIssue.getKey() + " " +newWorkaround +
"["+new Date().format('dd/MMM/yyyy HH:mm') +"]\n" + oldValue

                       singleCustomFieldValueUpdate( currentIssue, newValue,
workaroundFieldName, ComponentAccessor.getUserManager(  ).getUserByName(
adminUsername ), logger )
                       logger.debug("$scriptCode Updated workaround field on issue
${currentIssue.getKey()} to '$newWorkaround' due to updated value on parent issue
${triggerIssue.getKey(  )}")
                    }
                }
            } else {
```

```
                // No new workaround
                logger.debug( "$scriptCode No new workaround value found for issue
${triggerIssue.getKey(  )}, not updating linked Incidents" )
            }
        }
}
```

- Click Add

Test Step

- Go to the Problem Issue that was created in Lab 2
- Click on the edit button to open the edit issue dialog



- Find the Workaround field and enter some information. Make it something that replicates a real-life workaround. Such as 'Manually add your data to the admin section'
- Click Update
- Now go to the Incident issues created here and in Lab 2 (the ones linked to the Problem issue) and check that the Workaround field for these has been updated

EIM-4 Return to queue

I still cant access my data

Edit | Comment | Assign | More | Link to Known Problem | Backlog | Workflow | Admin | Export

Details

Type:	Incident	Status:	BACKLOG
Priority:	Medium		(View Workflow)
Component/s:	None	Resolution:	Unresolved
Labels:	None		
Workaround:	EIM-2 Manually add your data[21/Sep/2018 13:27]		

Description

Click to add description

Attachments

SLAs

0min Time to resolution within 4h

People

Assignee: Unassigned
Assign to me

Reporter: Admin demo

Request participants: None